2018 年臺灣國際科學展覽會
優勝作品專輯

作品編號　190003

參展科別　電腦科學與資訊工程

作品名稱　基於睡眠時期大腦活動之概念的類神經網路優化法研究

得獎獎項　大會獎：二等獎

英特爾電腦科學獎

加拿大科學展覽會 CWSF 正選代表

就讀學校　臺北市立建國高級中學

指導教師　許雅淳

作者姓名　吳東昱、葉彥辰

關鍵詞　加速進程、類神經網路、權重更新

作者簡介

我是就讀於建國中學二年級的吳東昱（左），上高中後便一頭栽入機器學習的世界裡。而演算法、數學模型是機器學習的核心，也是廣泛應用背後最根本的推手，因此我一直醉心於其中，尤其是被稱為「黑盒子」的類神經網路。研究時繁複的公式與數學推導、問題層出不窮的數據分析雖然令人頭痛，但作為一名被深深吸引的研究者，我感到很開心。

大家好，我是就讀建國中學二年級的葉彥辰（右）。自國中參加程式設計課程，我就對資訊秉持著極為濃厚的興趣，接觸機器學習領域後，更是深深為之著迷。在研究的過程中，即使每每遭遇挫折，但在家人、師長、朋友的支持下，終能一一克服。由衷感謝一路上相伴的所有人，也希望自己未來能保持著熱忱，享受探索的樂趣。

摘要

深度學習透過迭代的訓練，如沿著梯度反方向更新權重的梯度下降法，旨在找出損失函數建構出的多維函數圖形中，其全局最小解的變數組合。我們受到在睡眠中的快速動眼期啟發，此時期腦中的高頻率 γ 波可以強化學習效果。本研究模擬此現象，提出一使原損失函數擁有自適應的增幅功能的演算法（稱之為 REM 方法），且其中的超參數能夠根據其應用調整。

本研究將 REM 方法應用於三種經典的優化法，並且以五種異質的資料集測試。實驗結果指出，在搭配隨機梯度下降法（SGD）與自適應學習率優化法（Adagrad）時，REM 有顯著的優化效果。REM 方法不僅能大幅加速訓練進程，亦能避免特定的訓練問題。

Abstract

Deep learning, through iterative training, aims to seek the global minimum loss function value in multidimensional graph plotted against many variables. Our paradigm is inspired by the high-frequency gamma wave in Rapid Eye Movement Sleep (REMS), which maximizes the learning effects. Hence, we developed an algorithm termed REM that mimics the phenomenon. It adaptively modifies the loss function with flexible hyperparameters that can be adjusted according to its application.

In this paper, the REM algorithm is applied to three classic optimizers and tested in five heterogeneous datasets. The results indicate that the effect of the REM algorithm is significantly beneficial when applied to Stochastic Gradient Descent (SGD) or Adaptive Gradient Algorithm (Adagrad). Not only can the REM algorithm substantially accelerate the training process, but it can also prevent certain training problems.

一、 前言

（一） 研究動機

自類神經網路（neural network）發展出隱藏層（hidden layer），得以解決異或問題（XOR problem）後，經歷數十年的發展，已漸漸成為人類實際生活上、其他領域之學術研究上的一大利器。如 Hewitson, B.等（1994）的研究即闡述了類神經網路在地理學上的應用及幫助[6]。而為了讓類神經網路的性能更好及解決使用時可能遇到的問題，相關演算法的研究也從未停歇。

至今已有各種優秀的權重優化法和特殊的機制來提升類神經網路訓練的品質。以機制為例，有對統計資料進行預處理的標準化（normalization）、防止過擬合現象（overfitting）的正則化（regularization）、適當終止模型訓練的提前停止（early-stopping）、學習率的衰減機制（learning rate decay）等等。

在這個領域的研究，人類尤其發揮其善於聯想、模擬的思維。像是基於傳統梯度下降法，加入物理中的「慣性」概念而創造出的動量法（momentum），就可以用於解決局部最小值（local minimum）的問題；因此，本研究旨在深入探討類神經網路的數學原理，改善既有的演算法，對深度學習領域的發展做出貢獻。

我們從生物睡眠期間快速動眼期（以下簡稱 REM）的腦波圖得到啟發，將該機制的現象、功用應用於類神經網路中的倒傳遞算法（此處特指損失函數與梯度下降法），提出自適應損失函數的概念，期望增進類神經網路的訓練速度及解決訓練中可能遇到的問題。

（二） 研究目的

透過數學公式推導及實驗數據分析，驗證本研究提出的假說：以 REM 時期的大腦活動為藍本，將既有損失函數的變形，提升在訓練週期（epoch）中，類神經網路訓練的效果。以下為期望達到的研究目標：

1. 以損失函數之值、訓練週期、指數衰減等概念為基礎，建構出一套輔助原有的權重更新演算法的數學模型。
2. 比較有無此數學模型的相同程式，其準確度的差別，並以不同的資料集測試之。

3. 分析所得數據，評估該數學模型對不同特性之資料集的影響。
4. 將該數學模型搭配其他優化法（如 momentum）與機制（如學習率衰減），測試它們之間的交互影響並分析之。
5. 使用不同的類神經網路模型測試此數學模型，並針對不同模型做適當的變形。

二、 研究方法與過程

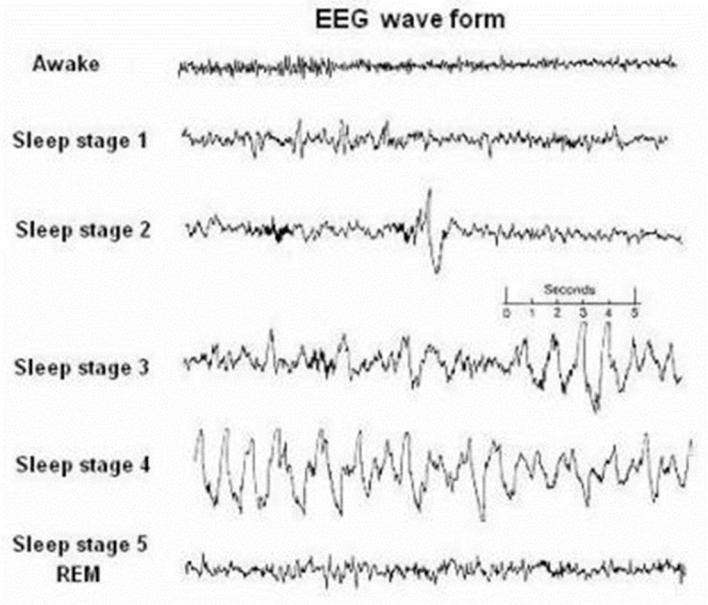
（一） 睡眠相關的文獻探討

1. 睡眠的各個階段

本研究主軸是由人類的睡眠機制得到啟發。為了闡述該發想與研究的設計，故先介紹與本研究相關的睡眠機制。睡眠中各個階段的腦部活動不同，腦波型態也相異：

（表 1 各睡眠階段的腦波型態與腦部活動）

階段名稱	腦波型態	腦部活動
清醒	α 波與 β 波	意識完全清楚
第一階段	由 α 波逐漸轉為 θ 波	意識仍然相對清楚
第二階段	主要為 θ 波，偶爾會出現睡眠紡錘波（sleep spindle）及 K 複合波（K-complex）	意識逐漸消失，但仍容易感受到外界刺激
慢波睡眠（slow-wave sleep, SWS）	δ 波（慢速、高震幅）	腦部血流顯著減少，不易察覺到外界刺激
快速動眼期（rapid eye movement, REM）	γ 波（高頻率波）	腦部相當活躍，較易察覺外界刺激



（圖 1 各睡眠階段的腦波圖）

2. 睡眠與記憶的關係

腦部在處理記憶時，會依序進行一系列的程序：登錄（encoding）、儲存（storage）、提取記憶（retrieval）。新記憶會先在海馬迴（hippocampus）中暫存，接著經過記憶鞏固（memory consolidation）轉為長期記憶（long-term memory）。尤其在快速動眼期（rapid eye movement，以下簡稱 REM）的睡眠，可以幫助腦部回顧記憶，能鞏固並且整合新的記憶。另外，神經元間的拮抗作用，將影響非快速動眼期（NREM）與快速動眼期（REM）之間的切換，以完成一個個完整的睡眠週期。

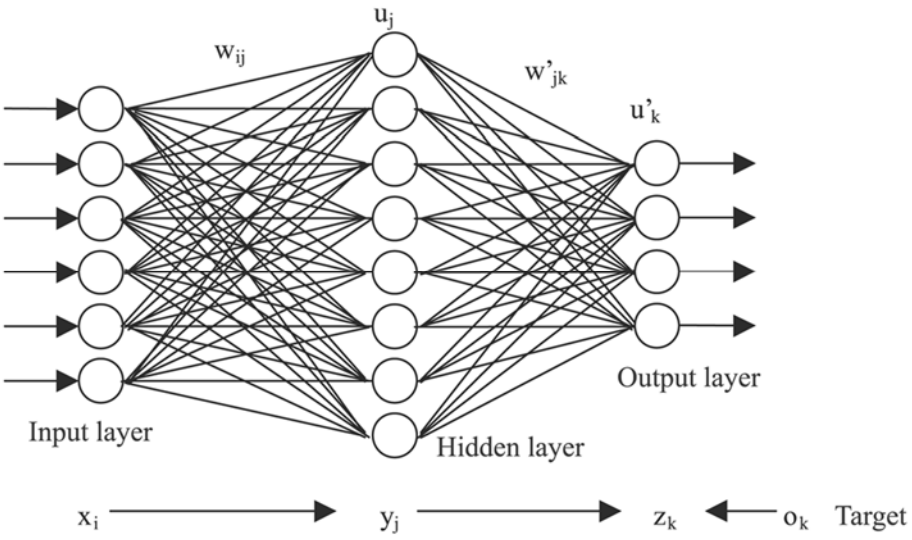
無論從腦波頻率的高低(即神經元活躍程度)，或是由該機制的生理作用來看，REM 時期都有顯著且獨特的生理意義。因此，本研究借鏡生物睡眠時期的 REM 時期進行類神經網路優化法的改良。REM 時期高頻率的腦波活動，其活躍程度堪比清醒時的狀態，並且能夠增強各神經元間的連結；本研究為類神經網路設置一個類似 REM 時期的機制，使類神經網路的權重更新更快速而穩定，即增加訓練時期的穩健性（robustness），並以此演算法搭配不同的優化法，觀察並分析其中的交互作用。

（二） 類神經網路相關的文獻探討

1. 類神經網路架構

不同的架構會賦予該網路對不同問題的解決能力，如分類（Classification）、迴歸（Regression）、圖像識別（Image Recognition）、自然語言處理（NLP）……等，視研究者或開發者的需求而定。以下探討三種常見的網路架構。

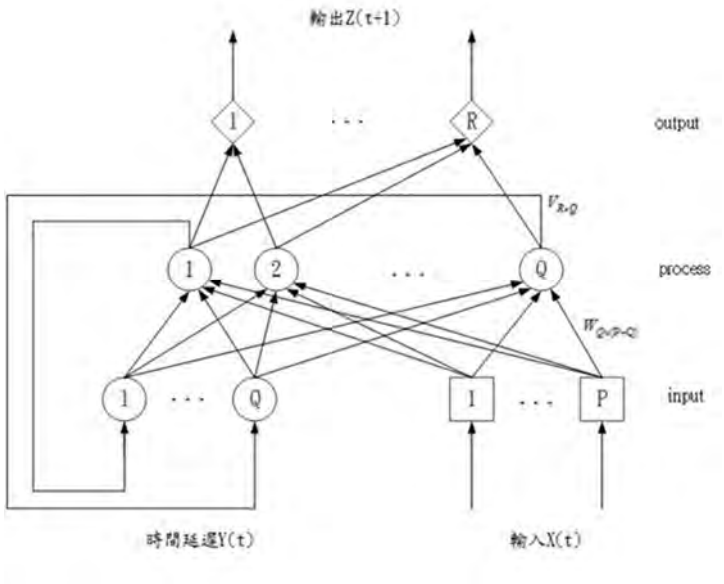
（1） 全連接神經網路（Fully-connected Neural Network）



（圖 2 全連接類神經網路架構圖）

全連接神經網路是最耳熟能詳、應用最廣泛的類神經網路之一，透過各層神經元間的權重連接之更新，可以逐漸擬合資料集提供的數據，因此得以解決分類或迴歸問題，也可以為其他學術領域的研究提供數據挖掘與資料分析的服務。雖然從理論上來說也可以處理圖片像素的輸入，但這種結構會致使類神經網路的參數過多、計算量龐大，且圖片的部分特性也無法表露出來，因此在電腦視覺領域中並不活躍。

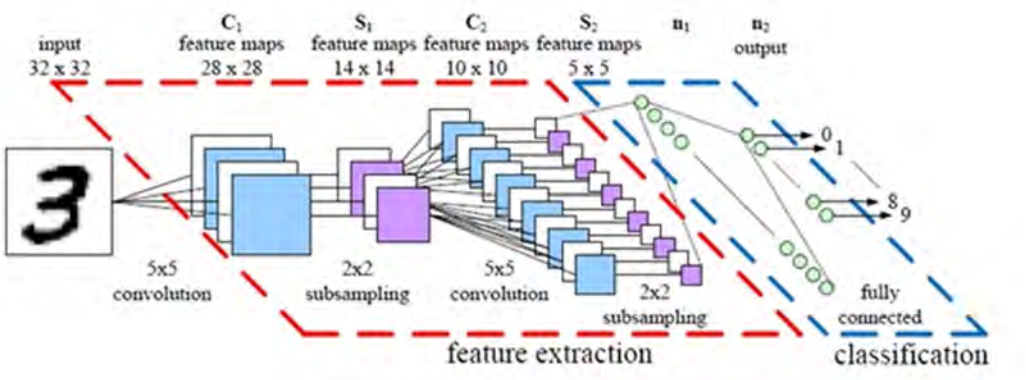
（2） 遞歸神經網路（Recurrent Neural Network）



（圖 3 遞歸類神經網路架構圖）

遞歸神經網路主要有兩大類，一為時間遞歸神經網路，一為結構遞歸神經網路，其中前者發展出「長短期記憶網路（LSTM）」這個經典的變體。這種結構讓它能夠處理連續性資料，在語句分析和語音處理上都很有效果。近年來在市場上大有噱頭的「程式創作音樂」，也是應用了這種類神經網路。

（3） 卷積神經網路（Convolutional Neural Network）



（圖 4 卷積類神經網路架構圖）

卷積神經網路透過特殊的卷積層(convolution)、池化層(pooling)，得以對圖片中的像素做分析處理，並一步一步從低階特徵提取出更高階、抽象的特徵來進行影像識別的工作。而像素的處理是其它種類神經網路難以做到的，故卷積類神經網路得

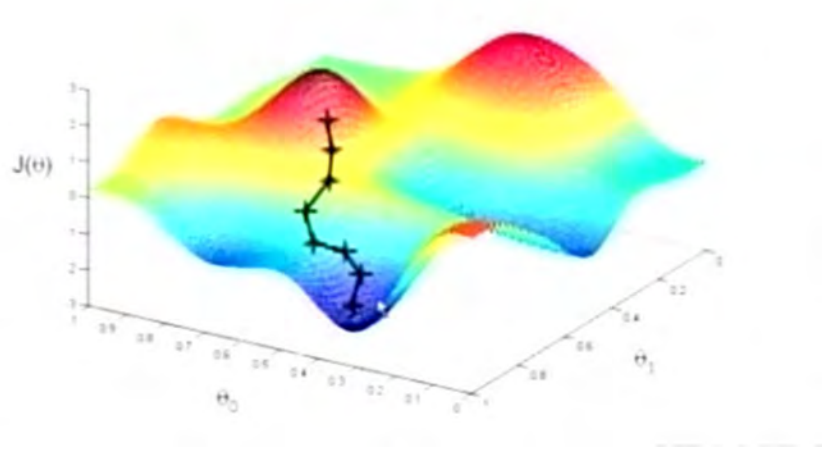
以成為應用最為廣泛的類神經網路之一，並且也是未來最有發展潛力、最被看好的網路。

3. 權重更新演算法

不論是上述何種類神經網路，其訓練都是以最佳化模型內部的參數為目的。至今發展出的權重更新演算法（即優化法，optimizer）有許許多多，各有其優缺點與算法特性，視程式設計者本身的需求而抉擇。以下探討三種常見的優化法：

（1） 批量梯度下降法（Batch Gradient Descent）

每次訓練都會遍歷資料集中的樣本，並計算梯度的大小，再乘上學習率，讓類神經網路沿著梯度的反方向（往更低點的方向）進行迭代的參數更新，以找到最適當的參數，使類神經網路擬合訓練時使用的資料集。另有隨機梯度下降法（Stochastic Gradient Descent）和小批量梯度下降法（Mini-Batch Gradient Descent）等變體，這三者都屬於傳統梯度下降法，只是對資料集的投餵方式不同。



（圖 5 梯度下降法示意圖）

本研究有採用的隨機梯度下降法（Stochastic Gradient Descent），不會在每次訓練中都進行遍歷，而是一次只會以一個資料點來進行梯度下降，所以速度會大幅高於批量梯度下降法。雖然該優化法存在著容易被資料集中的雜訊（noise）所影響的

缺點，不過本研究經過實測，確認使用的資料集並無因此缺點而產生任何訓練問題，所以仍決定採用隨機梯度下降法。

隨機梯度下降法的權重更新公式：

$$\Delta \mathrm{W} = \mathrm{W}' - \mathrm{W} = -\eta \frac{\partial E}{\partial W} \qquad \text{（式 1.1）}$$

