**銘　　傳　　大　　學**

**資 訊 工 程 學 系**

**專 題 研 究 總 審 文 件**

本校一一零學年度 資訊工程學系

組員：　　　楊子霆　　　、　　　陳維嶸　 　。

　　　張淯雯　　　、　　　游騰緯　　 。

所提專題研究：　應用Performer深度學習架構於辨識JAVA物件導向設計錯誤類型與位置之研究

指 導 教 授 ：　　　　　　王豐緒　　　　　　。

**中華民國　一一零　年　十二　月　七　日**

應用Performer深度學習架構於辨識JAVA物件導向設計錯誤類型與位置之研究

**摘要**

在大學資訊相關科系中，物件導向程式設計(Object-oriented programming)是一門重要的程式設計課程，但學生最生疏的卻是物件導向設計的相關概念以及程式的除錯。本研究的目標是讓學生在程式除錯的時候，能夠得到即時的指引與幫助。本研究使用Google推出的Performer深度學習技術，做為模型的基礎架構，訓練資料則是使用「Java程式設計」課程中學生所撰寫相關的程式碼，經過標註錯誤類別以及位置（錯誤所在的行數）進行訓練。藉由建立可輸出錯誤類別與錯誤行數的類神經模型，本研究期望可以藉此引導學生了解程式碼的錯誤與改善方法，提高學生的學習成效。

關鍵字：深度學習、Performer類神經架構、程式碼錯誤定位、Java程式設計、物件導向技術

目錄

第一章 緒論 1

1-1 研究背景與動機 1

1-2 研究目的 2

第二章 文獻探討 3

2-1 相關理論 3

2-1-1 深度學習 3

2-1-2 卷積神經網路(Convolutional Neural Network) 3

2-1-3 遞歸神經網路(Recurrent Neural Network) 3

2-1-4 LSTM 4

2-1-5 Transformer 4

2-1-6 Performer 5

2-2 程式錯誤分類與定位標註 6

2-2-1 XML 6

2-2-2 Swing 6

2-3 資料不平衡處理方式 7

2-3-1 採樣 7

2-3-2 加權 7

2-3-3 單分類(One Class Learning) 8

2-4 績效準則 8

2-4-1 綜合評價指標(F-Score) 8

2-5 相關研究與系統 8

2-5-1 程式錯誤定位(Program Error Location) 8

2-5-2 程式錯誤分類(Program Error Classification) 9

2-5-3 程式結構分類 9

2-5-4 程式錯誤預測 9

2-5-5 程式錯誤自動更正 10

第三章 研究方法 12

3-1 資料前處理 12

3-1-1 資料不平衡(Imbalanced data) 12

3-1-2 檔案分割 13

3-2 訓練樣本取得與分類 14

3-2-1 Java標記程式編輯器 15

3-3 類神經架構 20

3-3-1 類神經架構 20

3-3-2 FAVOR+(Fast Attention Via positive Orthogonal Random features) 21

3-4 損失函式 22

3-4-1 交叉熵損失函式(Binary Cross Entropy) 22

3-4-2 分類交叉熵函式(Categorical Cross Entropy) 22

3-5 績效準則 23

3-5-1 錯誤類型預測績效 23

3-5-2 位置定位預測績效 24

3-6 研究環境 25

第四章 研究結果 27

4-1 實驗與比較 27

4-2 系統實作 33

第五章 結論與未來研究方向 35

參考文獻 36

中文部分 36

英文部分 36

附錄 38

圖目錄

[圖 3-1擴增原理 13](#_Toc100331784)

[圖 3-2檔案分割與緩衝處理 14](#_Toc100331785)

[圖3‑3 Java標記程式編輯器的類別圖 16](#_Toc100331786)

[圖 3‑4 讀入Java檔 17](#_Toc100331787)

[圖 3‑5新增錯誤類型 17](#_Toc100331788)

[圖 3‑6 Xml的結構 20](#_Toc100331789)

[圖 3-7類神經架構示意圖 21](#_Toc100331790)

[圖 3-8 FAVOR+ 22](#_Toc100331791)

[圖4- 1訓練階段的錯誤分類loss值 28](#_Toc90304302)

[圖4- 2訓練階段的錯誤定位loss值 28](#_Toc90304303)

[圖4- 3訓練階段的錯誤分類accuracy值 29](#_Toc90304304)

[圖4- 4訓練階段的錯誤定位accuracy值 29](#_Toc90304305)

[圖4- 5訓練階段的錯誤分類loss值 30](#_Toc90304306)

[圖4- 6訓練階段的錯誤定位loss值 31](#_Toc90304307)

[圖4- 7訓練階段的錯誤分類accuracy值 31](#_Toc90304308)

