選擇權動態避險：使用深度學習
Dynamic Hedging of Options by Deep Learning

摘要

本文探討選擇權的動態避險，以深度學習方法導出給定 Black-Scholes 模型下標準歐式買權的動態 Delta 避險策略：使用滿足此機率模型的模擬路徑爲訓練資料，透過最小化給定風險準則進行深度學習訓練與計算。比較計算所得與理論公式解結果，除在初始與期末或深度價內價外時較有差距外，兩者結果非常接近。

We derive the dynamic hedging rules of the standard European option governed by the Black-Scholes model via deep learning techniques. Using simulated paths as train data and minimizing with the least square risk criterion, the computed results and the theoretical ones are in good agreement except when the underlying option is deep in or out of the money.

關鍵字: Delta 避險, Black-Scholes 模型, 選擇權, 類神經網路, 深度學習

Keywords Delta hedging, Black-Scholes model, options, artificial neural networks, deep learning

1. 研究動機與緒論

避險一詞泛指爲在期間內消除影響資產價值之標的計量變動風險而進行的投資策略，由相關的金融工具反向部位組合而成。由於避險操作需要成本，風險規避往往伴隨獲利降低，如何在避險與收益間取得平衡一直都是商業經營核心研究項目；與保險業息息相關的，則是匯率變動以及投資型商品的避險議題。

目前保險業的海外投資比例已達總資產的三分之二以上，在 IFRS 會計準則下保險公司負債必須以公允價值評量，利率與匯率的變動對於保險公司的資本運用管理影響深遠。據報導，台灣壽險業 2019 年購買避險契約成本與匯損爲新台幣 2912 億，已爲歷年新高；今 (2020) 年又值全球疫情與金融市場起伏，台幣對美元匯率至 2020 年十月間升至九年來新高，壽險業匯率變動避險成本遽增，對其生存與發展帶來高度挑戰。

投資型商品如變額年金 (Variable Annuity，VA) 等往往可視作選擇權的組合 (參見 Hardy (2003))，於是能藉由財務工程理論工具，配合設定標的且滿足給付規則之機率模型進行評價；此等商品的避險方式亦等價於選擇權的避險方式。選擇權的標準動態避險策略之一是 Delta 避險：計算選擇權價格對於其標的價格之變化率 Delta 得出避險期間各時點持有之標的部位比例，使得此包含選擇權與標的之投資組合總價值保持恆定，不隨標的價格而有所改變。傳統上要計算 Delta 首先必須設定機率模型，得出選擇權價格公式之後對變數標的取導數。Delta 避險必須依照標的價格變化隨時調整持有之標的部位比例，且不考慮標的交易成本以及流動性等種種限制，實際施行時必定造成誤差而降低避險效果。

近年來機器學習領域的蓬勃發展，開啓了金融領域現代化與自動化的新面向 (de Prado (2018, 2020); Hull (2020); Dixon, Halperin and Bilokon (2020)) 。機器學習結合大數據在程式交易與核保授信方面已有廣泛的應用，而在深度學習 (Deep Learning，DL；Goodfellow, Bengio and Courville (2016); Aggarwal (2018); Nielsen (2015)) 與深度加強學習 (Deep Reinforcement Learning，DRL；Sutton and Barto (2018); Szepesvári (2010)) 理論分別在圖像識別 (Ciresan et al. (2012)) 及電腦圍棋 (Silver et al. (2016)) 取得劃時代的成績後，應用這些原理在市場預測以及商品評價已有相當的成果。深度學習

即爲深層的類神經網路（Artificial Neural Networks，ANN），一種能從已知輸入/輸出訓練資料集合中擷取其函數關係並推廣的運算機制。Hutchinson, Lo and Poggio (1994); Buehler, Gonon, Teichmann and Wood (2019) 的文章充分證明了深度學習在建立選擇權動態避險策略上的可行性。由於深度學習機制需要的只是訓練資料，不需要事先假設其標的機率模型，且加入眞實市場中必定存在的交易成本與流動性限制並不困難，在適當的訓練之下深度學習可得出更有彈性、更符合實際需要的動態避險策略。

本文主要探討投資型商品—亦即選擇權—的避險策略。我們使用深度學習導出給定 Black-Scholes 模型下標準歐式買權的動態 Delta 避險策略 — 以滿足此機率模型的模擬路徑爲訓練資料，透過最小化給定風險準則進行深度學習訓練 — 並與理論解結果相比較。本文其餘章節安排如下：以範例闡釋不同選擇權避險策略之影響與簡述深度學習原理及架構後，我們討論使用深度學習計算動態 Delta 避險策略的方法並進行數值實驗比較所得結果與理論解，最後爲結論與展望。