其中，W 為各層神經元間的連結加權值，∂E 為各層神經元間的連結權重值的修正量，η 為學習速率(learning rate)，控制連結權重修正量的幅度。

（2） Momentum 方法

針對梯度下降法可能造成冗餘計算的問題，有研究者提出了「慣性」的概念，變形了原始的梯度下降法公式，規定每次的權重更新都要參考前一次的更新量，讓類神經網路在「下山」的過程中得到一個慣性速度。這能讓梯度下降過程變得較平滑、提升訓練的效能，且其算法特性被視為可以解決局部最小值（local minimum）的問題，傳統梯度下降法則缺乏這個能力。

Momentum 的權重更新公式：

$$\Delta \mathrm{W} = \mathrm{W}' - \mathrm{W} = -v_t = -\gamma v_{t-1} - \eta \frac{\partial E}{\partial W} \qquad \text{（式 1.2）}$$

其中v_t為在第 t 次的權重更新量；γ為慣性常數，常設為 0.5、0.9 或 0.99。

（3） Adagrad 方法

該優化法存在著一個衰減項，能在訓練過程中動態調整每個參數的學習率，這個技巧被稱為自適應學習率（adaptive learning rate），是傳統梯度下降法、momentum 方法及其變體 NAG 方法都沒有的。實驗證明，自適應學習率能改善許多訓練時可能遇到的問題，讓訓練速度更快，訓練進程也顯得非常穩定。

Adagrad 的權重更新公式：

$$\Delta \mathrm{W} = \mathrm{W}' - \mathrm{W} = -\frac{\eta}{\sqrt{\sum_{k=1}^{t-1}(\frac{\partial E_k}{\partial W_k})^2 + \epsilon}} \cdot \frac{\partial E}{\partial W} \qquad \text{（式 1.3）}$$

其中ϵ為極小的正數，可設為 10^{-8}。

4. 損失函數（loss function）

梯度下降法先透過損失函數來度量預測值與標籤值（target）之差距，並以此為根據進行權重的更新。常會添加正則項（regularization）來避免網路模型相對於資料集而言過度複雜。損失函數通常有「值恆不小於零」與「企圖使其為最小值」兩種特性，一個好的損失函數能建構出較有效果的更新規則，目前有兩種主流的損失函數形式，以下分別探討。

（1） 均方誤差損失函數（MSE）

將該損失函數的公式微分後，可知其目的在描述多維空間中兩點的歐式距離（Euclidean distance），歐式距離除了計算簡單之外，也是一個非常好用的相似性度量標準。因此，類神經網路的訓練常以此為損失函數，透過求解最小歐式距離的方式計算擬合曲線。MSE 公式如下：

$$MSE = \frac{1}{2n}\sum_{i=1}^{n}(T_i - Y_i)^2 \qquad \text{（式 1.3）}$$

其中 T 為期望值，Y 為預測值。

（2） 交叉熵損失函數（Cross Entropy）

Cross Entropy 亦稱為 K-L 散度，旨在計算兩個機率分佈間的距離，這也是一個很好的相似性度量標準。且比對其與 MSE 微分後的算式，可知前者不用考慮激勵函數（activation function）的導數，因此能避免梯度可能在訓練前期消失的問題。在分類問題中，該損失函數常配合 Softmax 函數使用。Cross Entropy 公式如下：

$$H(p,q) = -\sum p(x)\cdot logq(x) \qquad \text{（式 1.3）}$$

P(x)為期望機率分布，q(x)為預測機率分布。

上述兩種損失函數在實務應用上較常被使用，能提供的訓練效果也較佳，至於其它的 0-1 Loss、Hinge Loss 等，因已較無實用性，所以在本研究中不予詳細討論。

（三） 研究設備及器材

1. 硬體

（表 2 研究之硬體設備）

編號	名稱	規格
1	Apple MacBook Air	處理器：1.6 GHz Intel Core i5 記憶體：8.00 1600 MHz DDR3 作業系統：mac OS 10.12.1
2	ASUS X550CC Laptop	處理器：Intel Core i5-3337U 1.8GHz 記憶體：8.00GB 作業系統：Microsoft Windows 10 家用版

2. 軟體（包括函式庫）

（表 3 研究之軟體設備）

編號	名稱	版本
1	Python	3.6.1
2	Sublime Text Build 3143	Build 3143
3	TeXstudio 2.12.6	2.12.6
4	Csv	1.0
5	Pylab	與 Matplotlib 相同
6	Math	與 Python 相同
7	Microsoft Excel 2013	2013
8	IBM SPSS Statistics 22	22.0.0.0
9	SigmaPlot 10.0	10.0.0

3. 類神經網路架構

若採用如 Caffe、Tensorflow、scikit-learn 等開源函式庫，將難以進行部分研究所需資料的連續性觀察與細微部份的檢視，所以本研究使用 Brownlee, J. (2016)[8]提供的、單純由 python 及其內建函式庫實現的全連接類神經網路，並適當修改以符合本次研究之需求（原程式碼在資料的處理中有錯誤，且僅使用 SGD 優化法）。完整程式碼參照附錄。

該類神經網路使用了以下網路架構及機制：

（1） 全連接類神經網路（Fully-connected Neural Network）：由於其網路架構及神經元權重更新方式的設計，其結果較容易直接觀察，且因為所有架構的網路模型都用梯度下降法，所以不會有研究結果不能推廣至所有架構的疑慮。

（2） 單層隱藏層（Hidden Layer）：由於本研究取用的資料集複雜性不高，故單層隱藏層就已足以抓取足夠抽象的特徵。

（3） 標準化（Normalization）：特徵值數值的不同量級，會導致損失函數構成的函數圖形之輪廓十分扁長，因此需要使用標準化以將各特徵值映射到一固定區間，以使函數圖形較偏圓形。

（4） 獨熱編碼（One-Hot encoding）：由於資料集為分類問題，故以其將各分類結果映射至一固定長度的向量。同一分類結果對應的同一分量，其值為一，而其餘分量之值皆為零。

（5） 五折交叉驗證（5-folds Cross Validation）：使用嚴謹的交叉驗證，計算數學模型的使用與否，其造成的程式準確率差異。折數為五。

（6） 均方誤差損失函數（Mean Squared Error）：效果最好且被廣泛使用的損失函數之一，且微分後方程簡潔、容易觀察。本研究中統一使用該損失函數，以觀察 REM 方法對不同優化法提供的效果。

優化法依其算法特性可略分為三類，而本研究在這三個類別中各選擇一種優化法探討。以下分別闡述選擇原因：

（1） 隨機梯度下降法（Stochastic gradient descent, SGD）

傳統的梯度下降法有 BGD、SGD、MBGD 三種模式。

雖說以本研究使用的資料集規模，BGD 並不會造成網路運算量過大的問題，但其今日在實務上的應用已幾乎不可見，故本研究採用三者中最快速的 SGD。因為使用的資料集都是各大學或研究單位提供的數據，所以 SGD 擔心的雜訊過多問題也不足為慮。

（2） Momentum 方法

動量法有 Momentum 與 NAG 兩種，基本原理完全相同，後者為前者的變形，且前者的算法特性較為強烈。本研究選擇前者，因為只需要證明 Momentum 與 REM 方法 ` （即慣性項與自適應損失函數）的相容性，NAG 就也同時得證了。

（3） Adagrad 方法

自適應學習率有許多種實現方法，其中以 Adagrad、Adadelta、Adam、RMSProp 為完整的體系。它們的基本原理也都很相似。本研究選擇較典型的 Adagrad，檢驗自適應學習率與自適應損失函數的相容性與效果。

程式碼中，未使用下列方法：

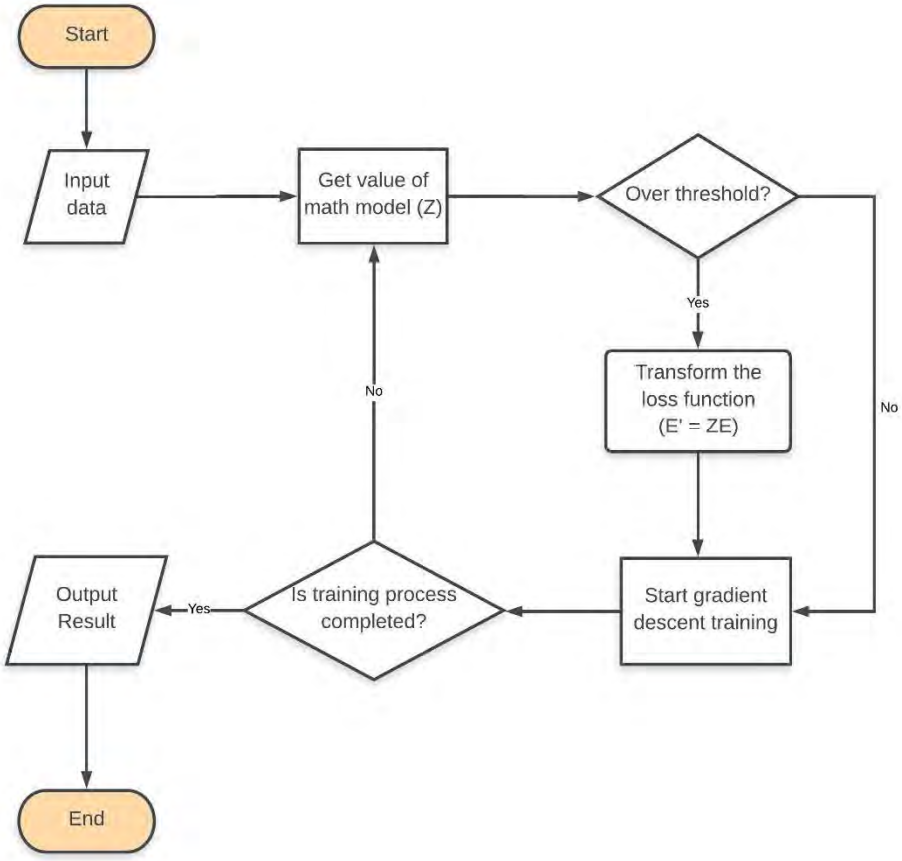
（1） 正則化（regularization）：損失函數通常是差值項與正則項的和，而在某些情況下，正則項可能會遠大於資料損失（即損失函數的總值大部分由正則化所貢獻），這將會降低本研究的實驗準確性。再者，使用的資料集明顯不會讓類神經網路因為太複雜，而造成過擬合的問題。因此，本研究中不須使用正則項。

（2） 提前停止(early-stopping)：為了觀察到長期訓練週期中準確率的變化，故不採用此機制。

（四） 演算法概念

不論是何種架構的類神經網路，其運作原理都是相似的，訓練的目的也都是最佳化參數，因此在本研究中，將以整體結構較簡單的全連接類神經網路來驗證本研究提出的假說：生物在 REM 時期時，腦波頻率較高，與清醒時相似；將此機制應用於深度學習領域上，就

可以透過一數學式來適當增加權重更新的幅度，進而加速類神經網路的訓練進程、加強網路學習的強健性，與 REM 時期時的腦波強化對生物學習行為的影響一般。



(圖 6 概念流程圖)

（五） REM 方法的推導與確立、公式的變形

1. 數學項（式中為 Z）形式

為了改變類神經網路的「活躍程度」，故將數學項作為損失函數（E）的實數係數，即：

$$E' = \mathrm{ZE} \qquad \text{（式 2.1）}$$

再來要決定數學項的形式，對它有以下要求：

（1） 在訓練週期（式中為 t）的漸漸增加中，其值要呈現慢慢衰減的狀況，亦即數學項的值與訓練週期成反向變動。

因為在類神經網路的訓練中，隨著訓練週期的增加，網路中的參數形成的超維函數圖形與其值所決定的點，點會越來越接近圖形的臨界點（critical point），因此權重更新的幅度應當漸漸降低。倘若數學項始終保持定值，則愈在訓練後期，愈容

易使類神經網路產生震盪（oscillation）、無法靠近臨界點。此概念與學習率的衰減類似。

根據此想法，可將數學項初步設為：

$$Z = e^{-\lambda t} \quad \text{（式 2.2）}$$

其中，λ 為衰減常數。指數上的負號實現了 Z 與 t 成反向變動的概念。底數設為尤拉數意在簡化運算。

（2） 恆大於 1

本研究的目的在於，透過變形損失函數的公式來放大梯度，以此加速訓練進程。因此，若數學項小於 1，一來與本研究的研究方向不同，二來反而會減少梯度下降法的更新幅度，得到反效果。因此，必須為此數學項補上一個常數 n（n＝1），使其恆大於 1。由式 2.2 可得：

$$Z = e^{-\lambda t} + 1 \quad \text{（式 2.3）}$$

（3） 能精準控制初期的值，並且比單純的指數衰減有更高的彈性

讓梯度在何時得到最佳的增幅值，將直接影響演算法的效果，尤其在初期幾個訓練週期的影響特別大。並且，不同的資料集與神經網路模型在前期也必定會適應不同程度與樣式的增幅，有些需要大量且快速下降的增幅、有些可以使用小量而穩定的增幅，因此本研究希望提供 Z 一個類似於「初始速度」的效果，增加該演算法的操作彈性與對不同模型的適應性。因此，數學項的形式最終確定為：

$$Z = e^{k-\lambda t} + 1 \quad \text{（式 2.4）}$$

其中，k 與 λ 為超參數（hyperparameter），k 決定初始增幅值、 λ 決定增幅值下降幅度，兩者的配合能提高數學項的彈性，增加了它在整個神經網路演算法的可用性。

本研究初期曾經將這兩個超參數都設為定值，並將式 2.4 中的 1 改成可隨意調整的超參數 n。不過在資料集測試與數據分析後發現，數學項的彈性非常低，難以實現對不同資料集的適應，也就是說對損失函數的變形並沒有達到非常好的效果；提出現有模式的數學項後，則嘗試過將尤拉數去除，即把數學項的指數衰減特性改寫為線性衰減，不過效果也不如指數衰減特性的數學項。

2. 閾值（threshold）的決定

閾值的設定是個很值得探討的問題。設置它的原因在於，本研究並不希望這個機制在程式從頭到尾的訓練中都被觸發，而是只希望它在訓練前半期存在。因為權重更新階段越到後期，權重的變化量理應越小，若在那時用數學項讓損失函數變大，進而使權重更新的幅度增加，可能會使訓練產生震盪，反而讓類神經網路無法收斂；若用數學項讓損失函數變小，而令權重更新階段在訓練後半期每輪迭代的更新值變更小，則會使程式在合理的訓練次數下難以收斂。

再者，既然決定此數學項必然隨訓練週期的增加而持續衰減，則到訓練後期的增幅值將會愈來愈接近 1，對損失函數的改變幾乎沒有影響，故可省略不計，以節省程式運算資源。因此閾值的設定是必要的，以決定該數學項要在何種情況下被觸發。

本研究對閾值有此要求：可以讓數學項在網路訓練的初期時被觸發，以適當提高損失函數的值，加速訓練進程；並在網路訓練中後期不觸發數學項，使其維持穩定的權重更新。

至於閾值的大小，將會影響演算法的訓練效果。若其值過大，數學項為損失函數貢獻的增幅將被限制，使得演算法效果不彰；若其值過小，則訓練後期數學項近乎無意義的增幅，僅能極小的改變損失函數值，造成運算資源的浪費。本研究曾將閾值設置得極小，分析數據後確認有許多無貢獻的小數點後多位運算，因此，本研究決定參考在數學視角上，對何種量級的數值會選擇省略。實測發現這樣的設置效果良好。

一般而言，在數學上：

$$if\ n \leq 10^{-4}$$
$$then\ n+1\ \cong 1 \qquad \text{（式 3.1）}$$

所以本研究把觸發數學項的條件訂為：

$$\text{if } e^{k-\lambda t} \geq 10^{-4}$$
$$\text{then } \mathrm{E}' = \mathrm{ZE} = (e^{k-\lambda t}+1)\cdot E \qquad \text{（式 3.2）}$$

3. REM 方法的數學意義

梯度下降法的核心原理是，以類神經網路模型中的權重與偏權值（bias）作為變數，構成一個在數學視角上屬於多變數函數的損失函數，並企圖找到最佳的參數組合，將其梯度向量的每個分量（即損失函數分別對每個參數的偏微分）都降至零，如此一來，梯度的向量長度會變得越來越小、直至為零，屆時與單位向量內積形成的方向導數也會越來越小、直至為零。亦即，梯度下降法能找到損失函數圖形組成的超維空間中的某一點（最佳的參數組合），且其對任何方向的斜率都會是零，則代表這個點落入了臨界點（critical point）—當然，無法保證它一定會落入極小值，也可能落入鞍點。

而 REM 方法中的數學項，在同一個訓練週期裡每個迭代貢獻的增幅值是固定的，若數學項大於閾值，則將所有分量增幅一個定值，且因為數學項與梯度的分量是乘積關係，所以越大的分量就會得到越大的增幅；反之，則所有分量維持不變。完成之後，函數圖形的某些方向在這次迭代中就會變陡，梯度下降法就會讓被增幅之分量對應的參數值有較大的更新值了。隨著訓練週期的增加，數學項因為閾值的關係而不會再被觸發，便不再改變損失函數的梯度，讓其維持穩定平緩的更新。

