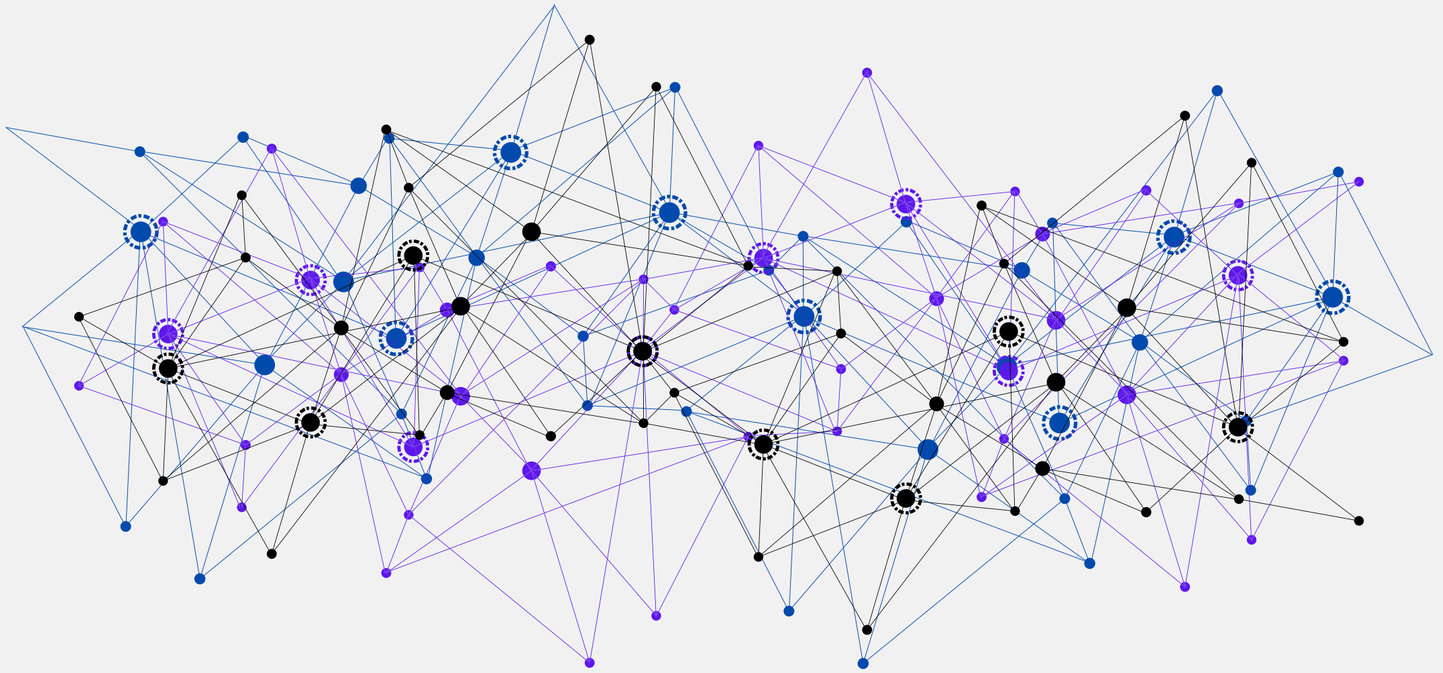
阿米巴神經網絡文檔



$$\mathbf{x}_{k+1} = (\mathcal{W}_k \sigma_k + \beta_k) \odot (W_k \mathbf{x}_k + \mathbf{b}_k)$$