2. 選擇權避險策略比較：範例

假設一金融機構透過場外交易發行歐式買權規格如下：到期日爲 20 星期 — $T = 20/52 = 0.3846$ (年)，履約價 $K = \$50$。假設標的風險資產滿足 Black-Scholes 模型：無風險年利率 $r = 0.05$，標的風險資產平均年收益率 $\mu = 0.13$，變異數 $\sigma = 0.2$，標的起始價格 $s_0 = \$49$，且買權定價爲 \$3。針對風險資產價格變動曝險量之避險策略，Hull (2015, Chapter 19) 提供了四種想法，分別爲不避險（naked）、掩護（covered）、停損（stop-loss）與 delta-bs。掩護指的是期初便買入標的風險資產，如此一來可在期末買權執行時直接移轉期初買入之標的風險資產。停損是掩護的修正：在避險期間內若標的風險資產價格大於履約價時買入標的風險資產，低於履約價時賣出。delta-bs 策略是在各避險期間時點計算買權價格對標的風險資產價格的導數 Δ，買入的標的風險資產量爲 $\Delta \times$（前後時點標的風險資產價差）。除了 delta-bs 之外，其他三種避險策略都是直覺可想到的。

針對不同的避險操作頻率，我們使用 Monte-Carlo 法模擬 10^6 道標的風險資產價格變動之路徑，針對每道路徑實行以上四種避險策略，觀察其避險損益並進行比較。爲簡單起見，避險損益不考慮折現與標的風險資產價值增長；避險策略的優劣將以避險效率比值決定，其定義爲：（避險效率比值）≡（避險損益標準差）/（期初買權價格）— 避險效率比值越低，避險策略越佳— 由 Black-Scholes 公式可得期初買權價格爲 2.4005，故避險效率比值與避險損益標準差成正比。使用 Python 程式語言撰寫之程式碼可參見附錄 A.1，$\Delta t = 0.25$ 各避險策略損益分佈圖如圖 1，計算結果如表 1。

Δt	delta-bs	mean	stop-loss	mean	covered	mean	naked	mean
5	0.4215	0.4505	1.0489	0.8989	1.4095	1.5034	1.6136	0.5542
4	0.3821	0.4478	1.0109	0.9212	1.4105	1.5015	1.6188	0.5454
2	0.2832	0.4500	0.9188	0.9546	1.4104	1.5068	1.6148	0.5479
1	0.2139	0.4492	0.8772	0.9720	1.4077	1.5053	1.6140	0.5533
0.5	0.1666	0.4494	0.8630	0.9785	1.4107	1.5006	1.6125	0.5558
0.25	0.1360	0.4501	0.8618	0.9857	1.4101	1.5009	1.6157	0.5529

表 1: 不同避險操作頻率下，各避險策略之避險效率比值與平均避險損益對照表。

首先留意買權的定價爲 \$3，比 Black-Scholes 公式價格 \$2.4005 多出 \$0.6 左右。表 1 的結果顯示在 delta-bs 避險策略情況下總損益的變化最少、最穩定，且避險操作越頻繁穩定性越高。圖 1 顯示，除了 delta-bs 避險策略外，其餘三種策略可能的損失極大，尤其在不避險的狀況下 — 但可能的獲利都高於 delta-bs 避險策略。以上的範例與數值結果指出了有效避險策略的複雜性，同時也演示了避險與獲利間取捨的微妙關係。