一般若將整個訓練進程的梯度下降等高線圖畫出，不同優化法都是在改變下降曲線以求更快速穩定的訓練；本研究提出的 REM 方法則是在動態的改變等高線圖本身，由強烈至平緩的稠密化當下空間中這一點的等高線。因此若將 REM 方法與不同種類優化法結合，勢必會有不同的變化。

4. 傳統梯度下降法的公式變形

把數學項代入傳統梯度下降法的公式當中。將（式 1.1）透過連鎖律（chain rule），可將$\frac{\partial \mathrm{E}}{\partial \mathrm{W}}$項展開為：

$$\frac{\partial E}{\partial W_{jk}} = \frac{\partial E}{\partial Y_j} \cdot \frac{\partial Y_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial W} \qquad （式 4.1）$$

其中， ∂E 為各層神經元間的連結權重值的修正量，W 為各層神經元間的連結加權值， $\mathrm{net_j}$ 為 j 神經元之輸入總和與偏權值（bias）之和，$\mathrm{Y_j}$為激勵函數（activation function）以$\mathrm{net_j}$作為自變數得到的應變數，即神經元輸出。

以下分別討論三個運算元：

$$\frac{\partial E}{\partial Y_{jk}} = \frac{\partial}{\partial Y_j}\left[\frac{1}{2}\sum_j^M \left(T_j - Y_j\right)^2\right] = -(T_j - Y_j) \quad \text{（式 4.2）}$$

$$\frac{\partial Y_j}{\partial net_j} = \frac{\partial}{\partial net_j} f\left(net_j\right) = f\left(net_j\right)\cdot\left[1 - f\left(net_j\right)\right] = Y_j \cdot (1 - Y_j) \quad \text{（式 4.3）}$$

$$\frac{\partial \mathrm{net}_j}{\partial \mathrm{W}_{jk}} = \frac{\partial}{\partial W_{jk}}\left[\sum_k^N W_{jk}H_k - \theta_j\right] = H_k \quad \text{（式 4.4）}$$

由上面三個數學式，合併後可得到：

$$\frac{\partial E}{\partial W_{jk}} = -(T_j - Y_j)\ \cdot Y_j \cdot (1 - Y_j)\cdot H_k \quad \text{（式 4.5）}$$

上式即為輸出層神經元的權重更新公式，因此若將原本的損失函數透過先前確立的數學項變形，即：

$$E' = \frac{1}{2}\sum_j^M \left(T_j - Y_j\right)^2 \cdot (e^{k-\lambda t} + 1) \quad \text{（式 4.6）}$$

則損失函數對類神經網路之輸出值的偏微分就是：

$$\frac{\partial E}{\partial Y_{jk}} = \frac{\partial}{\partial Y_j}\left[\frac{1}{2}\sum_j^M \left(T_j - Y_j\right)^2 \cdot (e^{k-\lambda t} + 1)\right] = -\left(T_j - Y_j\right)\cdot (e^{1-0.9t} + 1) \quad \text{（式 4.7）}$$

於是得到隱藏層神經元的權重更新公式（導證過程不再贅述）：

$$\frac{\partial E}{\partial W_{jk}} = [\sum_j^{N_{output}} \left(T_j - Y_j\right)\cdot Y_j\left(1 - Y_j\right)\cdot W_{jk}]\cdot H_k \cdot (1 - H_k)\cdot X_i \quad \text{（式 4.8）}$$

將數學項添加於損失函數上，該公式隨之變形為：

$$\frac{\partial E'}{\partial W_{jk}} = \left[\sum_j^{N_{output}} \left(T_j - Y_j\right)\cdot (e^{k-\lambda t} + 1)\cdot Y_j\left(1 - Y_j\right)\cdot W_{jk}\right]\cdot H_k \cdot (1 - H_k)\cdot X_i \quad \text{（式 4.9）}$$

將得到的數學項寫入類神經網路中權重更新的函式，以下以虛擬碼呈現：

Algorithm 1 SGD with REM

```
1. id = 1 {Default: the math model will be activated.}
2. {Update Weights}
3.
4. for 0 ≤ i < length of neural network do
5.    if i = 0 then {the input layer}
6.       inputs = all attributes except class attribute
7.    else {the hidden layer / the output layer}
8.       inputs = outputs of the previous layer
9.    end if
10.
11.   for each neuron in layer i do
12.      if id = 1 then
13.         if e^{k−λ×t} ≥ 0.0001 then {Activate the math model.}
14.            for 0 ≤ j < length of input do
15.               weight_j− = |η × δ_j × input_j × (e^{k−λ×t} + 1)|
16.            end for
17.            weight_{−1}− = |η × δ_j × (e^{k−λ×t} + 1)|
18.
19.         else {The math model will not be activated anymore.}
20.            id = 0
21.         end if
22.      end if
23.
24.      if id = 0 then
25.         for 0 ≤ j < length of input do
26.            weight_j− = |η × δ_j × input_j|
27.         end for
28.         weight_{−1}− = |η × δ_{−1}|
29.      end if
30.   end for
31.
32. end for
```

（圖 7 SGD 搭配 REM 演算法的虛擬碼）

其中 η 表示學習率（learning rate），δ_j表示第 j 個神經元的差距量。k 與 λ 為數學項當中的超參數，t 表示訓練週期。

5. Momentum 的公式變形

Momentum 為典型的「動量法」，會將上次訓練的梯度考慮進當次的權重更新，因此數學項會有不同於對 SGD 的影響。在前述傳統梯度下降法的公式變形中已有相關導證，以下省略公式變形的詳細推導，直接呈現運算後的結果。

Momentum 的權重更新公式，由（式 1.2）改寫後得到：

$$v_t = \gamma v_{t-1} + \eta \frac{\partial E}{\partial W} \quad \text{（式 5.1）}$$

$$\Delta \mathrm{W} = \mathrm{W}' - \mathrm{W} = -v_t = -\gamma v_{t-1} - \eta \frac{\partial E}{\partial W} \quad \text{（式 5.2）}$$

其中$\mathbf{v_t}$為在第 t 次的權重更新量；γ為慣性常數，常設為 0.5、0.9 或 0.99。

因此，對輸出層神經元的權重更新，在程式裡以代數處理形式表示即為：

$$v_t = \gamma v_{t-1} + \eta \cdot [-(T_j - Y_j) \cdot Y_j \cdot (1 - Y_j) \cdot H_k] \quad \text{（式 5.3）}$$

$$\Delta \mathrm{W} = \mathrm{W}' - \mathrm{W} = -v_t = -\gamma v_{t-1} + \eta \cdot (T_j - Y_j) \cdot Y_j \cdot (1 - Y_j) \cdot H_k \quad \text{（式 5.4）}$$

而對隱藏層神經元的權重更新，在程式裡以代數處理形式表示即為：

$$v_t = \gamma v_{t-1} + \eta \cdot [\sum_{j}^{N_{output}} (T_j - Y_j) \cdot Y_j (1 - Y_j) \cdot W_{jk}] \cdot H_k \cdot (1 - H_k) \cdot X_i \quad \text{（式 5.5）}$$

$$\Delta \mathrm{W} = \mathrm{W}' - \mathrm{W} = -v_t = -\gamma v_{t-1} - \eta \cdot [\sum_{j}^{N_{output}} (T_j - Y_j) \cdot Y_j (1 - Y_j) \cdot W_{jk}] \cdot H_k \cdot (1 - H_k) \cdot X_i$$

（式 5.6）

將作用於損失函數的數學項添入原公式，對輸出層與隱藏層神經元的權重更新分別變形為：

$$\Delta \mathrm{W} = -\gamma v_{t-1} + \eta \cdot (T_j - Y_j) \cdot (e^{k-\lambda t} + 1) \cdot Y_j \cdot (1 - Y_j) \cdot H_k \quad \text{（式 5.7）}$$

$$\Delta \mathrm{W} = -\gamma v_{t-1} - \eta \cdot [\sum_{j}^{N_{output}} (T_j - Y_j) \cdot (e^{k-\lambda t} + 1) \cdot Y_j (1 - Y_j) \cdot W_{jk}] \cdot H_k \cdot (1 - H_k) \cdot X_i$$

（式 5.8）

將數學項寫入類神經網路中權重更新的函式，以下以虛擬碼呈現：

Algorithm 2 Momentum with REM

1. $id = 1$ {Default: the math model will be activated.}
2. {Update Weights}
3.
4. **for** $0 \leq$ i $<$ length of neural network **do**
5. **if** $i = 0$ **then** {the input layer}
6. inputs = all attributes except class attribute
7. **else** {the hidden layer / the output layer}
8. inputs = outputs of the previous layer
9. **end if**
10.
11. **for** each neuron in layer i **do**
12. **if** id $= 1$ **then**
13. **if** $e^{k-\lambda \times t} \geq 0.0001$ **then** {Activate the math model.}
14. **for** $0 \leq$ j $<$ length of input **do**
15. $weight_j - = |\gamma \times \delta_j + \eta \times \delta_j \times input_j \times (e^{k-\lambda \times t} + 1)|$
16. **end for**
17. $weight_{-1} - = |\gamma \times \delta_{-1} + \eta \times \delta_{-1} \times (e^{k-\lambda \times t} + 1)|$
18.
19. **else** {The math model will not be activated anymore.}
20. $id = 0$
21. **end if**
22. **end if**
23.
24. **if** id $= 0$ **then**
25. **for** $0 \leq$ j $<$ length of input **do**
26. $weight_j - = |\gamma \times \delta_j + \eta \times \delta_j \times input_j|$
27. **end for**
28. $weight_{-1} - = |\gamma \times \delta_{-1} + \eta \times \delta_{-1}|$
29. **end if**
30. **end for**
31.
32. **end for**

（圖 8 動量法搭配 REM 演算法的虛擬碼）

其中 γ 表示動量法當中的慣性常數。

6. Adagrad 的公式變形

Adagrad 的權重更新公式，將（式 1.3）改寫後得到：

$$\mathrm{G}_t = \sum_{k=1}^{t-1} (\frac{\partial E_k}{\partial W_k})^2 \qquad （式 6.1）$$

$$\Delta \mathrm{W} = \mathrm{W}' - \mathrm{W} = -\frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \frac{\partial E}{\partial W} \qquad （式 6.2）$$

其中$\mathrm{G_t}$為該權重歷次訓練得到的梯度的平方之和，ϵ為極小的正數，可設為10^{-8}；

因此，對輸出層神經元的權重更新，在程式裡以代數處理形式表示即為：

$$\Delta \mathrm{W} = -\frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \frac{\partial E}{\partial W} = -\frac{\eta}{\sqrt{G_t + \epsilon}} \cdot (T_j - Y_j) \cdot Y_j \cdot (1 - Y_j) \cdot H_k \qquad （式 6.3）$$

而對隱藏層神經元的權重更新，在程式裡以代數處理形式表示即為：

$$\Delta \mathrm{W} = -\frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \frac{\partial E}{\partial W} = -\frac{\eta}{\sqrt{G_t + \epsilon}} \cdot [\sum_{j}^{N_{output}} (T_j - Y_j) \cdot Y_j (1 - Y_j) \cdot W_{jk}] \cdot H_k \cdot (1 - H_k) \cdot X_i$$

（式 6.4）

將作用於損失函數的數學項添入原公式，對輸出層與隱藏層神經元的權重更新分別為：

$$\Delta \mathrm{W} = -\frac{\eta}{\sqrt{G_t + \epsilon}} \cdot (T_j - Y_j) \cdot (e^{k-\lambda t} + 1) \cdot Y_j \cdot (1 - Y_j) \cdot H_k \qquad （式 6.5）$$

$$\Delta \mathrm{W} = -\frac{\eta}{\sqrt{G_t + \epsilon}} \cdot [\sum_{j}^{N_{output}} (T_j - Y_j) \cdot (e^{k-\lambda t} + 1) \cdot Y_j (1 - Y_j) \cdot W_{jk}] \cdot H_k \cdot (1 - H_k) \cdot X_i$$

（式 6.6）

上述幾式並未印出G_t被數學項影響的代數式，是因為雖然數學項的定義是添加於損失函數前，當次權重更新得到之梯度與G_t中的梯度都應該要被數學項影響，不過若歷次與當次梯度都附加上數學項，則可以透過數學項恆衰減的特性保證分母獲得之增幅值必大於當次梯度獲得之增幅值，同義於略微增幅 Adagrad 的衰減項，這並不會對訓練進程貢獻新的效果。其他使用自適應學習率的優化法多也有類似情況。

因此本研究定義，該數學項在與自適應學習率配合時，有效範圍只限於當次梯度中的損失函數，而不會影響自適應學習率其固有的機制。將數學項寫入類神經網路中權重更新的函式，以下以虛擬碼呈現：

Algorithm 3 Adagrad with REM

1. $id = 1$ {Default: the math model will be activated.}
2. {Update Weights}
3.
4. **for** $0 \leq$ i $<$ length of neural network **do**
5. **if** $i = 0$ **then** {the input layer}
6. inputs = all attributes except class attribute
7. **else** {the hidden layer / the output layer}
8. inputs = outputs of the previous layer
9. **end if**
10.
11. **for** each neuron in layer i **do**
12. **if** id $= 1$ **then**
13. **if** $e^{k-\lambda \times t} \geq 0.0001$ **then** {Activate the math model.}
14. **for** $0 \leq$ j $<$ length of input **do**
15. $weight_j - = |(\eta \div \sqrt{G_j + \epsilon}) + \eta \times \delta_j \times input_j \times (e^{k-\lambda \times t} + 1)|$
16. **end for**
17. $weight_{-1} - = |(\eta \div \sqrt{G_j + \epsilon}) + \eta \times \delta_{-1} \times (e^{k-\lambda \times t} + 1)|$
18.
19. **else** {The math model will not be activated anymore.}
20. $id = 0$
21. **end if**
22. **end if**
23.
24. **if** id $= 0$ **then**
25. **for** $0 \leq$ j $<$ length of input **do**
26. $weight_j - = |(\eta \div \sqrt{G_j + \epsilon}) + \eta \times \delta_j \times input_j|$
27. **end for**
28. $weight_{-1} - = |(\eta \div \sqrt{G_j + \epsilon}) + \eta \times \delta_{-1}|$
29. **end if**
30. **end for**
31.
32. **end for**

（圖 9 Adagrad 搭配 REM 演算法的虛擬碼）

其中G_j表示該權重歷次訓練得到的梯度的平方之和，ϵ為極小的正數，設為10^{-8}。

（六） 資料預處理與調參

本研究從加州大學爾灣分校的機器學習網站（UCI Machine Learning Repository），挑選五個數據量、特徵值（feature）、資料離散性各不相同的分類資料集。這些資料集在 Kaggle、麻省理工學院等機器學習相關網頁上也有提供，被廣泛的使用。

取得這些資料集後，先將資料進行預處理。首先，將資料集中的字串改為浮點數，並將分類結果移至末欄，以利程式運算，最後將資料點數量修剪成 5 的倍數，以確保能在使用五折交叉驗證（5-fold cross validation）的情況下精確追蹤損失函數。

先對第一個資料集做全面的詳細分析。至於第二到第五個資料集，為求精簡，將針對「該數學項對不同性質的資料集之訓練進程造成的影響」與「可能遇到或克服的訓練問題」來討論。另外，下文中的準確率是五折交叉驗證取得的平均值；損失函數-訓練週期圖只印出第一折交叉驗證時期的結果。

將訓練週期設為 200，並以短期、長期、微觀、宏觀等不同視角觀察；學習率設為 0.1；原始的 Momentum 所使用的慣性常數，在實驗 0.5、0.9、0.99 這三個常用值後，決定將其設為 0.9；數學項之常數 k、 λ 調參後決定最佳值；使用五折交叉驗證以確保實驗的嚴謹；依序套用 SGD、Momentum、Adagrad 三種優化法來測試其與 REM 方法的相容性、效果。

（七） 共變數分析（ANCOVA）

經過實驗得到各優化法的準確率-訓練週期圖後，本研究希望了解其結果之間是否存在著顯著性差異。因準確率會隨著訓練週期而變化，故採用共變數分析（Analysis of Covariance，以下簡稱 ANCOVA）進行資料檢定。也就是說，在排除訓練週期對於準確率的影響之後，各演算法的結果之間是否存在顯著性差異。

以 IBM SPSS Statistics 做為資料檢定軟體，使用其一般線性迴歸模型中的單變量分析。資料檢定中，訓練週期為共變量（covariance），演算法種類為自變數（independent variable），準確率為依變數（dependent variable），事後檢定則採用 Bonferroni 法。

檢定結果除了描述性統計資料與各組之間的 p 值（p value）外，亦會以「$F($組間自由度$,$組內自由度$) =$ F 值，$p = $ p 值」摘要的表示檢定結果。取 $p = 0.05$ 為顯著層級。

另外，為求表格以精簡的方式呈現，將各演算法按照下列方式編號、對應：

（表 4 優化法編號對照表）

編號	演算法
1	SGD（未使用 REM 方法）
2	SGD（使用 REM 方法）
3	Momentum（未使用 REM 方法）
4	Momentum（使用 REM 方法）
5	Adagrad（未使用 REM 方法）
6	Adagrad（使用 REM 方法）