$$\varphi(\mathbf{x}_1) := \mathbf{x}_p$$

作者
李翺汶博士

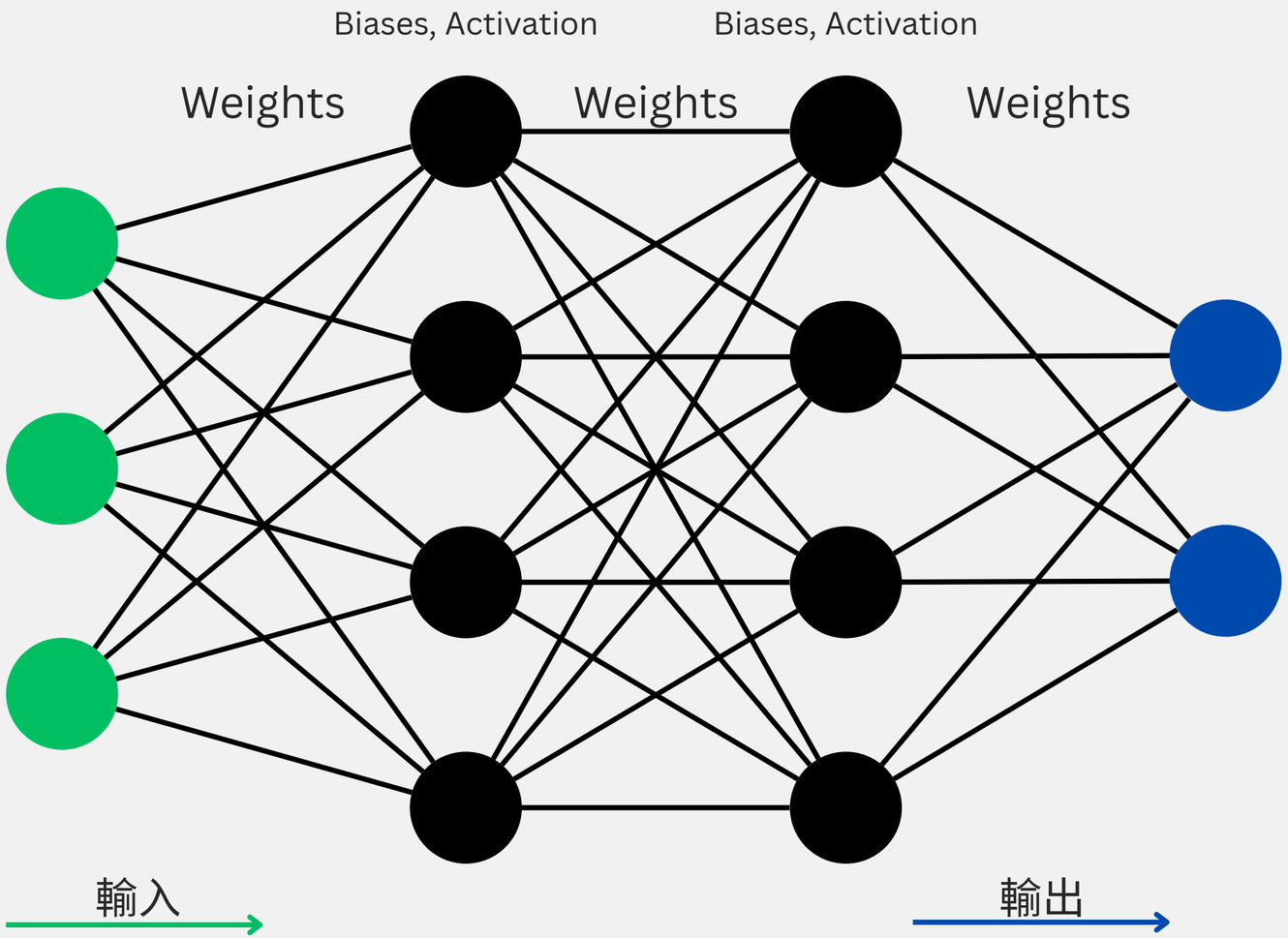
引言

- 我們目標是要研發一個更有效率的**人工神經網絡 (*Artificial Neural Network* (ANN))** 架構
- 主要參考是 ***Self-Expanding Neural Network* (SENN)** [1] 的架構
- 模型名字叫**阿米巴神經網絡 (*Amoeba Neural Network* (ANN))**, 因為它可以自我擴張和變型
- 研發成功將會是一個革命性創新
- 可以令**語言大模型 (LLM)** 更加有效率, 訓練更便宜
- 由於我們還在研究和測試階段, 所以會在短時間內有大量改動
- 要有心理準備

計劃

- 我們先從最簡單的**圖像識別 (OCR) 人工智能 (*Artificial Intelligence* (AI))** 開始
- 原因是 OCR 已經有大量的 AI 訓練數據, 可以為我們定下標準, 作出模型表現上的比較
- 如果初步確定阿米巴神經網絡是比傳統人工神經網絡有效率的話, 我們可以再測試在其他地方的應用
- 例如: 預測附隨機性的事件和金融市場價格等等
- 如研究經費許可, 我們更可以測試於模型較小的已**知識蒸餾 (*Knowledge Distillation*)** [2] 的 LLM 上
- 例如: **ChatGPT**, **DeepSeek**

人工神經網絡架構概念圖



詞彙及定義

- **矢量 (*Vector*)**: 縱排及橫排的單一排數例
- 例子:
  $\begin{pmatrix} 0 & 1 & 2 \end{pmatrix}$ $\begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix}$
- **矩陣 (*Matrix*)**: 縱橫二維的數表
- 例子:
  $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$
- 矢量其實都歸類為 (一維的) 矩陣