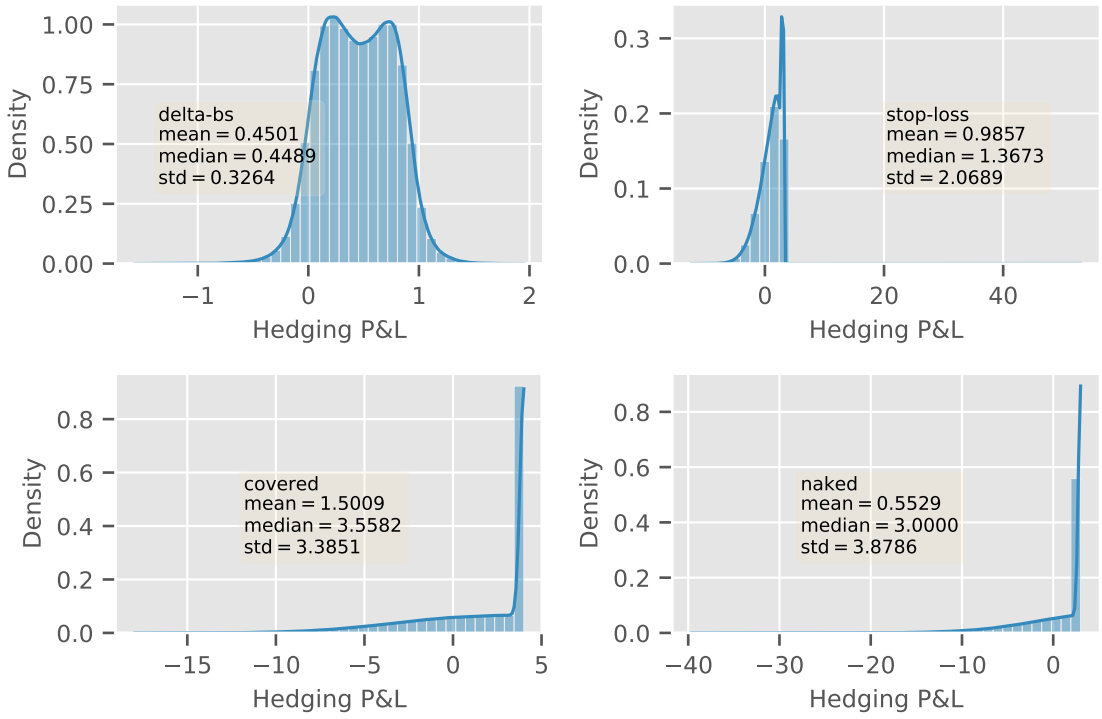

圖 1: 避險操作頻率 $\Delta t = 0.25$ 時，各避險策略損益分佈。

3. 深度學習簡介

深度學習是一種強化的類神經網路機制，廣泛用於機器學習中的分類與預測問題。類神經網路是一種模仿生物神經系統結構與功能的數學模型，由大量的人工神經元連結組成，憑藉輸入/輸出已知資料最佳化聯繫各人工神經元的突觸（synapse）權重後，此一模型便可用來估計未知資料的函數關係。類神經網路機制中最常見的前饋網路（feed-forward networks）指的是多層的人工神經元叢集，每一層由若干彼此不互相連結的人工神經元構成，且以層爲單位，單向輸入/輸出。圖 2 的前饋網路由具備三個輸入神經元的輸入層、各有四、四、三個神經元的三隱藏層與兩個輸出神經元的輸出層組成。假設前饋網路有 L 層，

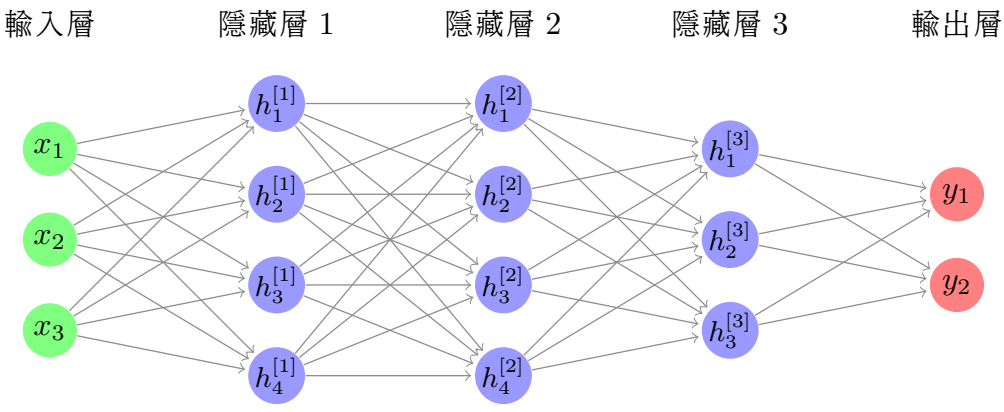

圖 2: 前饋網路示意圖。

每層分別有 $(n_1, n_2, \ldots, n_L)$ 個神經元，第 1 層爲輸入層，第 L 層爲輸出層，其餘爲隱藏層。對於每個 $1 \leqslant l \leqslant L-1$，定義活化函數（activation function）$\varphi^l : \mathbb{R}^{n_{l+1}} \mapsto \mathbb{R}^{n_{l+1}}$ 與線性函數 $w^l : \mathbb{R}^{n_l} \mapsto \mathbb{R}^{n_{l+1}}$

$$w^l(x) = a^l x + b^l$$

其中矩陣 $a^l \in \mathbb{R}^{n_l} \times \mathbb{R}^{n_{l+1}}$ 的 (i,j) 元素 a^l_{ij} 爲連結第 l 層的第 i 個神經元至第 $l+1$ 層的第 j 個神經元之權重，$b^l \in \mathbb{R}^{n_{l+1}}$，同時定義 $f^l \equiv \varphi^l \circ w^l$，我們稱函數 $F : \mathbb{R}^{n_1} \mapsto \mathbb{R}^{n_L}$

$$F(x) = w^L \circ f^{L-1} \circ f^{L-2} \circ \cdots \circ f^1(x) \tag{1}$$

爲此前饋網路之數學表示模型，其權重集爲 $w \equiv \{a^l, b^l \,|\, 1 \leqslant l \leqslant L-1\}$；爲強調其權重集 w，也可記作 $F(w,x)$。常用的活化函數爲非線性，且各分量之函數形式皆相同。由 Hornik (1991, Theorems 1, 2) 的結果，若活化函數爲連續，前饋網路的數學表示模型便能任意的逼近給定的連續函數。

前饋網路數學表示模型 $F(w,x)$ 的最佳權重集 $\widehat{w}$ 由已知輸入/輸出訓練資料集 $\{X,Y\}$，透過最佳化給定損失函數（loss function）$\mathcal{L} : \mathbb{R}^{n_L} \times \mathbb{R}^{n_L} \mapsto \mathbb{R}$ 與網路模型形成的目標函數得出:

$$\widehat{w} = \underset{w}{\operatorname{argmin}}\, \mathcal{L}(F(w,X),Y)$$

由於前饋網路的權重集元素數量極爲龐大，而目標函數往往非常複雜且不具全域凹性，此問題有效求解的方法一直是研究焦點。改進最佳化前饋網路目標函數最常用的隨機梯度下降法（stochastic gradient descent，SGD），有效提高複雜前饋網路的訓練效率與分類 / 預測能力的一系列方法通稱爲深度學習；我們將使用深度學習增強的前饋網路稱作深度前饋網路。