三、 研究結果與討論

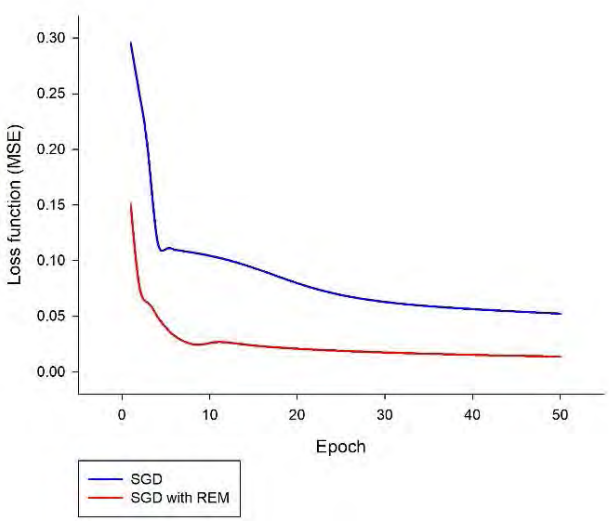
（一） 鳶尾花分類資料集

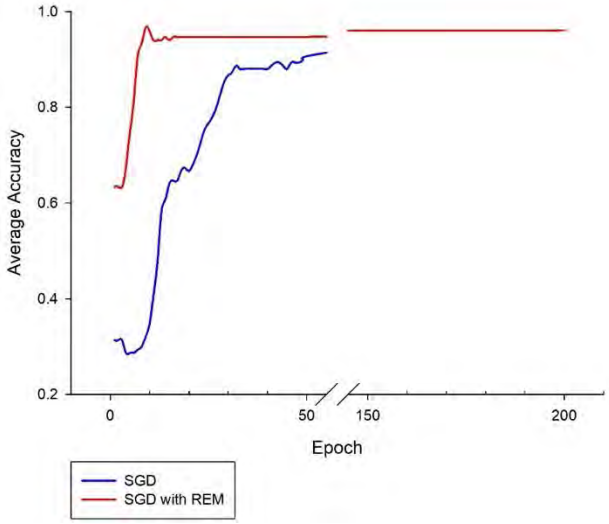
1. 資料集簡介

最經典且具指標性的分類資料集，常被引用於各論文中。

有 150 筆資料，4 個特徵值，鳶尾花品種被分為 3 類。

2. 以 SGD 為優化法的訓練進程變化分析





（左圖 10 （SGD）鳶尾花分類資料集之損失函數-訓練週期圖）

（右圖 11 （SGD）鳶尾花分類資料集之準確度-訓練週期圖）

類神經網路在添加 REM 方法後，其損失函數-訓練週期圖產生了顯著的改變。以下歸納出三點變化：

（1） 在訓練前期，損失函數的值看似未被放大，反而小於原始的損失函數值。

損失函數會隨著訓練週期的增加而慢慢遞減十分合理。加上恆大於一的數學項之後，因為訓練速度的提升，使得後期的損失函數值較低，並且也不會觸發數學項，這可以視為 REM 方法的效果，因為其設置目的為加快訓練速度；但由圖 10 可知，訓練初期的損失函數值也變小了，這與 REM 方法提供的前期增幅效果不符。這是因為圖中每個訓練週期之損失函數值的計算方法是，同一個訓練週期中所有迭代的損失函數值之平均，加上 REM 方法後訓練速度的大幅加快，使得損失函數值急速下降，因此前幾個訓練週期的平均值反而沒有原本的大。

（2） 訓練前期的損失函數值呈現更加平滑的下降。

推測是因為該資料集最佳的初始速度值較大（$k = 3.7$），所以訓練前期的損失函數值大多由增幅值來貢獻與影響。且衰減常數較小（$\lambda = 0.3$），因此隨著訓練週期的增加，增幅值的遞減不會那麼強烈，也就讓損失函數的下降變得平穩了。

（3） 訓練中後期，函數圖維持穩定平緩；且其損失函數值恆小於未使用 REM 方法的類神經網路。

因為在訓練中後期，損失函數不會被 REM 方法影響，造成不必要的放大，權重更新歸於穩定。且由於類神經網路的訓練進程被大幅且穩定的加快，所以在同一個訓練週期下，其損失函數值就會小上許多。由圖 10 知，在訓練中後期，未使用 REM 方法的損失函數圖形仍有些緩慢下降的趨勢，代表它離訓練終點還有些距離；同時使用 REM 方法的損失函數圖形之斜率則已趨近零。

準確度隨著訓練週期上升的函數圖則有以下變化：

（1） 準確度在訓練初期有了明顯的改進。

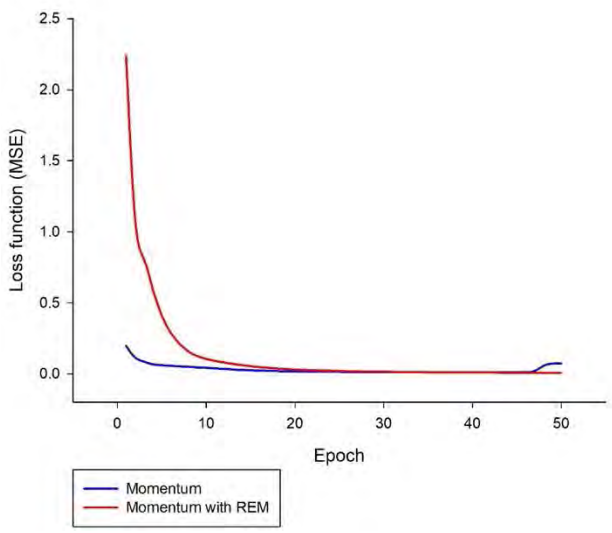
此即為本研究的目的：增強類神經網路訓練進程中的強健性（robustness）。在同樣的訓練週期下，因為訓練速度的提升，使類神經網路可以在前期就得到較高的準確性，這也增加了類神經網路對小型資料集的支持度。

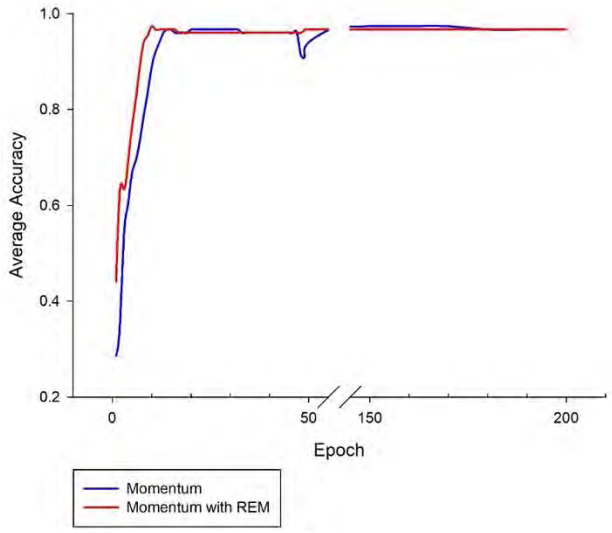
（2） 準確度在訓練中後期的差距變小，至訓練終點時幾近相同。

準確度相差幅度漸漸縮小，是因為損失函數之值極小，且數學項也早已低於閾值，因此不會觸發數學項，故類神經網路訓練的進程回歸慢而平穩，和正常狀態時相同。並且，雖然中後期不會再觸發數學項，但此時的紅色線早已趨於平穩，若使用 early-stopping 的話，應早就可以判定為訓練完成；而藍色線在中期仍有整體上升的波動，代表其尚未接近訓練終點。

使用 REM 方法後，約在第 10 個訓練週期就已訓練完畢；而未使用的話，要在約第 110 個訓練週期才完成，顯然在 Iris 這個經典資料集裡，SGD 能與 REM 方法高度相容，不過權重更新緩慢也是傳統梯度下降法本身就存在著的問題，即使是其中速度較快的 SGD 也不及之後測試的 Momentum 與 Adagrad。不過，在最終訓練終點的準確率都是差不多的，因為本研究沒有對整體的網路模型或是資料集的特徵選取做改變。

3. 以 Momentum 為優化法的訓練進程變化分析





（左圖 12 （動量法）鳶尾花分類資料集之損失函數-訓練週期圖）

（右圖 13 （SGD）鳶尾花分類資料集之準確度-訓練週期圖）

圖 12 有一處與圖 10（SGD）不同，就是在訓練前期，使用 REM 方法後的損失函數大幅超越未使用 REM 方法的。

回溯「圖中損失函數值的計算方法是，同一個訓練週期中所有迭代的損失函數值之平均」該敘述，可知道 SGD 的損失函數圖算是特例，因為其對訓練效果的增加非常顯著才會如此；而 Momentum 並未有同一訓練週期中損失函數值的大幅陡峭下降，因此 REM 方法為損失函數的影響就可直接由函數圖觀察到了—在訓練前期為損失函數提供一個隨訓練週期增加而逐漸變小、但卻恆大於一的加成。

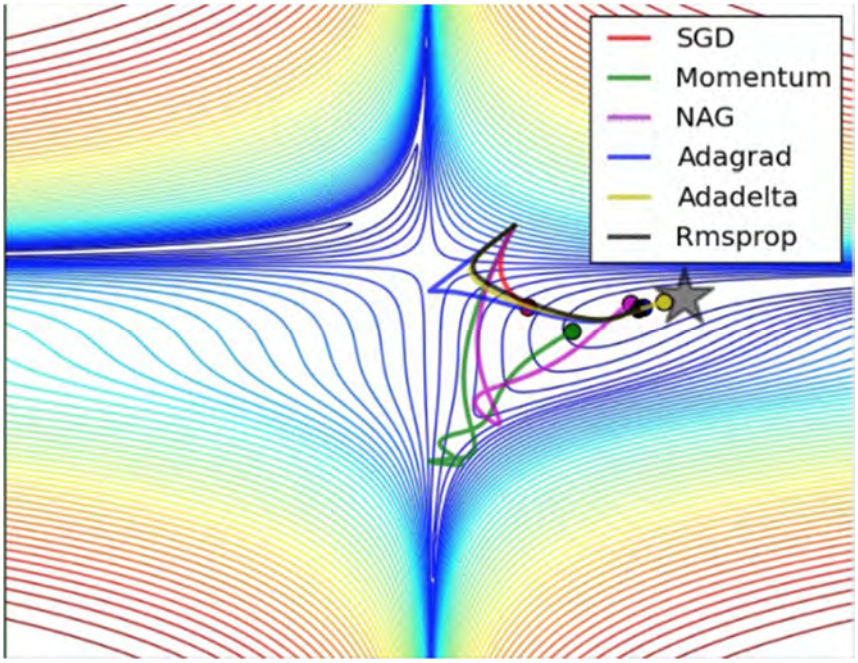
而由圖 13 可知，雖然前期的訓練強健性仍有所改善，不過明顯地沒有像是對 SGD 那麼良好的優化。初步判斷有兩個可能的原因：

（1） 比對圖 13 與圖 11（SGD）後，可知 Momentum 本身的速度是大幅勝過 SGD 的，但兩種優化法在添加 REM 方法後都得到差不多優良的結果，所以 Momentum 的進步相較之下就會比較小。

（2） 在諸多優化法中，Momentum 與 NAG 這兩種動量法的梯度下降圖形是非常獨特的，它們對上次訓練的梯度也參考進當次的權重更新，因此在訓練過程中，會沿著損失函數構成之圖形的等高線做快速地滑動，且前者的滑動量比後者高，可能影響了 REM 方法對損失函數變形的效果。

先前的調參結果指出，未添加 REM 方法的 Momentum 之慣性常數雖然都是以 0.9 為佳，但在添加 REM 方法後，慣性常數的最佳解都會隨之降低，如本資料集添加 REM 方法後的最佳慣性常數即為 0.6。

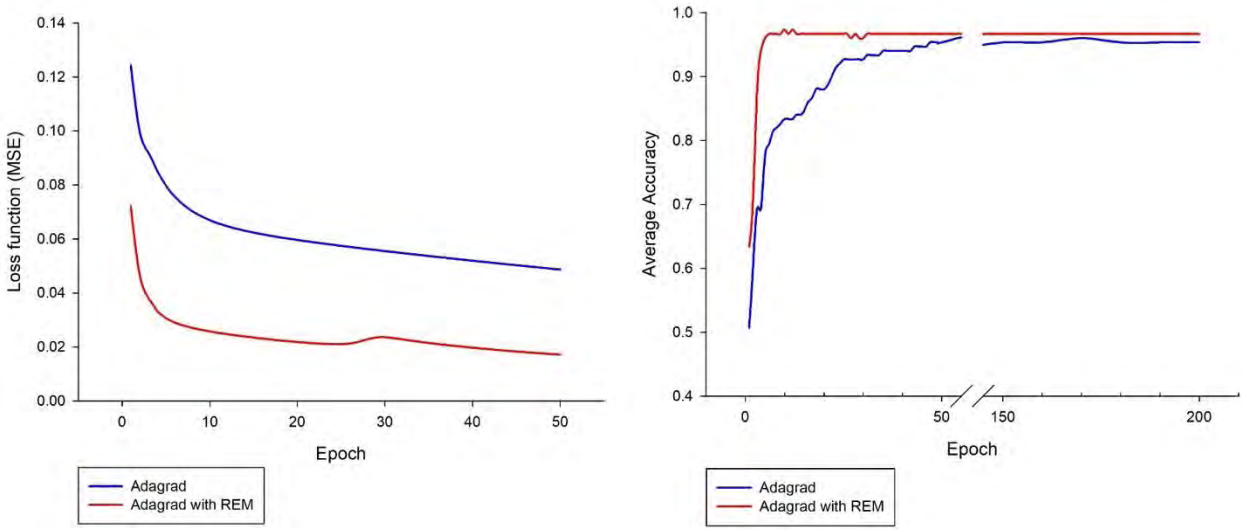
由 Momentum 的公式可知：慣性常數越大就代表對上次訓練越重視，在等高線上的滑動也會越強烈；慣性常數越小則反之，並且當慣性常數設為零，此時的 Momentum 優化法同義於傳統梯度下降法。因此，應該是 Momentum 的算法特性與 REM 方法相容性的問題，前者在等高線上的滑動，並不適於再配合損失函數的變形，否則將會讓滑動的效果變差。



（圖 14 梯度下降法俯視圖）

4. 以 Adagrad 為優化法的訓練進程變化分析

最後，以自適應學習率的 Adagrad 優化法與 REM 方法進行測試。Adagrad 的速度是優於前兩者的，訓練過程也很穩定。若添加 REM 方法後能夠使原始的 Adagrad 有更好的效果，便能證明，自適應學習率與自適應損失函數的相互影響能創造出更快速的訓練進程。



（左圖 15 （Adagrad）鳶尾花分類資料集之損失函數-訓練週期圖）

（右圖 16 （Adagrad）鳶尾花分類資料集之準確度-訓練週期圖）

圖 15 有以下兩點討論：

（1） 訓練前期的損失函數值比前兩種優化法算出的值更小。

因為 Adagrad 會用歷次訓練產生之梯度平方的和，再開根號所算出來的值，當作當次學習率的分母（分子就是初始學習率），所以它可以對稀疏（即分母小）的參數進行更大幅的更新；對頻繁（即分母大）的參數進行更小量的更新，故其訓練速度與品質會比固定學習率的 SGD 及 Momentum 更好。在使用 REM 方法後，更是會對大部份頻繁參數再進一步地放大，而對大部份稀疏參數則小量放大——之所以說大部份，是因為 Adagrad 與 REM 方法「是否增幅訓練」的標準只能說有某種程度上的關聯，但又顯然並不完全相同：前者是分母是否大於一，後者是當前 REM 方法是否大於閾值，兩者不一定會同時觸發。不過以圖 15 來看，兩者結合的效果是良好的，訓練更加快速又不失其穩定。

（2） 在訓練初期，添加 REM 方法的損失函數值反而小於未添加的。

與 SGD 相同，若添加後訓練狀況極好的話，就會有此種情況，因此這項實驗成功由損失函數值的視角證實確實能優化訓練效果。

圖 16 有以下兩點討論：

（1） 訓練過程中的準確率與前兩種優化法相比都高上許多、訓練強健性很高，而以添加 REM 方法後的 Adagrad 優化法為最佳。

Adagrad 一般會是三者中效果最好的優化法（不過並非絕對，在不同網路模型與資料集中可能會有不同狀況）。由圖 16 可從準確率的視角得證自適應學習率與變形損失函數的相容性，即自適應學習率與自適應損失函數可以兼容的第二個證明。

（2） 使用原始的 Adagrad 訓練至第 200 個週期後，其準確率始終也達不到使用 REM 方法後的。

Adagrad 的衰減項存在著「分母單調遞增」的缺點存在，亦即，學習率只會越來越小而不會變大，因此到訓練後期的權重更新幾乎沒有辦法有足量的改變，準確率也就難以再上升了；而添加 REM 方法後，可視為在訓練時添加了另一項加速權重更新的能量，因此在前期自適應學習率尚未衰減到極小值之前，就已經到達訓練終點。故以實驗結果而言，數學項能夠變相彌補 Adagrad 優化法的此缺點。

5. ANCOVA 檢定結果

（1） 訓練週期為 50 的準確率分析摘要：$F(5, 293) = 41.904, p = 0.000$

（2） 訓練週期為 50 的準確率分析：描述性統計資料

（表 5 鳶尾花分類資料集之各演算法平均準確度）

演算法	平均數	標準偏差
1	0.69093	0.22642
2	0.91480	0.08792
3	0.92360	0.15504
4	0.87893	0.10673
5	0.95213	0.09305
6	0.87575	0.06024

（3） 訓練週期為 50 的準確率分析：組間 p 值

（表 6 鳶尾花分類資料集之各演算法間顯著性）

	1	2	3	4	5	6
1		0.000	0.000	0.000	0.000	0.000
2	0.000		1.000	1.000	1.000	1.000
3	0.000	1.000		1.000	1.000	0.076
4	0.000	1.000	1.000		0.456	1.000
5	0.000	1.000	1.000	0.456		0.006
6	0.000	1.000	0.076	1.000	0.006	