激活函數

- **激活函數 (*Activation Function*)** 是負責控制神經元輸出的函數
- 我們需要做測試的目的就是想找出最優化的激活函數
- 代碼為 **activation_function(value, degree)**
- 定義: **activation_function(value, 0)=value**
- 例子: **power(value, degree)=value**degree**

Vandermonde 矩陣

- **Vandermonde 矩陣 (*Vandermonde Matrix*)** 是一個取進一個矢量和一個函數作為輸入的函數
- 輸出為激活函數應用於一個矢量而得出的矩陣
- 虛擬碼:

$$\text{vandermonde\_matrix}(\text{vector}, \text{activation\_function}) = \begin{pmatrix} \text{activation\_function}(\text{vector}[1], 0) & \dots & \\ \dots & & \\ \text{activation\_function}(\text{vector}[\text{m}], 0) & \dots & \text{activation\_function}(\text{vector}[\text{m}], \text{n}) \end{pmatrix}$$

- 例子: **vector_power(vector, max_power)** 函數上是相等於 **vandermonde_matrix(vector, power)**

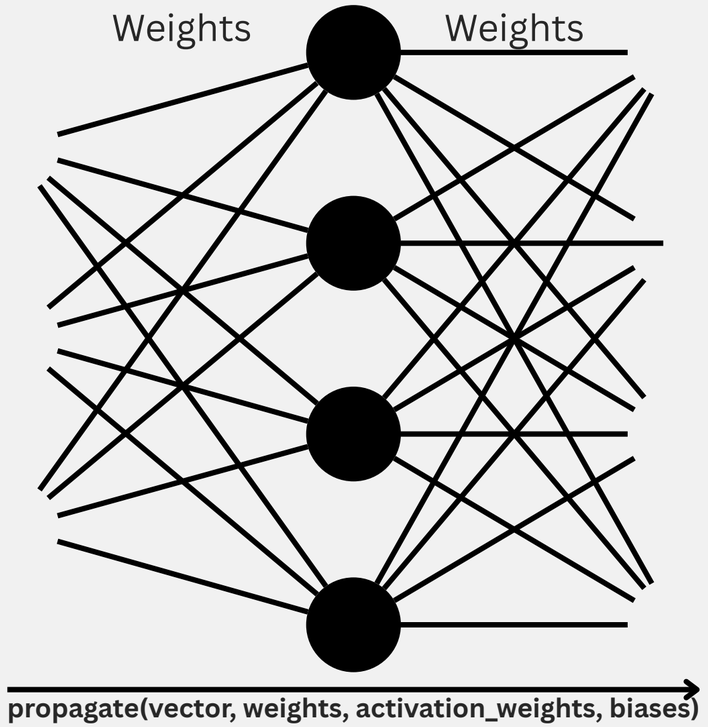
對角乘積

- **對角乘積 (*Diagonal Product*)** 是我們引入和命名的一個函數
- 數學文獻上尚未有此定義
- 我們定義這函數是為了計算 **propagate(vector, weights, activation_weights, biases)**
- 虛擬碼: **diagonal_product(matrix_1, matrix_2)**
- 例子:

$$\begin{pmatrix} 1 & 2 & 3 \\ 5 & 5 & 5 \end{pmatrix} \ltimes \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{pmatrix} = \begin{pmatrix} 1 \times 1 + 2 \times 2 + 3 \times 3 \\ 5 \times 1 + 5 \times 1 + 5 \times 1 \end{pmatrix} = \begin{pmatrix} 14 \\ 15 \end{pmatrix}$$

Propagate 函數

- **Propagate 函數**是一層神經元對輸入操作後傳遞給下一層的輸出
- 虛擬碼: **propagate(vector, weights, activation_weights, biases)**
- 我們先計算過渡矢量
- **intermediate_vector=weights@vector+biases**
- 再計算過渡矢量的激活, 輸出是個矩陣
- **activated_vector= vandermonde_matrix(intermediate_vector, activation_function)**
- 最後做個對角乘積就是我們要的輸出
- **diagonal_product(activation_weights, activated_vector)**



Softmax 函數

- **Softmax** 是一個矢量函數
- 虛擬碼: **softmax(vector, temperature)**
- 輸出是 AI 為每個可能性判斷出來是正確答案的偽概率
- 它「軟性」地提取矢量中的最大值
- **Temperature** 是溫度, 這裏的溫度只是個參數名字, 跟物理溫度沒有關係
- 溫度愈低, 矢量裏的最大值會愈接近 1, 其他值會接近 0
- 溫度愈高, 矢量裏的數值會較為平均分佈
- 溫度參數的設定是為了讓 LLM 的輸出附有隨機性更人性化
- 溫度低 LLM 的輸出會用詞較生硬和機器化, 但每次對輸入的回答較為一致
- 溫度太高 LLM 的輸出會變得語無論次, 胡說八道
- 在做圖像識別時我們可以把溫度設成 1, 因為我們永遠選最大值作為 AI 的答案輸出, 所以溫度參數對輸出沒有任何影響