4. 深度學習動態避險策略

考慮機率空間 $(\Omega, \mathcal{F}_{\{0\leqslant t\leqslant T\}}, \mathbb{P})$ 及 Black-Scholes 模型: 不失普遍性，令無風險利率 $r=0$，而在物理測度 $\mathbb{P}$ 下標的風險資產價格過程 $S_t, 0 \leqslant t \leqslant T$ 滿足

$$\mathrm{d}S_t = S_t\,\mu\,\mathrm{d}t + S_t\,\sigma\,\mathrm{d}W_t^{\mathbb{P}}, \quad S_0 = s_0$$

其中報酬率 $\mu \geqslant 0$，變異數 $\sigma > 0$，標的風險資產價格初始值 $s_0 > 0$ 且 $W_t^{\mathbb{P}}$ 爲物理測度 $\mathbb{P}$ 下的標準 Wiener 過程。此完備模型存在唯一風險中立機率測度 $\mathbb{Q}$，標的風險資產價格過程 S_t 在此風險中立機率測度 $\mathbb{Q}$ 下可寫作

$$\mathrm{d}S_t = S_t\,r\,\mathrm{d}t + S_t\,\sigma\,\mathrm{d}W_t, \quad S_0 = s_0$$

其中 W_t 爲測度 $\mathbb{Q}$ 下的標準 Wiener 過程。

我們研究在此 Black-Scholes 模型下歐式買權的避險策略問題。令歐式買權之期限爲 T，到期時給付爲 $f(S_T)$，其中 f 爲給定的給付函數。我們知道在時點 t 且標的風險資產 $S_t = s$ 時標準歐式選擇權價格 $p(t,s) = \mathsf{E}_{\mathbb{Q}}\left\{f(S_T) \,|\, S_t = s\right\}$，其中 $\mathsf{E}_{\mathbb{Q}}\{\cdot\}$ 爲風險中立測度 $\mathbb{Q}$ 下的條件期望值。由 Bouchard and Chassagneux (2016, Theorem 4.3)，

$$p(t,s_0) + \int_t^T \partial_s p(\tau, S_\tau)\,\mathrm{d}S_\tau = f(S_T)$$

在此完備 Markov 模型下的動態避險策略爲

$$\Delta(t,s) = \partial_s p(t,s)$$

在此考慮二次化避險原則，亦即下列最小化問題

$$\inf_H \mathsf{E}_{\mathbb{Q}}\left\{\left(f(S_T) - p(0,S_0) - \int_0^T H_t\,\mathrm{d}S_t\right)^2\right\}$$

其中 H 遍歷所有允許的隨機過程。我們利用深度前饋網路來近似 H：對每個時點 t 建立一個深度前饋網路 $g_t(\cdot)$，且 H_t 滿足

$$H_t = g_t(S_t)$$

監督學習問題可敍述爲：輸入爲一組 S_t，$0 \leqslant t \leqslant T$，輸出值 0，損失函數 $\mathcal{L}$ 爲

$$\mathcal{L} = \left(f(S_T) - p(0, s_0) - \int_0^T g_t(S_t)\,\mathrm{d}S_t \right)^2$$

上式中的隨機積分

$$\int_0^T g_t(S_t)\,\mathrm{d}S_t$$

可以 N 個深度前饋網路 $g_0(\cdot), g_1(\cdot), \ldots, g_{N-1}(\cdot)$，透過

$$\sum_{i=0}^{N-1} g_i(S_i)(S_{i+1} - S_i)$$

離散化。我們以模擬方式生成此深度前饋網路的訓練與測試資料集。考慮標的風險資產價值過程 S_t 的對數值在風險中立測度 $\mathbb{Q}$ 下的連續時間變動過程

$$\mathrm{d}\log S_t = \left(r - \frac{\sigma^2}{2} \right) \mathrm{d}t + \sigma\,\mathrm{d}W_t \tag{2}$$

此過程的離散化方式如下：將標準歐式買權有效期間 $[0, T]$ 分成 N 個長度爲 $\frac{T}{N}$ 的時間區間，(2) 可寫作

$$\log S_i = \log S_{i-1} + \left(r - \frac{\sigma^2}{2} \right) \frac{T}{N} + \sigma \sqrt{\frac{T}{N}}\, Z_i, \quad i = 1, \ldots, N$$

其中 Z_i 爲標準常態分佈隨機變數值，而 S_t 在分割的時間區間上的值 $(S_0, S_1, \ldots, S_N)$ 可藉由指數運算得到，顯然的等價於 $(S_0, Z_1, \ldots, Z_N)$，又等價於 $(S_0, X_1, \ldots, X_N)$ — X_i 是符合平均值 $\left(r - \frac{\sigma^2}{2}\right) \frac{T}{N}$，變異數爲 $\sigma^2 \frac{T}{N}$ 的常態分佈的獨立隨機值。這樣的一組路徑 $(S_0, S_1, \ldots, S_N)$ 即爲輸入，而輸出爲 0；訓練與測試資料集便從分別反覆此過程 n_{train}，n_{test} 次得出。