（二） 「是否捐贈血液」分類資料集

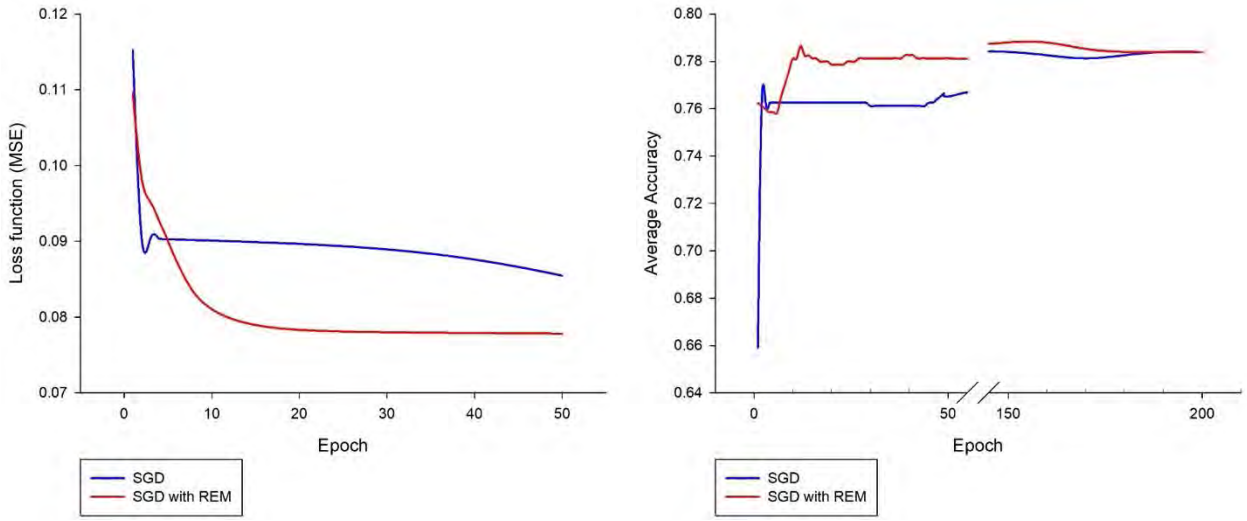
1. 資料集簡介

該資料集的內容很特別，它以血液捐贈者的先前資料，判斷那人在某時捐血的機率。進一步說 ，它是在「預測人類的動向（捐血與否）」，並且以某人的「捐血做次數」、「捐血總

毫升數」、「第一次捐血至今時長」、「上次捐血距今時長」為四個特徵值。本研究推測它的特徵值在類神經網路中的分類器訓練中並非特別良好，不過經討論後仍決定要以此資料集為測試，畢竟它是由大學機構所提供，也有在其他論文被引用過，有一定的參考性。

有 745 筆資料，4 個特徵值，分為「在 2007 年 3 月是否有捐血」2 類。

2. 以 SGD 為優化法的訓練進程變化分析



（左圖 17 （SGD）血液分類資料集之損失函數-訓練週期圖）

（右圖 18 （SGD）血液分類資料集之準確度-訓練週期圖）

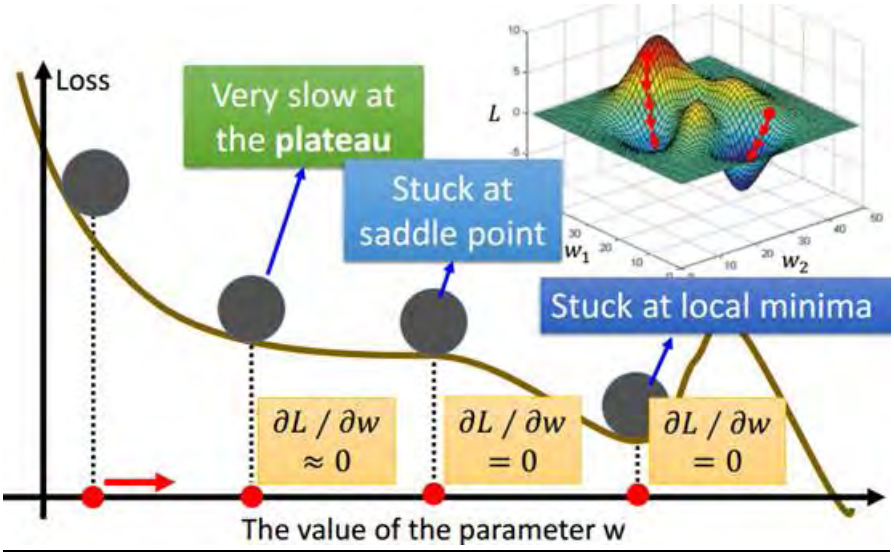
SGD 似乎因為其僵硬的訓練方式，使得類神經網路訓練過程中的資料處理與數據變化有些固定的表徵，資料集測試與數據分析中的數個函數圖顯示，SGD 優化法的損失函數值在訓練初期有時候會呈現陡峭的降低。這應當不是損失函數形式的關係，因為在使用 Momentum 跟 Adagrad 時較少出現這個情況；本研究先前曾實測過未使用 REM 方法的 SGD，測試過其他未在本報告中提及的資料集，其損失函數值-訓練週期圖幾乎皆呈現陡峭下降的狀態。

雖然添加 REM 方法後的確是能達到較好的訓練效果，不過該資料集在使用 SGD 優化法時，其訓練後的準確度始終無法突破 80%，就算將訓練週期提升至 200 個也一樣，且該資料集並不複雜，單層隱藏層就足以處理。該訓練問題的產生有兩種可能：

（1） 該資料集在建構時並未選取適當的特徵值

因為本研究是直接取用該資料集，所以無法確知該資料集在特徵工程（feature engineering）階段的處理。不過，就算已經過特徵選擇（feature selection）這一工序，本研究仍推斷這些特徵值的並不是該資料集的最佳選擇，如「第一次捐血至今時長」該特徵值就可直觀地發現它無法良好地切割不同的標籤。而若該資料集的特徵值無法讓類神經網路完美執行分類，則不管使用何種優化法，基本上模型準確率都會有其一定的上限，因此在 Momentum、Adagrad 的再確認後，便可分析出是否是資料集特徵的缺陷。

（2） 訓練落入了局部最小值。



（圖 19 局部最小值示意圖）

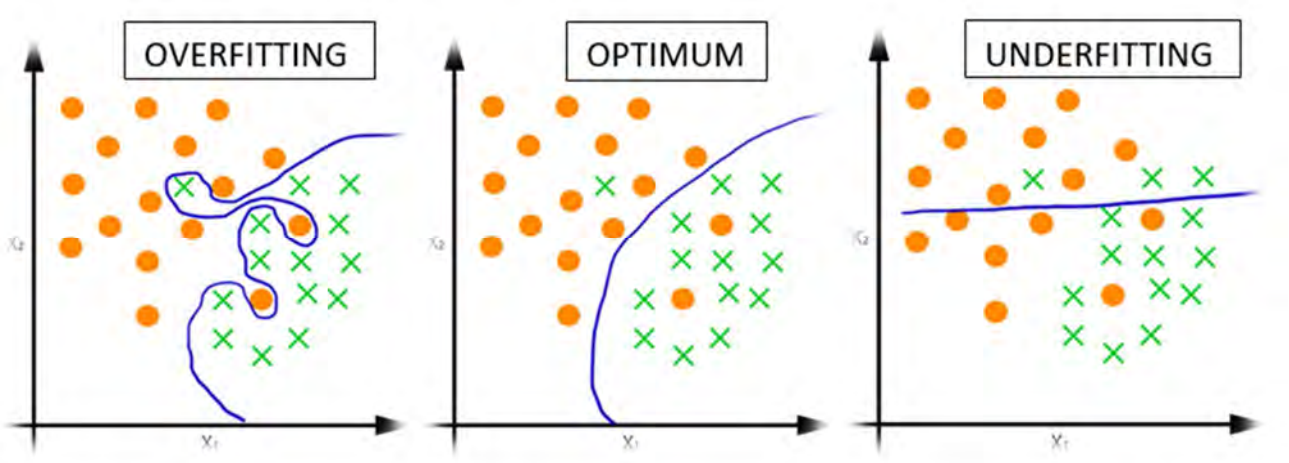
在梯度為零或近乎為零的狀況下，權重更新的效果相當有限。此時的類神經網路未必已找到全局最小值，而有可能落在圖 19 的三個區域。由圖 18 可見準確率在小範圍內波動，且落入局部最小值是傳統梯度下降法經常遇到的問題，故初步推測落入局部最小值的機率較高。之後將測試能脫離局部最小值的 Momentum 優化法，可經數據分析推斷此為特徵值或局部最小值的問題。

（3） 過度擬合問題（overfitting）

若網路模型相較於使用的資料集而言太過複雜、也就是設置太多神經元與隱藏層的話，將會使訓練出的變數組合得以擬合所有訓練資料，此時的訓練準確率極高，但因為該變數組合無法良好判斷其餘未參與訓練進程的資料點，所以將使測試

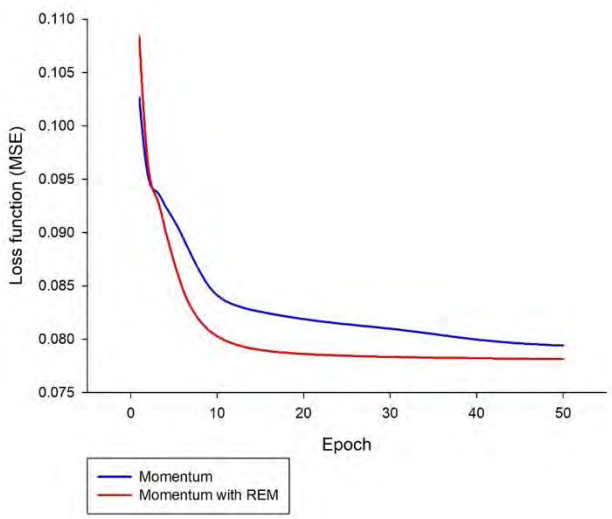
準確度低落許多，未來在實務上應用的效果也將不彰。可以使用正則項或是 dropout 方法來解決此問題。

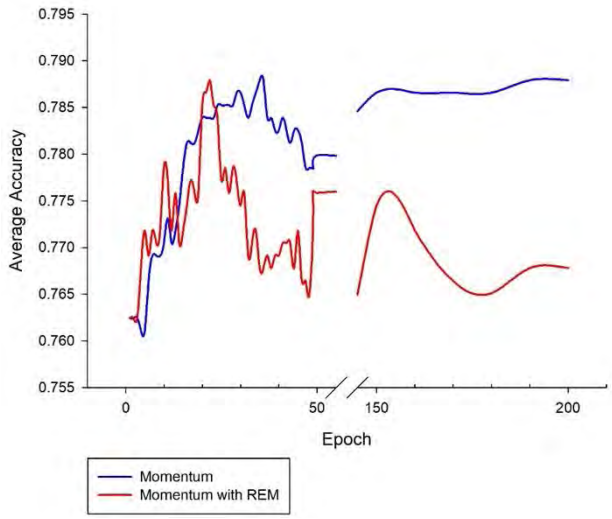
但是，本研究所使用的網路模型並不龐大，沒有導致過度擬合的前提，在紀錄訓練準確率之後，也證實並無發生該問題。至於模型過於簡單造成的擬合不足（underfitting）問題，更無發生的可能。



（圖 20 擬合狀況示意圖）

3. 以 Momentum 為優化法的訓練進程變化分析





（左圖 21 （動量法）血液分類資料集之損失函數-訓練週期圖）

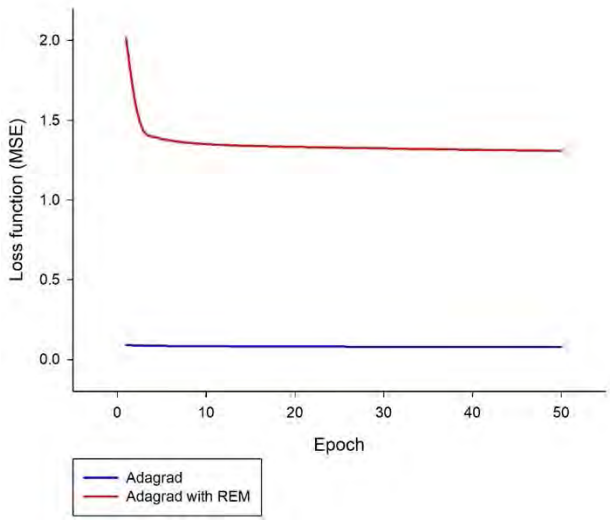
（右圖 22 （動量法）血液分類資料集之準確度-訓練週期圖）

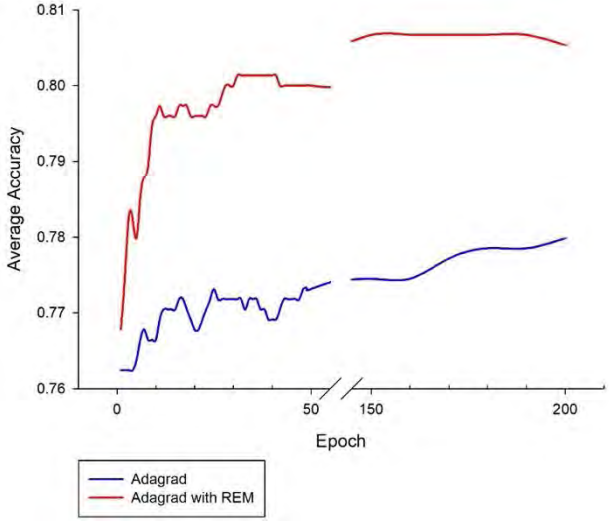
圖 22 指向兩個結論：

（1） Momentum 不論有無添加 REM 方法，準確率也都無法突破到 80%，因此推翻了訓練落入局部最小值的可能。

（2） 雖然添加 REM 方法後，訓練強健性在訓練週期到 50 次之前有略微的提升，不過兩者的訓練品質都不算好，再次驗證了動量法與 REM 方法相容性低的論點。

4. 以 Adagrad 為優化法的訓練進程變化分析





（左圖 23 （Adagrad）血液分類資料集之損失函數-訓練週期圖）

（右圖 24 （動量法）血液分類資料集之準確度-訓練週期圖）

以 Adagrad 訓練該資料集時，數學項的衰減常數最佳值為零，此時數學項恆大於閾值，所以對損失函數的增幅就不會受訓練週期的影響而降低或終結。這是非常特殊的狀況，因為此時兩種自適應機制的交互作用是消失的，亦即初始學習率只需乘上一由數學項決定後的定值即可達到最佳訓練效果。本研究認為衰減常數為零是數學項的特例，此時代表初始學習率需再調整。

觀察準確率的改變，訓練進程被大幅度優化。即使如此，在前 200 個訓練週期中，三種優化法、六種訓練結構下的準確率最高也只有 Adagrad 配合 REM 方法的使用能達到 80%。本研究推測該資料集的特徵值並不利於類神經網路分類器的訓練，或許此結果已達本研究中使用的類神經網路架構之極限。

5. ANCOVA 檢定結果

（1） 訓練週期為 50 的準確率分析摘要：$F(5, 293) = 109.702, p = 0.000$

（2） 訓練週期為 50 的準確率分析：描述性統計資料

（表 7 血液分類資料集之各演算法平均準確度）

演算法	平均數	標準偏差
1	0.76016	0.01463
2	0.77785	0.00742
3	0.77884	0.00790
4	0.78117	0.00834
5	0.76974	0.00298
6	0.79597	0.00733

（3） 訓練週期為 50 的準確率分析：組間 p 值

（表 8 血液分類資料集之各演算法間顯著性）

	1	2	3	4	5	6
1		0.000	0.000	0.000	0.000	0.000
2	0.000		1.000	0.400	0.000	0.000
3	0.000	1.000		1.000	0.000	0.000
4	0.000	0.400	1.000		0.000	0.000
5	0.000	0.000	0.000	0.000		0.000
6	0.000	0.000	0.000	0.000	0.000	

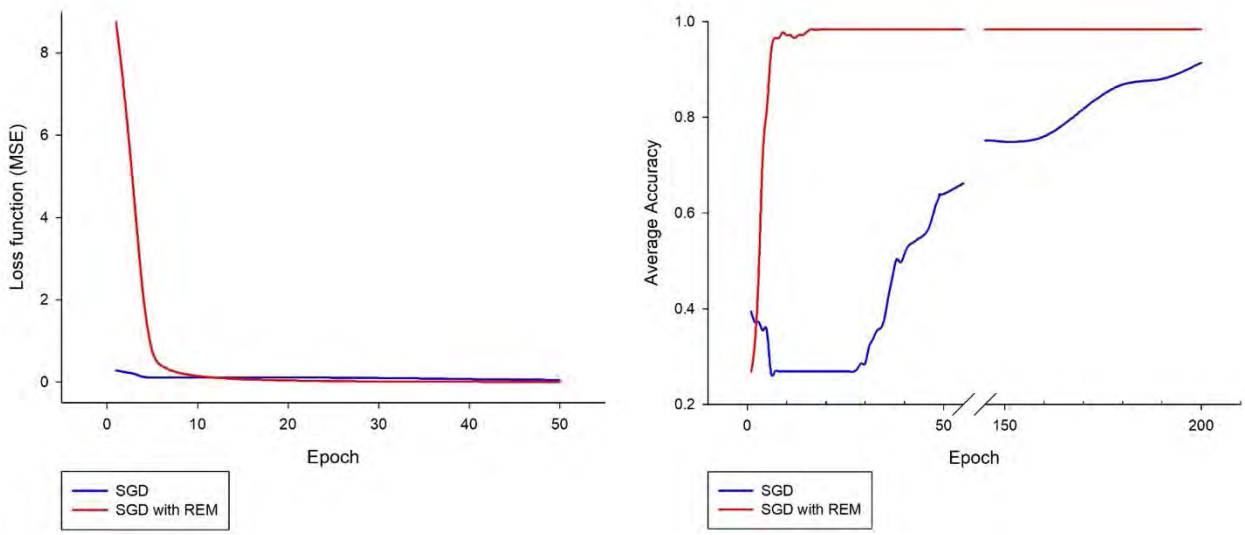
（三） 酒分類資料集

1. 資料集簡介

在第三次實驗測試中，特意選擇了這個數據量小、特徵值卻很多的經典資料集，它以酒精含量與一些化學性質來判斷酒類。本研究接著要測試數學項在數學視角上對高維梯度向量增幅後能對梯度下降提供的效果及在實務上對離散資料集的效果。