[圖4- 8訓練階段的錯誤定位accuracy值 32](#_Toc90304309)

[圖4- 9系統架構圖 33](#_Toc90304310)

[圖4- 10錯誤辨識系統首頁畫面 34](#_Toc90304311)

[圖4- 11辨識結果為部份正確的辨識畫面 34](#_Toc90304312)

表目錄

[表 3‑1錯誤列表 14](#_Toc87967521)

[表 3‑2 說明產生出的Xml結構與屬性的意義 18](#_Toc87967522)

[表 4-1 績效結果 32](#_Toc90304363)

# 緒論

此章將敘述是什麼樣的理由促使本研究的產生，在本章可以瞭解本研究所要得到的結果為何以及我們所需具備的能力。

## 研究背景與動機

C++是我們在大學生涯中最開始接觸的語言，到大二後我們迎接新的語言—Java，但Java與C++的設計觀念相差懸殊，Java的設計風格需要靈活使用物件導向的概念，因此導致部分同學無法快速改變思考模式，最終造成學習進度落後的情況，因此我們想藉由開發智慧系統可以分析學生程式的架構，讓學生可以得知自己的程式概念邏輯中，有哪些設計的概念與邏輯不清楚或是缺點，進而使學生對物件導向（OOP)的概念能夠更精實。藉此，不僅可以幫助上課不敢舉手發問的同學，也可以幫助想精進自我的學生們，讓學生能夠更順利更有效率的得到即時的引導與教學。

我們思考著學習的方式並不一定只有聆聽教授的教學和書本上枯燥的黑白文字，在我們思考本研究題目時，發掘學長姐們有做錯誤分類，因此我們想藉由改良歷屆學長姐研究主題與方法，並加上新的功能，讓訓練模型能夠指引學生學習的方向。

本研究著重於物件導向程式架構，其中包含整理出來的35種錯誤，其餘的不在本研究範圍。若程式有語法錯誤的問題，亦不在本研究範圍內，建議可以先到編譯器除錯後，再運用本系統進行智慧型的診斷任務。

## 研究目的

本研究希望能靈活利用深度學習的Performer類神經網路架構於教育體系上，藉由歷屆學生在Java程式設計課程中的程式碼並利用深度學習中的Keras套件，訓練模型對於Java程式架構錯誤辨別與定位的準確度，並設計即時且專屬的互動介面，企圖將該模型發展為輔助學生的最佳利器，以提升學習成效。

適用對象：正在學習物件導向設計的學生及想增進物件導向的使用者

研究科目：物件導向技術

# 文獻探討

## 相關理論

* 1. **深度學習**

深度學習是機器學習的其中一項，其使用類神經網路進行資料表徵學習的演算法，並能建立較好的模型，以進一步應用在各項任務上(如分類等)。深度學習透過表徵學習，避免了手動提取特徵的麻煩，允許計算機學習使用特徵的同時，也學習如何提取特徵。

* 1. **卷積神經網路(Convolutional Neural Network)**

卷積神經網路(CNN)是一種前饋神經網路，多層卷積、池化層及全連接層為主要組成。卷積神經網路的人工神經元的主要功能是使電腦能自動選取出有用的特徵取代人工選取的特徵，卷積層透過學習卷積核權重挑選出有用的資料，當成類神經網路的輸入特徵，池化層利用濾波器取出部分區塊最大值或平均值取代濾波器內的所有值來降低採樣。這兩層使卷積神經網路可以利用輸入資料形成的二維陣列，在圖片、影像、語音辨識和自然語言方面給出更好的結果，不像其它神經網路只是擷取資料進行運算。CNN對於影像辨識這部分的表現較為突出，從卷積核擷取影像的局部特徵，且讓影像各區共享此卷積核，即能改善類神經網路將影像放大成1×N向量時，輸入資料失去局部關聯性的問題。

* 1. **遞歸神經網路(Recurrent Neural Network****)**

遞歸神經網路(RNN)是用於處理序列資料的神經網路，其原理是將神經元的輸出再回饋給神經元的輸入，使網路的輸出取決於之前的計算，讓神經網路具有記憶的功能。然而，遞歸神經網路因長序列訓練過程中的梯度消失和梯度爆炸的問題會造成效能不彰。

* 1. **LSTM**

LSTM改善RNN在長期記憶的不足，最重要的觀念是在分析上層和目前層的資訊時，可以透過邏輯函數來調整與控制，且決定是否遺忘上層的資訊，同時能在更長的序列中能有更好的訓練結果。然而，LSTM因為計算結構與時序的問題，因此無法將GPU的平行計算能力充分利用，而讓訓練速度大幅降低。