5. 數值實驗結果

我們使用 `Python` 程式語言與開源深度學習函數庫 `TensorFlow`（Abadi et al. (2015)）與 `Keras`（Chollet et al. (2015)）進行數值實驗。本實驗使用的參數如表 2 所示。

經過程式（參見附錄 A.2）建構深度前饋網路後，我們以前節所提方式模擬出訓練與測試資料集並進行訓練與測試，結果如圖 3 所示。在標的風險資產起始值爲 100 的情形下，訓練與測試集的平均損益在 0.0013，損益中位數落在 0.02 左右，而標準差接近 1.33。圖 4 顯示在四個不同時點時理論與深度學習 Delta 值的比較：在初始與接近期末時兩者差距較大，深度價內價外亦然，其餘情形下兩者非常接近；透過圖 5 可更能看出時點與價內價外關係對 Delta 值的影響。

6. 結論與展望

本文使用深度學習方法計算 Black-Scholes 模型中標準歐式買權的動態避險 Delta 策略，並與理論值相比較。除在期初與期末時點與深度價內價外情形時，深度學習計算值與理論值間誤差甚小。考慮理論公式的複雜度與深度學習以訓練資料而非模型機率假設爲核心的計算方式，這樣的結果是非常正向、具發展性的。除了試驗使用不同的深度前饋網路架構對計算結果的影響之外，多避險資產與實際避險交易成本機制的引入、各種風險判斷準則與損失函數的組合、以及深度加強學習方法的配合等等都是可能的研究方向。

參數	數值	定義
T	$\frac{20}{52}$	選擇權持有期間
r	0.05	無風險利率
σ	0.2	標的變異數
S_0	100	標的起始價格
K	100	履約價
N	30	持有期間分割數
m	1	避險資產種類數
d	2	每分割之分層數
n	8	每分層之節點數
n_{train}	5×10^5	訓練情境數
n_{test}	10^5	測試情境數

表 2: 本文使用計算參數。

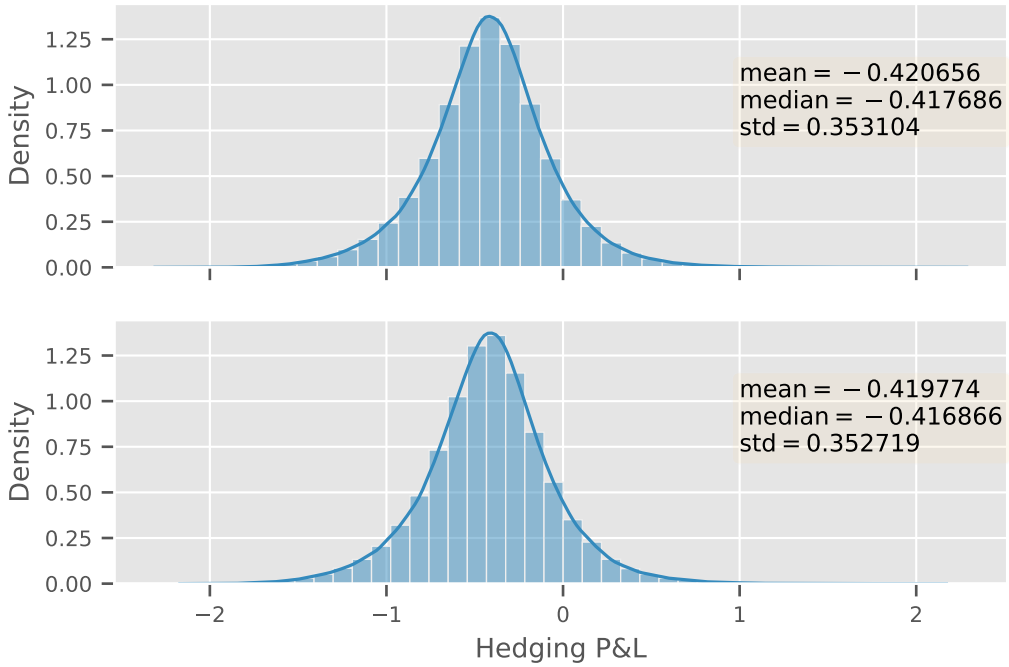

圖 3: 深度學習模型之避險損益機率分佈。上圖: 使用 $n_{\text{train}} = 5 \times 10^5$ 訓練情境。下圖: 使用 $n_{\text{test}} = 10^5$ 測試情境。