有 175 筆資料，13 個特徵值，分為 3 種酒類。

2. 以 SGD 為優化法的訓練進程變化分析



（左圖 25 （SGD）酒分類資料集之損失函數-訓練週期圖）

（右圖 26 （動量法）血液分類資料集之準確度-訓練週期圖）

該資料集使用 SGD 後得到的增幅十分強烈，在訓練初始的損失函數值被放大到 8、9 左右。如此大的值在正常的 MSE 幾乎不會出現，因而是數學項進行了巨量的增幅。

以下討論兩點：

（1） 為什麼未使用 REM 方法的 SGD 優化法效率並不好？

傳統梯度下降法的弊病在此資料集表現得十分明顯。多變數、少數據點構成的超維函數圖形會十分崎嶇，讓以當前梯度為權重更新唯一標竿的 SGD 窒礙難行，就像一步一步艱困的走下崎嶇山麓一樣。除了沒效率之外，也極可能落入局部最小值，因為這類圖形的臨界點非常多。而先前的兩個資料都較為平穩，因為特徵值少且數據品質佳，因此觀察不到這種狀況。

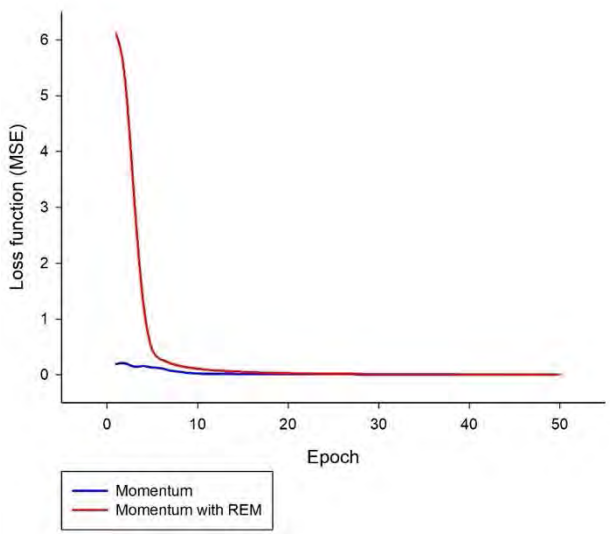
（2） 為什麼使用 REM 方法後就能克服原本的問題？

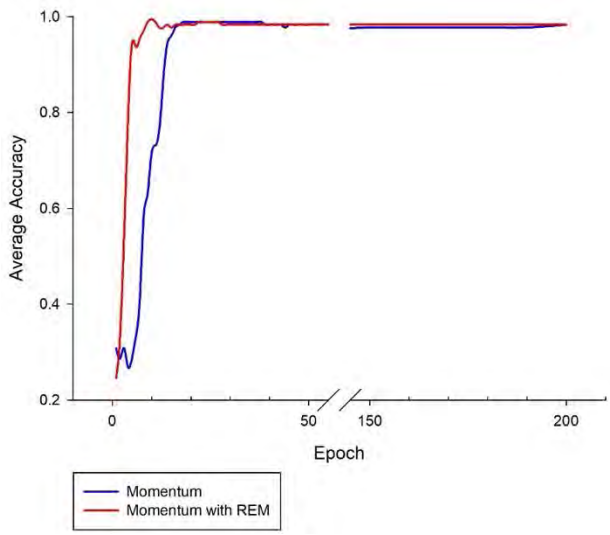
從先前算出的傳統梯度下降法公式變形中，可知若是把附加於微分後的損失函數的數學項，移到學習率之前，那麼就等於為使用使用固定學習率的 SGD 優化法（SGD 也可以設置衰減的學習率）提供衰減項了，而自適應學習率的表現通常都很

優秀，因為固定學習率會使得權重更新就算在前期也無法大幅更新，梯度只能緩慢下降。

因此，使用固定學習率的 SGD 優化法可以說是本研究未來應用的一個特例，它同義於動態改變學習率的大小。不過現今幾乎沒有在使用固定的學習率，許多實驗早已證實它的諸多弊病。

3. 以 Momentum 為優化法的訓練進程變化分析





（左圖 27 （動量法）酒分類資料集之損失函數-訓練週期圖）

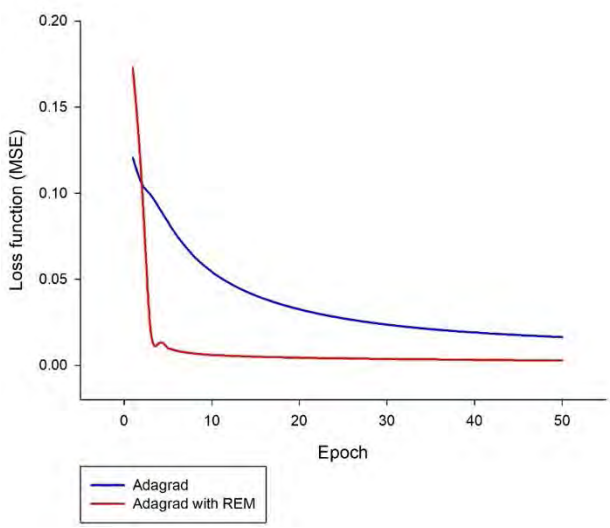
（右圖 28 （動量法）酒分類資料集之準確度-訓練週期圖）

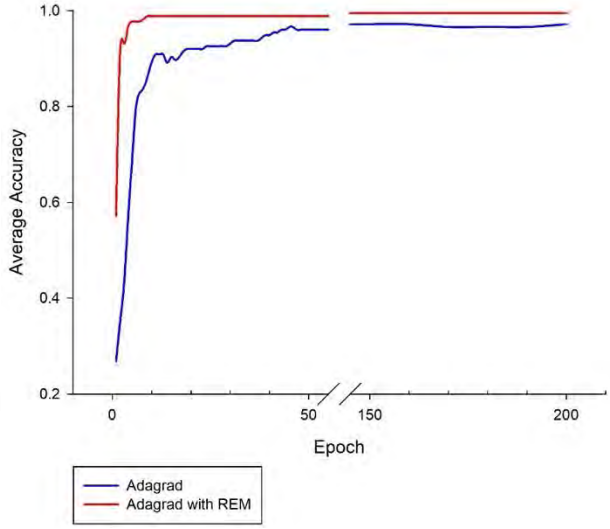
圖 27 中的損失函數值略微超過了 6，是數據分析中目前遇到第二大的被增幅值，可以明顯看出為人為增幅的痕跡。

相較前兩個資料集，該資料集中使用 Momentum 並沒有優化訓練品質，而由其與圖 25（SGD）的比較，可知在稀疏資料集中，SGD 與 Momentum 的表現都不太好。

在添加 REM 方法後，雖然並無自適應學習率與 REM 方法對於學習率之值的交互作用，但整體訓練效果仍優秀許多，這與前兩個資料集得到的結果是相悖的，不過或許是因為該資料集存在著一個顯著的特性：數據離散。因此本研究推測，雖然 Momentum 在一般的情況下與 REM 方法的相容性普通，但在「Momentum 表現不好的訓練進程中」，添加 REM 方法依舊可以改善其訓練強健性，而目前測得的可能，就是將 Momentum 應用於稀疏資料集時。

4. 以 Adagrad 為優化法的訓練進程變化分析





（左圖 29 （Adagrad）酒分類資料集之損失函數-訓練週期圖）

（右圖 30 （Adagrad）酒分類資料集之準確度-訓練週期圖）

該稀疏資料集中，Adagrad 的訓練進程比 SGD 好許多，並比 Momentum 慢些，但函數曲線是最平滑的。Dean, J.等（2012）[9]提到了 Adagrad 除了極大地提高 SGD 的強健性外，並且也適合處理稀疏數據，圖 30 與前兩個優化法的分析顯然支持該論文的論點，也能再次證明 REM 方法的添加能讓 Adagrad 在稀疏數據中擁有更好的發揮，速度更快、不失穩定。

5. ANCOVA 檢定結果

（1） 訓練週期為 50 的準確率分析摘要：$F(5, 293) = 157.070, p = 0.000$

（2） 訓練週期為 50 的準確率分析：描述性統計資料

（表 9 酒分類資料集之各演算法平均準確度）

演算法	平均數	標準偏差
1	0.37062	0.12497
2	0.93405	0.15351
3	0.85794	0.24128
4	0.94034	0.15142

5	0.87645	0.15269
6	0.97622	0.06010

（3） 訓練週期為 50 的準確率分析：組間 p 值

（表 10 酒分類資料集之各演算法間顯著性）

	1	2	3	4	5	6
1		0.000	0.000	0.000	0.000	0.000
2	0.000		0.049	1.000	0.382	1.000
3	0.000	0.049		0.022	1.000	0.000
4	0.000	1.000	0.022		0.199	1.000
5	0.000	0.382	1.000	0.199		0.002
6	0.000	1.000	0.000	1.000	0.002	

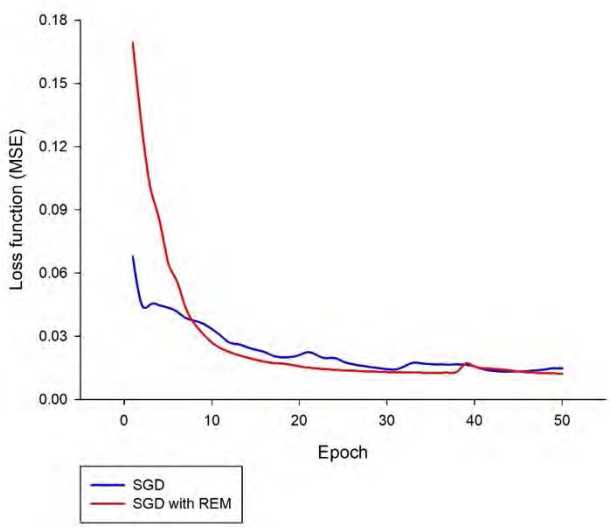
（四） 手寫字分類資料集

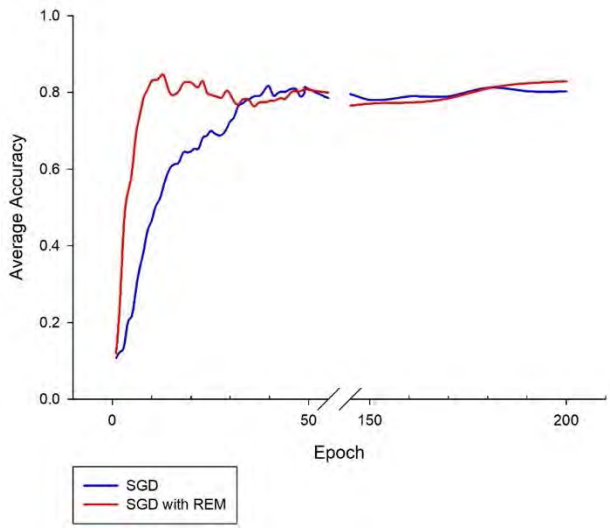
1. 資料集簡介

因為本研究並非使用卷積神經網路，所以無法運用 MNIST 經典的手寫字圖像資料集，不過依舊有適合全連接類神經網路的手寫字分類資料集可供本研究使用。該資料集能測試 REM 方法對使用較大量數據之訓練進程的優化效果。

有 10990 筆資料，16 個特徵值，手寫字為數字 0~9。

2. 以 SGD 為優化法的訓練進程變化分析





（左圖 31 （SGD）手寫字分類資料集之損失函數-訓練週期圖）

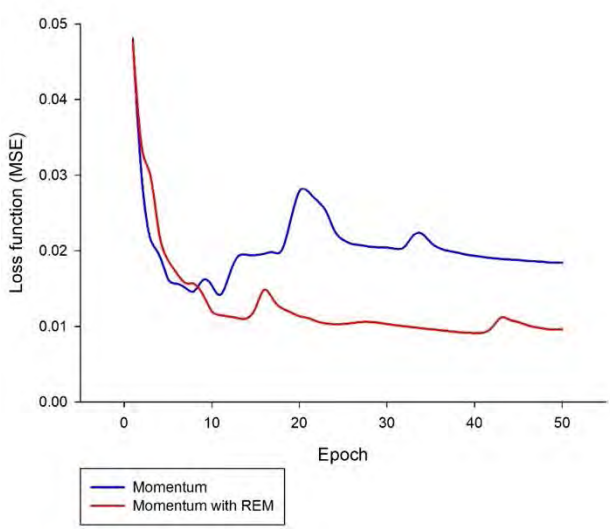
（右圖 32 （動量法）酒分類資料集之準確度-訓練週期圖）

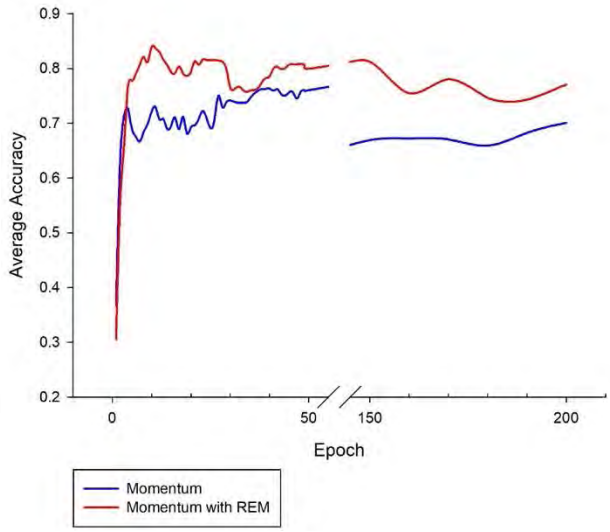
雖然該資料集特徵值、標籤多，但因為其數據量在小型類神經網路中已算是龐大，因此不會有前一個資料集的稀疏問題。圖 32 可展現 REM 方法在大量資料集的訓練進程中，其加速訓練的效果。

添加 REM 方法後的訓練終點提前許多，這是因為傳統梯度下降法的訓練速度緩慢。紅色線在近第 20 個訓練週期、藍色線在第 40 個訓練週期時開始一定幅度的準確率震盪，推測是因為固定學習率的關係。雖然本研究曾提及，REM 方法在配合 SGD 或 Momentum 時是可以視為被加諸於學習率之前的，不過因為數學項恆大於一的特性，所以其並無降低學習率的功能，意即，在訓練後期可能遇到的震盪問題並不會被解決。

不過，本研究認為該資料集的準確率應可再提升，同對血液分類資料集的嘗試，除了實測三種優化法與 REM 方法的配合之外，也將探討其他可能遇到的訓練問題。

3. 以 Momentum 為優化法的訓練進程變化分析



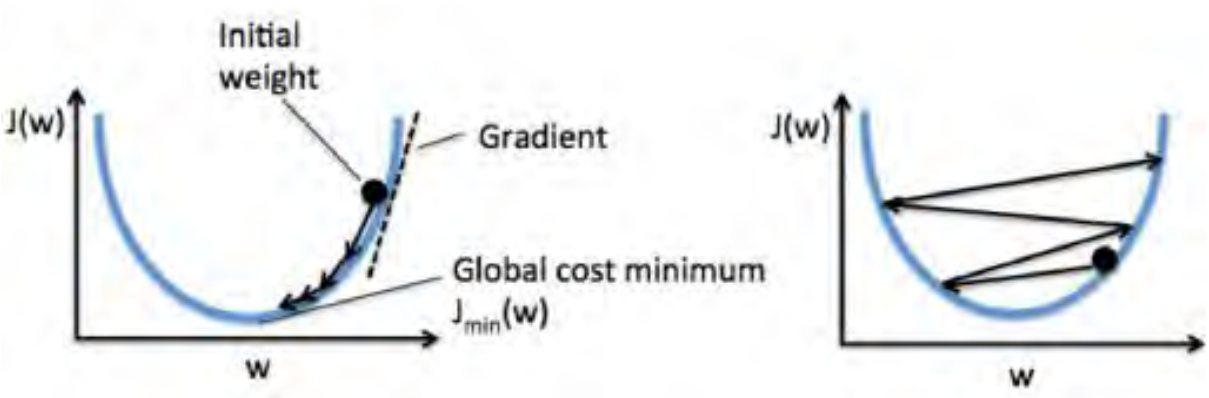


（左圖 33 （動量法）手寫字分類資料集之損失函數-訓練週期圖）

（右圖 34 （動量法）酒分類資料集之準確度-訓練週期圖）

由圖 33 可發現，未添加 REM 方法前，損失函數值在第 1~20 個訓練週期間形成了一個凹槽，這可能是短暫的 overshoot 現象，會略微降低神經網路的學習效率。