神經網絡

- **神經網絡 (*Neural Network*)** 就是我們想構建的最終函數, 這函數輸入是我們最初問題的輸入, 輸出就是神經網絡思考後給我們的答案
- 未訓練好的神經網絡給出來的答案基本上是亂碼, 不能夠直接使用
- 虛擬碼: **Neural_Network(vector, weights, activation_weights, biases, temperature)**
- 首先我們輸入 **vector**
- 虛擬碼: **next_vector=vector**
- 然後, 每一層神經元的輸出傳遞給下一層
- 虛擬碼: **next_vector=propagate(next_vector, weights[k], activation_weights[k], biases[k])**
- 最後一層的 **next_vector** 輸出我們給他做個 **softmax**
- 虛擬碼: **output_vector=softmax(next_vector, temperature)**
- **output_vector** 就是神經網絡給出來的答案輸出

損失函數

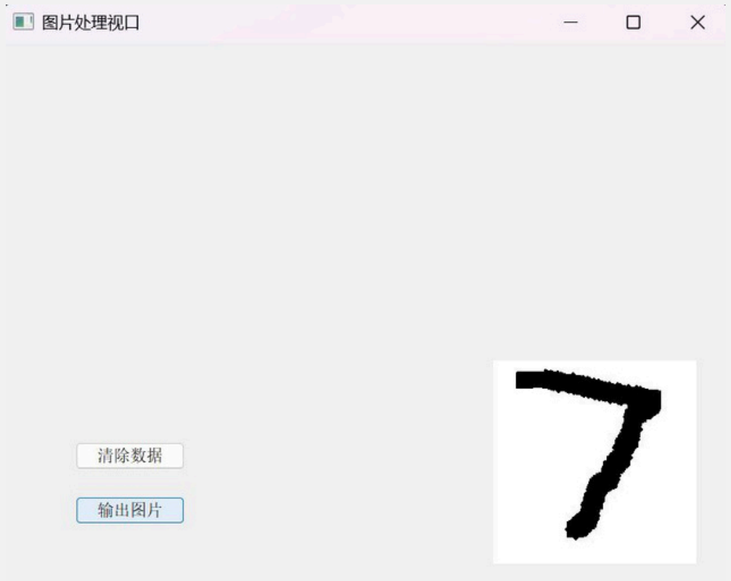
- **損失函數 (*Loss Function*)** 是把我們模型的預測跟實際值落差量化的函數
- 落差愈大, 損失值愈大
- 因此, 我們給 AI 的反饋, AI 要對神經網絡更改從而把損失最小化
- 此外, 我們可以在損失函數加上不同數項, 從而對 AI 施加不同的限制
- 損失函數是其中一個我們需要不斷更新改良的地方
- 例子: 最簡單的損失函數就是**平方損失函數 (*Quadratic Loss Function*)**

$$\text{quadratic\_loss\_function}\left(\begin{pmatrix}0\\1\\0\end{pmatrix}, \begin{pmatrix}1\\1\\1\end{pmatrix}\right) = (0-1)^2 + (1-1)^2 + (0-1)^2 = 2$$

- 虛擬碼: **quadratic_loss_function(prediction, actual)**

圖像識別

- 我們會使用網上開源的**用户介面 (*User Interface* (UI))**以及 AI 訓練資料以節省時間
- 而矩陣數據我們用 Excel 儲存
- UI 需要客製化, 在圖片處理視窗裏加設「用家反饋」以及顯示 “Loss” 值



矩陣數據儲存

- 矩陣數據我們用 Excel 儲存
- 程式必須可以從 Excel 檔取出矩陣數據
- 訓練後再把訓練後的矩陣另存新 Excel 檔

用户介面客製化

在UI裏加入;

- 用户反饋: 用户可以輸入正確答案反饋以助AI學習
- 顯示Loss值: Loss值可以方便我們進行實時數據研究

數據格式轉換

- 先把圖片轉換為黑白值矩陣
- 再把矩陣用 **reshape** 轉成矢量
- 矩陣轉成矢量沒有格式上的要求
- 所以挑選最簡單的方式即可
- 示例:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix} \xrightarrow{\textbf{reshape}} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{pmatrix}$$

參考文獻

- [1] **Self-Expanding Neural Network (SENN)**: https://arxiv.org/abs/2307.04526
- [2] **知識蒸餾 (*Knowledge Distillation*)**: https://arxiv.org/pdf/1503.02531