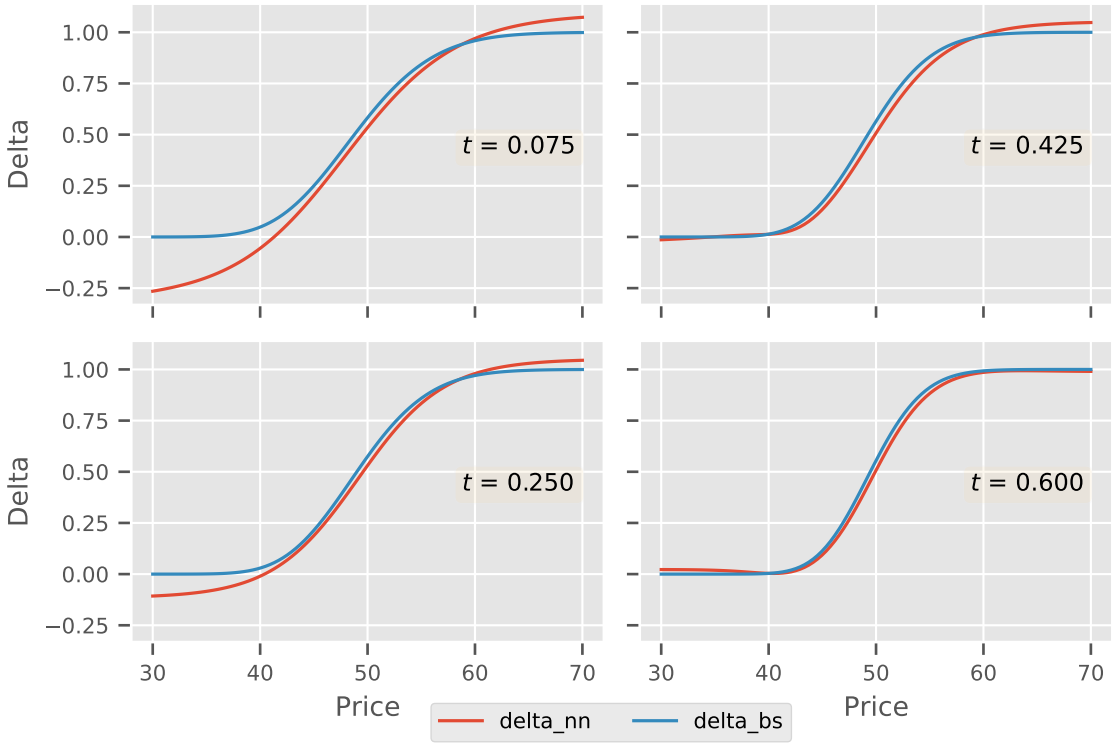

圖 4: 給定 Black-Scholes 模型下，不同時點 t 理論 Delta 值與深度學習 Delta 值之比較圖。

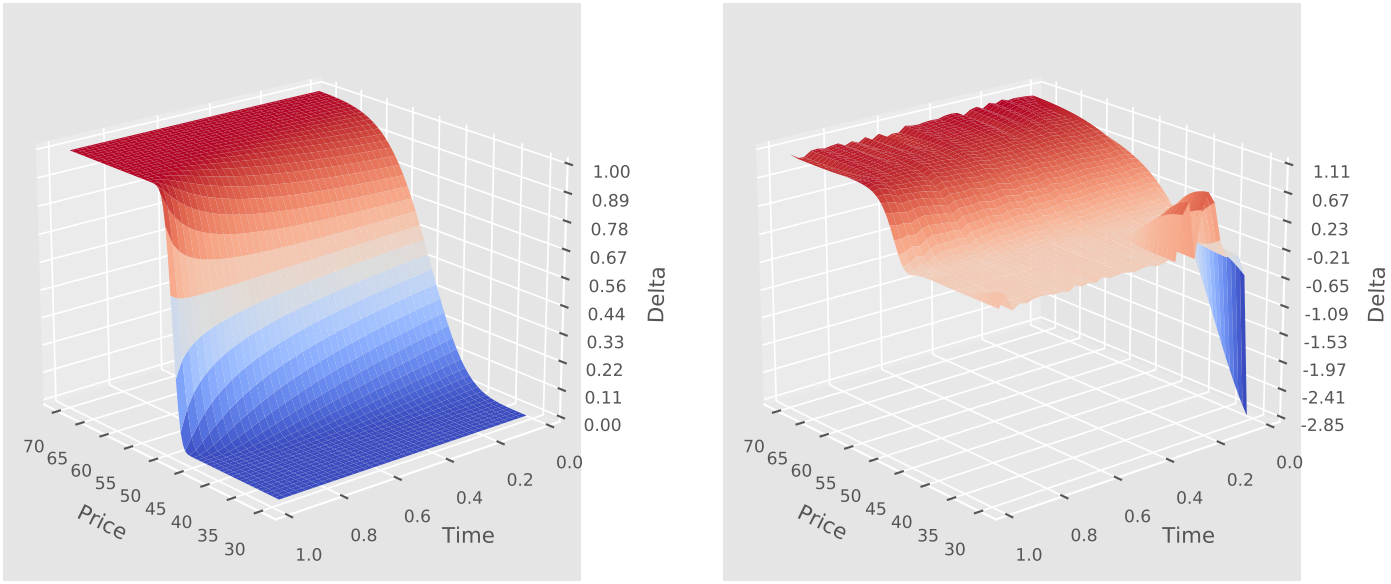

圖 5: 給定 Black-Scholes 模型下，Delta 值與時間間隔 / 標的價格之關係。左圖爲理論公式解，右圖爲深度學習解。

参考文献

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. URL: `https://www.tensorflow.org/`. software available from tensorflow.org.