若以梯度下降常用的「走下山」為比喻，可以通俗的解釋為：登山時因為步伐邁得太大，而從東側山麓直接踏到了西側山麓，且海拔位置還更高。推測是因為 Momentum 較為激烈的更新所造成，而 REM 方法的添加可穩定地加速訓練進程，所以就可變相消除 overshoot 現象了。不過本研究認為，若數學項的超參數若設置不當，也可能會造成該現象的發生。

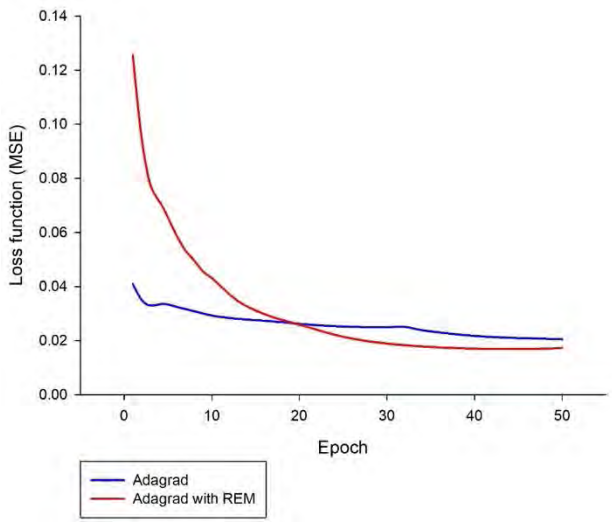


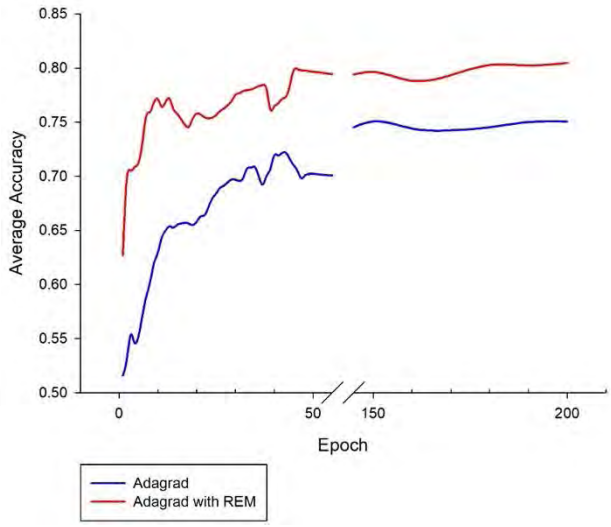
（圖 35 overshoot 示意圖）

再以準確率變化的視角分析。Momentum 的訓練速度快，即使未添加 REM 方法，在第 4 個訓練週期已有極高的準確率；添加後更能提高整體訓練效果，不過準確率的峰值仍不如預期。

本研究也在該資料集分析並測試了不同種可能遇到的訓練問題，但最後的過擬合現象也因為 training accuracy 的紀錄而證明其並未發生，所以僅有固定學習率造成的訓練震盪可以解釋，且由於 Momentum 使用了固定學習率，所以該現象的發生也有其根據，不過仍須再透過 Adagrad 使用來確認是震盪問題、或已是該類神經網路架構下的極限。

4. 以 Adagrad 為優化法的訓練進程變化分析





（左圖 36 （Adagrad）手寫字分類資料集之損失函數-訓練週期圖）

（右圖 37 （動量法）酒分類資料集之準確度-訓練週期圖）

未添加 REM 方法前的 Adagrad 非常不適合運用在該資料集中，可由其公式得出原因：分母在每一次訓練週期都得到大幅提升，因此訓練速度越趨緩慢，但此時仍離訓練終點遙遠。理論上夠量的訓練週期仍能使該優化法得到良好的收斂值，不過那顯然非常不符合時間成本，而 REM 方法的添加將會與衰減項共同決定權重更新的步伐，圖 37 可展示該影響對訓練進程產生的正向效果。

添加 REM 方法後對整體強健性的增幅明顯可見，且也可知在大量資料集中，REM 方法無論從長期或短期的訓練而言，都適用於任一種特性的優化法。準確率的峰值為 85%，本研

究認為，在該類神經網路的架構之下，若不建立多層隱藏層配合正則項，這已是最好的結果。

5. ANCOVA 檢定結果

（1） 訓練週期為 50 的準確率分析摘要：$F(5, 293) = 21.252, p = 0.000$

（2） 訓練週期為 50 的準確率分析：描述性統計資料

（表 11 手寫字分類資料集之各演算法平均準確度）

演算法	平均數	標準偏差
1	0.62713	0.20503
2	0.75458	0.13694
3	0.71656	0.05938
4	0.77903	0.08162
5	0.66412	0.05421
6	0.76151	0.03019

（3） 訓練週期為 50 的準確率分析：組間 p 值

（表 12 手寫字分類資料集之各演算法間顯著性）

	1	2	3	4	5	6
1		0.000	0.000	0.000	0.691	0.000
2	0.000		0.606	1.000	0.000	1.000
3	0.000	0.606		0.012	0.072	0.233
4	0.000	1.000	0.012		0.000	1.000
5	0.691	0.000	0.072	0.000		0.000
6	0.000	1.000	0.233	1.000	0.000	

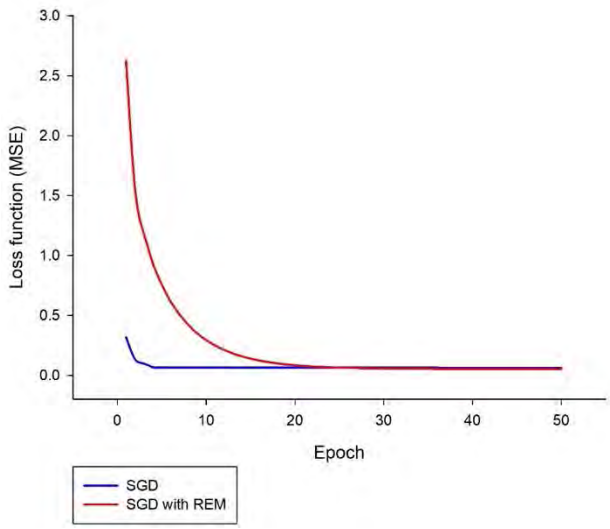
（五） 玻璃分類資料集

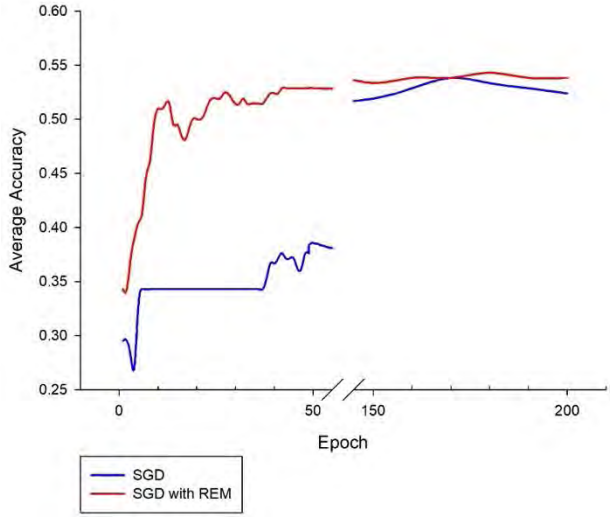
1. 資料集簡介

最後要測試的資料集極度稀疏，標籤（label）有 6 種。經過實測，該資料集的準確率無法達到一般資料集應有的正常值，就算使用兩層隱藏層也無法改善，這應該要歸咎於過度離散的資料點。雖然這類資料集在深度學習中的應用價值不大，但本研究依舊決定在數據分析的最後一個部分選擇它，分析在類神經網路感到乏力的訓練進程中，損失函數的變形對訓練效果之增益。

有 210 筆資料，9 個特徵值，玻璃種類被分為 6 類。

2. 以 SGD 為優化法的訓練進程變化分析





（左圖 38 （SGD）玻璃分類資料集之損失函數-訓練週期圖）

（右圖 39 （動量法）酒分類資料集之準確度-訓練週期圖）

圖 39 有以下兩點討論：

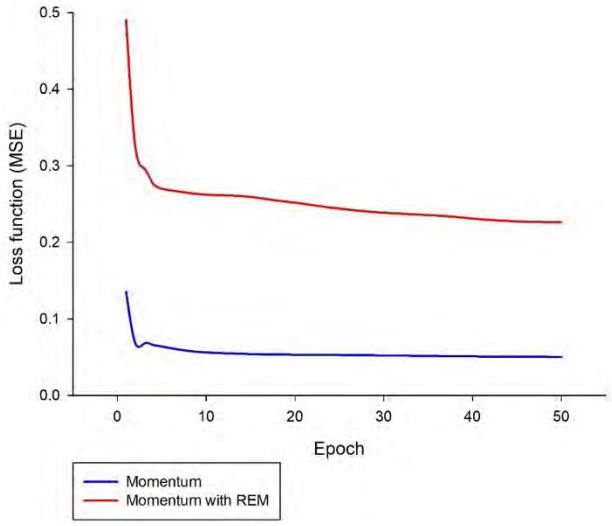
（1） 不論有無添加 REM 方法，整體的準確率都是五個資料集中最低的。

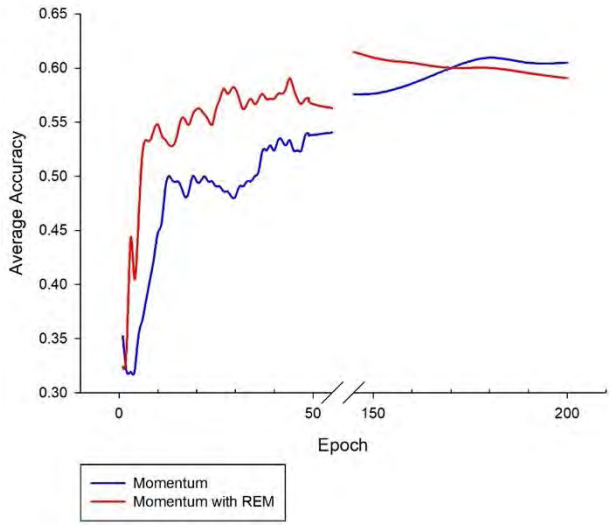
在資料集簡介中就有提及，該資料集的數據量不僅少且標籤多，那麼每個標籤對應的數據量就更少了，因此神經網路無法抓取到足夠多的特徵去判斷，所以準確率雖然高於平均機率六分之一，不過卻也不甚理想。

（2） 添加 REM 方法後，訓練強健性提升許多。

在前 50 個訓練週期，未使用 REM 方法的 SGD 顯得很乏力；則使用 REM 方法後獲得的權重更新的新推力，就會發揮出很大的功效了。前文也曾提及，使用固定學習率的 SGD 優化法是本研究的特例，因為權重更新的偏微分數學公式在實際展開成代數式後，數學項實際上就能提供類似於指數衰減學習率的效果了，而自適應學習率對訓練的增益早已被證實。

3. 以 Momentum 為優化法的訓練進程變化分析



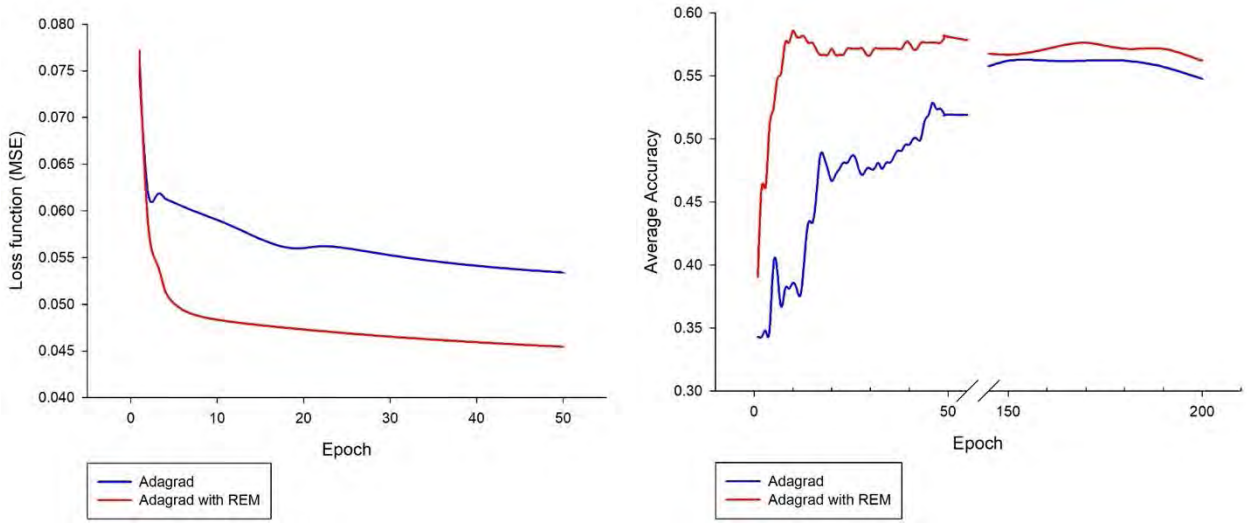


（左圖 40 （動量法）玻璃分類資料集之損失函數-訓練週期圖）

（右圖 41 （動量法）酒分類資料集之準確度-訓練週期圖）

由圖 41 可知，雖然在第 170 個訓練週期後的準確率略低，但整體的訓練效果與準確率的峰值仍較不添加 REM 方法為好。這也可印證本研究先前提出的一個論點：Momentum 的算法特性使其在一般情況下與 REM 方法的相容性不算好，但在 Momentum 發揮得較差的訓練進程中（目前已知有稀疏資料集），REM 方法的添加卻能賦予它大幅的增益。

4. 以 Adagrad 為優化法的訓練進程變化分析



（左圖 42 （Adagrad）玻璃分類資料集之損失函數-訓練週期圖）

（右圖 43 （動量法）酒分類資料集之準確度-訓練週期圖）

雖然 Adagrad 一樣無法很好地處理該資料集，不過添加 REM 方法後的訓練效果依舊好上許多，本研究提出的數學項（損失函數的變形）與自適應學習率，對權重更新幅度形成的交互作用在此處依舊成立。也就是說，就算原始的 Adagrad 在資料集中表現不佳，交互作用也可以正常發揮，不會因為自適應學習率的發揮不佳，而喪失對類神經網路訓練強健性的增益，這也符合生物睡眠中 REM 時期該有的效果。

5. ANCOVA 檢定結果

（1） 訓練週期為 50 的準確率分析摘要：$F(5, 293) = 261.862, p = 0.000$

（2） 訓練週期為 50 的準確率分析：描述性統計資料

（表 13 玻璃分類資料集之各演算法平均準確度）

演算法	平均數	標準偏差
1	0.34504	0.02204
2	0.49552	0.04799
3	0.47742	0.06091
4	0.54419	0.05573

5	0.45733	0.05490
6	0.56200	0.03511

（3） 訓練週期為 50 的準確率分析：組間 p 值

（表 14 玻璃分類資料集之各演算法間顯著性）

	1	2	3	4	5	6
1		0.000	0.000	0.000	0.000	0.000
2	0.000		0.080	0.000	0.000	0.000
3	0.000	0.080		0.000	0.030	0.000
4	0.000	0.000	0.000		0.000	0.113
5	0.000	0.000	0.030	0.000		0.000
6	0.000	0.000	0.000	0.113	0.000	

（六） 數學項（Z）之超參數特性探討

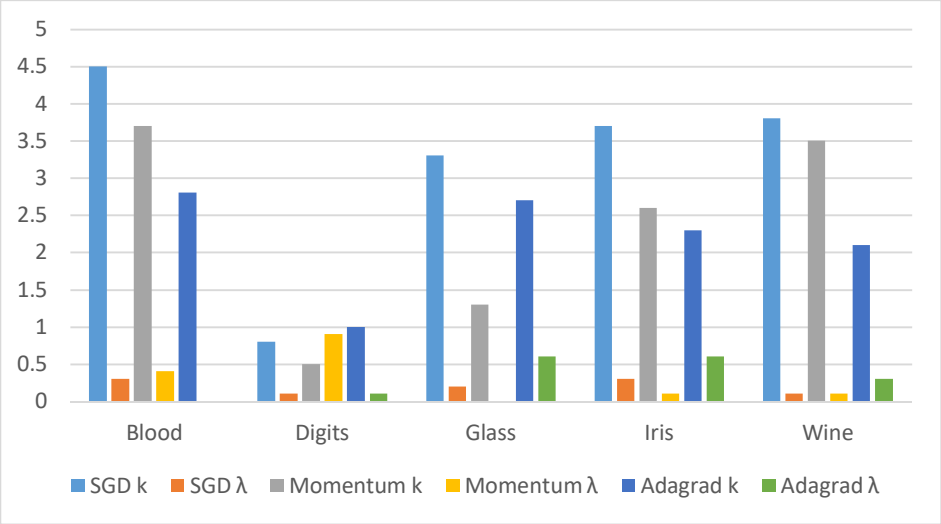
超參數是沒有絕對準則、須依據經驗法則或一些特殊方法得知其較佳套用數值的參數，如學習率、神經元層數、慣性常數等皆是。本研究為了提升 REM 方法在應用上之價值，故分析並探討其數學項中兩個超參數（k、lambda）對於不同種資料集、優化法的特性。