Aggarwal, C.C., 2018. Neural Networks and Deep Learning: A Textbook. Springer International, Switzerland.

Bouchard, B., Chassagneux, J.F., 2016. Fundamentals and Advanced Techniques in Derivatives Hedging. Springer International, Switzerland.

Buehler, H., Gonon, L., Teichmann, J., Wood, B., 2019. Deep hedging. Quantitative Finance 19, 1271–1291. doi:`10.1080/14697688.2019.1571683`.

Chollet, F., et al., 2015. Keras. `https://keras.io`.

Ciresan, D.C., Meier, U., Masci, J., Schmidhuber, J., 2012. Multi-column deep neural network for traffic sign classification. Neural Networks 32, 333–338. URL: `http://dblp.uni-trier.de/db/journals/nn/nn32.html#CiresanMMS12`.

Dixon, M.F., Halperin, I., Bilokon, P., 2020. Machine Learning in Finance: From Theory to Finance. Springer International, Switzerland.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning: Adaptive Computation and Machine Learning. The MIT Press, Cambridge, Massachusetts.

Hardy, M., 2003. Investment Guarantees: Modeling and Risk Management for Equity-Linked Life Insurance. John Wiley & Sons, New York.

Hornik, K., 1991. Approximation capabilities of multilayer feedforward networks. Neural Networks 4, 251–257.

Hull, J.C., 2015. Options, Futures, and Other Derivatives. Tenth ed., Pearson, Boston.

Hull, J.C., 2020. Machine Learning in Business: An Introduction to the World of Data Science. Second ed., Independently Published.

Hutchinson, J.M., Lo, W., Poggio, T., 1994. A nonparametric approach to pricing and hedging derivatives securities via learning networks. Journal of Finance 49, 851–889.

Nielsen, M.A., 2015. Neural Networks and Deep Learning. Determination Press. URL: `http://neuralnetworksanddeeplearning.com/index.html`.

de Prado, M.L., 2018. Advances in Financial Machine Learning. John Wiley & Sons, Hoboken, N.J.

de Prado, M.L., 2020. Machine Learning for Asset Managers. Cambridge University Press, Cambridge.

Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D., 2016. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489. doi:10.1038/nature16961.

Sutton, R.S., Barto, A.G., 2018. Reinforcement Learning: An Introduction. Second ed., The MIT Press, Cambridge, Massachusetts.

Szepesvári, C., 2010. Algorithms for Reinforcement Learning. Morgan & Claypool, San Rafael, C.A.

附錄 A. Python 程式碼

附錄 A.1. 避險策略比較

```python
import numpy as np
from scipy.stats import norm
from numpy import log, sqrt, exp, zeros, ones, std, mean, maximum, cumsum, cumprod, reshape, transpose

np.random.seed(1000)
phi = norm(loc=0, scale=1).cdf

def bs_d1(S, dt, r, sigma, K):
    return (log(S / K) + (r + sigma**2 / 2) * dt) / (sigma * sqrt(dt))

def bs_price(S, T, r, sigma, K, t):
    dt = T - t
    d1 = bs_d1(S, dt, r, sigma, K)
    d2 = d1 - sigma * sqrt(dt)
    return S * phi(d1) - K * exp(-r * dt) * phi(d2)

def bs_delta(S, T, r, sigma, K, t):
    dt = T - t
    d1 = bs_d1(S, dt, r, sigma, K)
    return phi(d1)

def mc_paths(s0, T, sigma, r, n_sims, n_steps):
    rv = np.random.randn(n_sims, n_steps)
    dt = T / n_steps
    sT = s0 * cumprod(exp((r - sigma**2 / 2) * dt + sigma * sqrt(dt) * rv), axis=1)
    return reshape(transpose(np.c_[ones(n_sims) * s0, sT]), (n_steps + 1, n_sims))

N = 5         # time disrectization
```

```
s0 = 49.       # initial value of the asset
K = 50.        # strike for the call option
T = 20. / 52   # maturity
sigma = 0.2    # volatility
premium = 3    # option premium
r = 0.05
n_sim = 10 ** 6

def pnl_delta(deltas, paths, K, price, alpha):
    ds = paths[1:, :] - paths[:-1, :]
    hedge = np.sum(deltas * ds, axis=0)
    payoff = maximum(paths[-1, :] - K, 0)
    pnls = -payoff + hedge + price
    return pnls

def pnl_delta_bs(s0, K, r, sigma, T, paths, alpha):
    times = zeros(paths.shape[0])
    times[1:] = T / (paths.shape[0] - 1)
    times = cumsum(times)
    bs_deltas = zeros((paths.shape[0] - 1, paths.shape[1]))
    for i in range(paths.shape[0] - 1):
        t = times[i]
        bs_deltas[i, :] = bs_delta(paths[i, :], T, r, sigma, K, t)
    return pnl_delta(bs_deltas, paths, K, bs_price(s0, T, r, sigma, K, 0),
    ↪  alpha)

table = []
for N in [4, 5, 10, 20, 40, 80]:
    bs = bs_price(s0, T, r, sigma, K, 0)
    paths = mc_paths(s0, T, sigma, r, n_sim, N)
    pnl_stop_loss = zeros((n_sim, 1))
    pnl_covered = zeros((n_sim, 1))
    pnl_naked = zeros((n_sim, 1))
    epsilon = 1e-2
    for i in range(n_sim):
        position, cost = 0, 0
        for price in paths[:, i]:
            if price >= K + epsilon:
                if position == 0:
                    position = 1
                    cost += price
            elif price <= K - epsilon:
                if position == 1:
                    position = 0
                    cost -= price
        final = paths[-1, i]
```

```
        pnl_stop_loss[i] = premium - cost + (K if final >= K else (final if
        ↪  position else 0))
        pnl_covered[i] = premium - s0 + (K if final >= K else final)
        pnl_naked[i] = premium - maximum(final - K, 0)

    pnl_bs = pnl_delta_bs(s0, K, r, sigma, T, paths, 0.5)
    table.append((20. / N, std(pnl_bs) / bs, mean(pnl_bs), std(pnl_stop_loss) /
    ↪  bs, mean(pnl_stop_loss), std(pnl_covered) / bs, mean(pnl_covered),
    ↪  std(pnl_naked) / bs, mean(pnl_naked)))
```

附錄 A.2. 深度學習與理論

```
from keras.models import Model
from keras.layers import Input, Dense, Subtract, Multiply, Lambda, Add
from keras.engine.topology import Layer
from keras import initializers as init
import keras.backend as K
import numpy as np
from numpy import log, exp, sqrt, ones, zeros, tanh, matmul, squeeze
from scipy.stats import norm

def bs(s0, strike, T, sigma):
    return s0 * norm.cdf((log(s0/strike) + 0.5 * T * sigma**2) / (sqrt(T) *
    ↪  sigma)) - strike * norm.cdf((log(s0 / strike) - 0.5 * T * sigma**2) /
    ↪  (sqrt(T) * sigma))

def delta_bs(s, k):
    return norm.cdf((log(s/strike) + 0.5 * (T - k * T / N) * sigma**2) /
    ↪  (sqrt(T - k * T / N) * sigma))

N, s0, strike, T, sigma = 30, 100, 100, 1.0, 0.2

price_bs = bs(s0, strike, T, sigma)

m, d, n = 1, 2, 8
layers = []
for j in range(N):
    for i in range(d):
        if i < d - 1:
            nodes = n
            layer = Dense(nodes, activation='tanh', trainable=True,
            ↪  kernel_initializer=init.RandomNormal(0, 0.1),
            ↪  bias_initializer=init.RandomNormal(0, 0), name=str(i)+str(j))
        else:
```

```
            nodes = m
            layer = Dense(nodes, activation='linear', trainable=True,
             ↪  kernel_initializer=init.RandomNormal(0, 0.1),
             ↪  bias_initializer=init.RandomNormal(0, 0), name=str(i)+str(j))
        layers = layers + [layer]

price = Input(shape=(m,))
hedge = Input(shape=(m,))
inputs = [price] + [hedge]

for j in range(N):
    strategy = price
    for k in range(d):
        strategy = layers[k + (j) * d](strategy)
    incr = Input(shape=(m,))
    logprice = Lambda(lambda x: K.log(x))(price)
    logprice = Add()([logprice, incr])
    pricenew = Lambda(lambda x: K.exp(x))(logprice)
    priceincr = Subtract()([pricenew, price])
    hedgenew = Multiply()([strategy, priceincr])
    hedge = Add()([hedge, hedgenew])
    inputs = inputs + [incr]
    price = pricenew
payoff = Lambda(lambda x: 0.5 * (K.abs(x - strike) + x - strike) -
 ↪  price_bs)(price)
outputs = Subtract()([payoff, hedge])

model_hedge = Model(inputs=inputs, outputs=outputs)
model_hedge.compile(optimizer='adam', loss='mean_squared_error')

def get_x(n):
    return [s0 * ones((n, m))] + [zeros((n, m))] +
     ↪  [np.random.normal(-(sigma)**2 * T / (2*N), sigma * sqrt(T) / sqrt(N),
     ↪  (n, m)) for i in range(N)]

n_train = 5 * 10**5
xtrain = get_x(n_train)
ytrain = zeros((n_train, 1))

n_test = 10**5
xtest = get_x(n_test)
ytest = zeros((n_test, 1))

model_hedge.fit(x=xtrain, y=ytrain, epochs=250, verbose=True, batch_size=100)

weights = model_hedge.get_weights()
```

```
def delta_nn(s, j):
    length = s.shape[0]
    g = zeros(length)
    for p in range(length):
        ghelper = tanh(s[p] * (weights[j * 2 * d]) + weights[j * 2 * d + 1])
        g[p] = np.sum(squeeze(weights[2 * (d - 1) + j * 2 * d]) *
        ↪  squeeze(ghelper))
        g[p] = g[p] + weights[2 * d - 1 + j * 2 * d]
    return g
```