（表 15 最佳超參數組合）

	SGD		Momentum			Adagrad	
	k	λ	k	λ	m	k	λ
Blood	4.5	0.3	3.7	0.4	0.5	2.8	0.0
Digits	0.8	0.1	0.5	0.9	0.6	1.0	0.1
Glass	3.3	0.2	1.3	0.0	0.8	2.7	0.6
Iris	3.7	0.3	2.6	0.1	0.6	2.3	0.6
Wine	3.8	0.1	3.5	0.1	0.3	2.1	0.3

將超參數整理成表。將 Momentum 的慣性常數（代號為 m）也考慮進來，可發現全部都低於未使用 REM 方法前一致的 0.9。並且，較顯然的表徵尚有「k 幾乎大於 λ」，先前文獻的搜集僅得知 λ 應是一個常小於 1 的衰減常數，列表後可清楚得知 k 這一個與 λ 相互制衡的

超參數應是，以一較大的值牽制 λ 與訓練週期的乘積—當然，因為訓練週期呈現單調遞增，所以數學項終究會不再被啟動。



（圖 44 最佳超參數組合）

若以資料集特性區分，可知唯一一個大量資料集對應的兩個超參數都明顯較小，而其他相對而言較為稀疏的四個資料集則較為相近。當較小的最佳值同時出現於兩個超參數中，一般代表數學項傾向於「小幅但不短暫」的增幅效果；而本研究中大多數的增幅情形則是「大幅但不短暫」的增幅效果，這是非巨量資料集最有可能的最佳解情況—因為資料點較少，所以單次訓練週期進行的迭代次數並不多（即數學項的增幅次數少），而數學項的適用時間就會隨之延長了。

因此，可以推測，若把本研究之訓練方法全數改為以小批量形式處理，所測得 λ 之最佳值就會降低，以延長數學項的作用時間。

若以優化法特性區分，SGD 的 k 值都是三種優化法中的最大，數學項作用時間也都不短，本研究推測這或許與 SGD 被雜訊干擾而顯得較盲目的梯度下降，及較為單調的權重更新方式有關；Momentum 與 Adagrad 則無明顯表徵。

總而言之，本研究推測 k 與 λ 的最佳值會受資料集離散程度，與優化法的權重更新方式之複雜度影響。這兩個超參數的比值與值的大小會直接影響 REM 方法的作用時間與增幅強度，應用時可並先評估一個適當的初始增幅值來決定 k 值，再依該資料集的訓練進程前中期時約略的訓練週期，推算這兩個超參數間的比例關係，進而決定衰減常數 λ 。

（七） 資料集測試與數據分析重點條列

1. 自適應損失函數是可行的，本研究提出的 REM 方法即大幅增加了類神經網路訓練時期的強健性。
2. 損失函數值的下降會更加平滑，因為數學項是採用指數衰減的形式。
3. 在小型、稀疏、巨量、特徵多寡等特性各不相同的資料集中，該演算法的表現都十分良好。
4. 在 SGD、Momentum、Adagrad 三種參數優化法中，SGD 與 Adagrad 一般都跟 REM 方法有很高的相容性，Momentum 則差些，但 Momentum 在離散資料集中與 REM 方法的配合將使其效果優於使用 REM 方法的 Adagrad。
5. Momentum 在使用 REM 方法後，最佳的慣性常數會降低，因為數學項的增幅不支持 Momentum 在等高線上的滑動。
6. REM 方法的添加能提供訓練時的另一能量，而變相改善 Adagrad 學習率的分母因為恆常衰減，而造成訓練中後期更新緩慢的缺點。
7. 若以「下山」作為比喻，則 REM 方法也可以理解為，提供一個類似「初始加速度」的效果，這在問題與討論中會詳論。
8. 損失函數變形與自適應學習率的相容性高，但效力的發揮不一定同步（可能一個發揮得很好一個發揮得不好）。
9. 兩個超參數可透過選用的資料集以及優化法推得一較佳的解，即其受資料集疏密與優化法算法特性影響。

四、 結論與應用

（一） 總結與應用

本研究從生物睡眠機制中的 REM 時期為發想，提出了自適應損失函數的概念，並設置一套名為 REM 方法的機制，能適當放大正常訓練下的權重更新效果，並且，這套演算法能隨著訓練進程不斷調整自身的值，擁有自適應的能力。

本研究提出之演算法是彈性且高度可控的，它擁有不斷改變的特性和可以讓使用者自行調控的超參數，以在不同類神經網路模型、資料集與優化法中發揮最好的效果。並且從資料集的實測來看，它並不會造成負面效果，而只是單純地改善參數優化的過程。在資料分布平均的良好資料集中，除了準確率在訓練前期就有很高的參考性外，整體訓練的加速進程也會更為明顯；除此之外，調參的過程也能提高 REM 方法的效果。

將來，它可以被應用於使用梯度下降法更新權重的類神經網路，並提升其訓練的強健度，使準確度在初始快速提升。這可以為各學術或應用領域中欲使用類神經網路的單位帶來巨大的方便性，因為不用擔心自身的運算資源不足而無法整合出大型的系統；或得要連續數週的訓練才能完成一個深層網路。

我們在研究階段就飽嘗電腦運算效率讓資料分析進展緩慢之苦，且據新聞提及，許多小型公司與研究者在 AI 時代也因為無法負擔巨量耗時的運算，而只能望洋興嘆。因此更希望本研究能透過開啟一個新的嘗試——損失函數的自適應，來加速類神經網路的訓練效果，為類神經網路的發展盡一份力，讓深度學習這個技術能被廣施於需要用它來解決問題的地方。

（二） 問題與討論

研究中有一些未盡完成的地方或值得再論述的問題、現象，以下一一討論之。

1. 將閾值轉化為自適應形式？

演算法中，決定數學項是否啟動的閾值是否能從定值變本研究設置閾值的目的是為了評估每一次訓練週期的損失函數到底該不該被變形，雖然把閾值設為定值能精準達成只在前期加速進程的目的，但評估的標準就會類似於這次損失函數值對損失函數值-訓練週期圖形之絕對位置；而設置自適應閾值的話就代表本研究的評估標準改變了，從絕對位置變為相對位置，也就是說當下的訓練週期變成會影響閾值的大小。至於這樣的改變是好或不好，仍有待檢驗。

2. REM 方法能與自適應學習率搭配嗎？

Adagrad 即為一種自適應學習率的優化法，而實驗結果顯示其可以與 REM 方法良好地相容（見 三、研究結果與討論）。

原因是雖然兩者同樣為權重更新公式下的運算元之一的變形方法，但這兩種機制其實著重於不同方面，有著不同的功能：學習率的指數衰減是為了讓神經網路在訓練後期能維持平穩的收斂，而本研究提出的演算法是為了增強類神經網路在訓練時期的強健性，以及整體的訓練速度；並且，從數學角度來看，損失函數的變形提供的是其超維圖形梯度的改變，這與學習率的功能也不相同，因此從實驗結果上與理論上來看，REM 方法跟學習率衰減的確能夠互相搭配，形成一個對權重更新增幅值的相互抗衡之作用，為類神經網路模型提供更好的效果。

3. 若把梯度下降比喻為「下山」，則這套演算法起到了什麼作用？

這套演算法也可以比喻為，讓一顆球在滾下山時的前幾秒—離最低點仍然遙遠時—給予它考慮摩擦力的加速度，如此除了能使它在前期的滾動速度變得更快、擁有一定的動能去衝出訓練初期的鞍部（局部最小值）外，加速度的量值也會隨著滾動而遞減，最後趨於零了。

（三） 未來展望

1. 以更嚴謹的數學方法推導該演算法在類神經網路中的影響。
2. 在不同的類神經網路模型上做測試，並讓該演算法在不同結構的網路中做適當變形。
3. 測試或研發指數衰減形式以外的自適應損失函數機制，嘗試以 cross entropy 檢驗之。

五、 參考文獻

[1] 王奕鈞（2006）。類神經網路應用於地籍坐標轉換之研究。國立政治大學地政學系碩士論文。取自於 https://nccur.lib.nccu.edu.tw/handle/140.119/35873

[2] 許家偉（2012）。全面啟動你的夢境。科學月刊第四十三卷第三期（pp. 202-235）。

[3] Alimoglu, F. (1996). “Combining Multiple Classifiers for Pen-Based Handwritten Digit Recognition.” MSc Thesis, Institute of Graduate Studies in Science and Engineering, Bogazici University. Dataset retrieved from https://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits.

[4] Fisher, R. A. (1936). "The use of multiple measurements in taxonomic problems" Annual Eugenics, 7, Part II, 179-188. Dataset retrieved from https://archive.ics.uci.edu/ml/datasets/iris.

[5] Forina, M., Leardi, R., Armanino, C., & Lanteri,S. (1988). PARVUS - An Extendable Package for Data Exploration, Classification and Correlation. J. Chemometrics, 4, 191 – 193. Dataset retrieved from https://archive.ics.uci.edu/ml/datasets/wine.

[6] Hewitson, B., & Crane, R. G.. Neural Nets: Applications in Geography. Springer Science & Business Media, 1994.

[7] Evett, I. W., & Spiehler, E. J. (1987). Rule Induction in Forensic Science. KBS in Government, 107-118. Dataset retrieved from https://archive.ics.uci.edu/ml/datasets/glass+identification.

[8] Brownlee, J. (2016). How to Implement the Backpropagation Algorithm from Scratch in Python. Algorithms from Scratch. Retrieved from https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/

[9] Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., & Ng, A. Y. (2012). Large scale distributed deep networks. Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1, Pages 1223-1231.

[10] Yeh, I. C., Yang, K. J., & Ting, T. M. (2008). Knowledge discovery on RFM model using Bernoulli sequence. Expert Systems with Applications, Volume 36, Issue 3, Part 2, 5866-5871. Dataset retrieved from https://archive.ics.uci.edu/ml/datasets/Blood+Transfusion+Service+Center.

附錄

（一） 程式碼（Adagrad with REM algorithm）

```
from random import seed
from random import randrange
from random import random
from csv import reader
from math import exp
```

```
from math import sqrt

# Load a CSV file
def load_csv(filename):
    dataset = list()
    with open(filename, 'r') as file:
        csv_reader = reader(file)
        for row in csv_reader:
            if not row:
                continue
            dataset.append(row)
    return dataset

# Find the min and max values for each column
def dataset_minmax(dataset):
    minmax = list()
    stats = [[min(column), max(column)] for column in zip(*dataset)]
    return stats

# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset, minmax):
    for row in dataset:
        for i in range(len(row)-1):
            row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])
```

```
# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
    dataset_split = list()
    dataset_copy = list(dataset)
    fold_size = int(len(dataset) / n_folds)
    for i in range(n_folds):
        fold = list()
        while len(fold) < fold_size:
            index = randrange(len(dataset_copy))
            fold.append(dataset_copy.pop(index))
        dataset_split.append(fold)
    return dataset_split

# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
    folds = cross_validation_split(dataset, n_folds)
    scores = list()
```

```
    for fold in folds :
        train_set = list ( folds )
        train_set . remove ( fold )
        train_set = sum ( train_set , [ ] )
        test_set = list ( )
        for row in fold :
            row_copy = list ( row )
            test_set . append ( row_copy )
            row_copy [ - 1 ] = None
        predicted = algorithm ( train_set , test_set , * args )
        actual = [ row [ - 1 ] for row in fold ]
        accuracy = accuracy_metric ( actual , predicted )
        scores . append ( accuracy )
    return scores

# Calculate neuron activation for an input
def activate ( weights , inputs ) :
    activation = weights [ - 1 ]
    for i in range ( len ( weights ) - 1 ) :
        activation += weights [ i ] * inputs [ i ]
    return activation

# Transfer neuron activation
def transfer ( activation ) :
    return 1.0 / ( 1.0 + exp ( - activation ) )
```

```
# Forward propagate input to a network output
def forward_propagate(network, row):
    inputs = row
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron['weights'], inputs)
            neuron['output'] = transfer(activation)
            new_inputs.append(neuron['output'])
        inputs = new_inputs
    return inputs

# Calculate the derivative of an neuron output
def transfer_derivative(output):
    return output * (1.0 - output)

# Backpropagate error and store in neurons
def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
        if i != len(network)-1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
```

```
                    error   +=   ( neuron [ 'weights' ] [ j ]   *   neuron
[ 'delta' ] )
                    errors . append ( error )
        else :
            for   j   in   range ( len ( layer ) ) :
                neuron   =   layer [ j ]
                errors . append ( expected [ j ]   -   neuron [ 'output' ] )
        for   j   in   range ( len ( layer ) ) :
            neuron   =   layer [ j ]
            neuron [ 'delta' ]   =   errors [ j ]   *   transfer_derivative
( neuron [ 'output' ] )

# Update network weights with error
def  update_weights ( network ,   row ,   l_rate ) :
    global idf
    idf = 1
    for   i   in   range ( len ( network ) ) :
        inputs   =   row [ : - 1 ]
        if   i   !=   0 :
            inputs   =   [ neuron [ 'output' ]   for   neuron   in   network [ i
-   1 ] ]
        for   neuron   in   network [ i ] :
            if idf == 1 :
                if ( exp ( k   -   lamb   *   current_epoch ) ) < 0.0001:
                    idf = 0
            if idf == 1 :
```

```
                for j in range(len(inputs)):
                    neuron['sum_of_pregradient_squares'][j] +=
(neuron['delta'] * inputs[j]) ** 2
                    neuron['weights'][j] += (l_rate / sqrt
(neuron['sum_of_pregradient_squares'][j] + epsilon)) * neuron
['delta'] * inputs[j] * (exp(k - lamb * current_epoch)
+ 1)
                neuron['sum_of_pregradient_squares'][-1] += (neuron
['delta']) ** 2
                neuron['weights'][-1] += (l_rate / sqrt(neuron
['sum_of_pregradient_squares'][-1] + epsilon)) * neuron
['delta'] * (exp(k - lamb * current_epoch) + 1)
            else:
                for j in range(len(inputs)):
                    neuron['sum_of_pregradient_squares'][j] +=
(neuron['delta'] * inputs[j]) ** 2
                    neuron['weights'][j] += (l_rate / sqrt
(neuron['sum_of_pregradient_squares'][j] + epsilon)) * neuron
['delta'] * inputs[j]
                neuron['sum_of_pregradient_squares'][-1] += (neuron
['delta']) ** 2
                neuron['weights'][-1] += (l_rate / sqrt(neuron
['sum_of_pregradient_squares'][-1] + epsilon)) * neuron
['delta']

# Train a network for a fixed number of epochs
```

```
def train_network(network, train, l_rate, n_epoch, n_outputs):
    for epoch in range(n_epoch):
        global current_epoch
        current_epoch = epoch
        for row in train:
            outputs = forward_propagate(network, row)
            expected = [0 for i in range(n_outputs)]
            expected[row[-1]] = 1
            backward_propagate_error(network, expected)
            update_weights(network, row, l_rate)

# Initialize a network
def initialize_network(n_inputs, n_hidden, n_outputs):
    network = list()
    hidden_layer = [{'weights':[random() for i in range(n_inputs + 1)], 'sum_of_pregradient_squares':[0.0 for i in range(n_inputs + 1)]} for i in range(n_hidden)]
    network.append(hidden_layer)
    output_layer = [{'weights':[random() for i in range(n_hidden + 1)], 'sum_of_pregradient_squares':[0.0 for i in range(n_hidden + 1)]} for i in range(n_outputs)]
    network.append(output_layer)
    return network

# Make a prediction with a network
def predict(network, row):
```

```
    outputs = forward_propagate(network, row)
    return outputs.index(max(outputs))

# Backpropagation Algorithm With Stochastic Gradient Descent
def back_propagation(train, test, l_rate, n_epoch, n_hidden):
    n_inputs = len(train[0]) - 1
    n_outputs = len(set([row[-1] for row in train]))
    network = initialize_network(n_inputs, n_hidden, n_outputs)
    train_network(network, train, l_rate, n_epoch, n_outputs)
    predictions = list()
    for row in test:
        prediction = predict(network, row)
        predictions.append(prediction)
    return(predictions)


# Main function
seed(1)
global k, lamb
k = 2.3
lamb = 0.6

filename = 'iris.csv'
dataset = load_csv(filename)

for c in range(len(dataset)):
```

```
    dataset [ c ]  =  [ float ( element )  for  element  in  dataset
[ c ] ]
    dataset [ c ] [ -1 ]  =  int ( dataset [ c ] [ -1 ] )

minmax  =  dataset_minmax ( dataset )
normalize_dataset ( dataset , minmax )

n_folds = 5
l_rate = 0.1
n_hidden = 10
epsilon = 0.00000001
n_epoch = 10
mse = []
epoch_count = []

scores  =  evaluate_algorithm ( dataset ,  back_propagation ,  n_folds ,
l_rate ,  n_epoch ,  n_hidden )

print ( 'cross-validation : %s'  %  scores )
print ( 'Mean Accuracy:%.3f%%' % ( ( sum ( scores ) / float ( len
( scores ) ) ) ) )
```

【評語】190003

此作品是基於人腦於睡眠時，腦波較高的觀察，在類神經網絡訓練上，引入類似的觀念，將把 loss function 加入一放大係數，藉此達到快速收斂，縮短訓練時間。比作品創意很好，甚有發展潛力。建議參賽者，從觀察到 Loss function 的推導過程，能有更有組織的表達。在實驗部分，可嘗試更的 Data Set.