* 1. **Transformer**

Transformer [1] 是一種利用注意力機制的深度學習模型。注意力機制使模型能直接觀察句子中任何較早的狀態並從中提取狀態，從而提供有關遠距離相關令牌的更清晰的訊息。

另外，Transformer和RNN不同的是不需要順序處理順序數據。例如，如果輸入數據是自然語言的語句，則Transformer能夠先處理結尾在處理開頭。由此可見，Transformer相較於RNN來說，其同意更多的平行運算，因而減少了訓練時間，模型訓練速度更快，結果也有可能更好。

然而， Transformer需要高效的稀疏矩陣乘法運算，而這些運算並不是所有加速器都能實現，而且通常需要疊加更多注意力層來補償稀疏表示，使得難與其他訓練模型一起使用，因此需要重新訓練且會消耗大量能量。最後，Transformer的時間複雜度為，訓練起來較耗時且空間所需也較大[2]。

Transformer在NLP領域中，在記憶體的空間佔用上不像長短期記憶（Long Short-Term Memory）如此龐大。LSTM即是雙向的RNN結構，以大規模平行計算來說RNN的序列結構對於執行時間與記憶體占用上需要很大的空間，RNN因其結構問題沒辦法完全利用硬體的平行計算能力，反觀Transformer就不存在序列問題，對其來說每個時間操作可以並行一起計算。

Transformer的好處在於可以使用自注意力層(Self\_Attention)以及Multi-head Self Attention 進行語言意義的抽取（關於NLP句子中依賴語意特徵的問題，自注意力層本身就可以解決該問題，但因為在集合訊息的時候，當前面的單字和句子中隨意單字皆會產生連結，因此提前把這件事情解決），並透過正弦位置編碼保留輸入句子單字、詞之間的相對位置訊息，這一套處理程序非常強大，不像RNN和CNN必須經過隱藏層節點序列依序往下層傳，和需要透過增加網路深度來選取較遠距離的特徵，Transformer在這部分看起來明顯是相對直觀。

* 1. **Performer**

Performer [8] 使用了一個有效的廣義注意力框架，可以實現不同的相似性度量的注意力機制。廣義注意力機制為在原有的注意力機制中，透過序列自身轉換出的query和key序列分別對應於矩陣的行和列，進行相乘並透過SoftMax形成一個注意力矩陣，做為相似性分數(score)。這種方法可以將注意力矩陣分解為原始query和key的隨機非線性函式的乘積，即為隨機特徵，進而更有效地對相似的資訊進行編碼。透過此種矩陣分解，得以使用線性儲存隱式注意力矩陣，同時也可以得到一個線性時間的注意力計算機制，透過重新排列矩陣乘法來逼近常規注意機制的結果。

Performer能更有效的利用空間，降低記憶體過載的可能，故本研究決定採用Performer作為類神經訓練模型架構。

## 程式錯誤分類與定位標註

### XML

XML(Extensible Markup Language)是一種標記式語言，XML設計用來傳輸和儲存資料，其焦點是資料的內容，且XML允許創作者定義自己的標籤和自己的文件結構，對XML最好的描述是:XML是獨立於軟體和硬體的資訊傳輸工具。XML在Web中起到的作用不會亞於一直作為Web基石的HTML，且XML也是各種應用程式之間進行資料傳輸最常用的媒介。

本研究之所以使用XML，是因為XML便於整理與存取資料，我們可以利用XML將資料分類並整合起來後，便可依照標籤屬性提取任何需要的資料。

### Swing

Swing是Java為圖形介面應用開發提供的一組工具包，是Java基礎類的一部分，Swing包含了構建圖形介面（GUI）的各種元件，一個Java 的圖形介面，由各種不同型別的“元素”組成，例如: 視窗、選單欄、對話方塊、標籤、按鈕、文字框等等，這些“元素”統一被稱為元件（Component），元件按照不同的功能，可分為頂層容器、中間容器、基本元件。

本研究使用到了Swing內的JFrame元件(頂層容器)來設計標註程式編輯器，此元件能讓程式產生視窗，並在此之上再加上按鈕、選單等元件(基本元件)，使得編輯器能擁有自己的GUI，這樣能讓我們在標註程式時更加方便使用且能更快速完成。

## 資料不平衡處理方式

### 採樣

採樣(Sampling)方法是處理資料不平衡較常用的方法。通過對訓練資料集進行抽樣處理，以減緩資料不平衡的數據集變成平衡的數據集情況。常見的採樣分為Oversampling和Undersampling。

Oversampling是把種類較少的樣本複製成較多份，一個節點會在多維空間中重複出現，會導致分點的對錯非常兩極，以至於訓練出來的模型將會有Overfitting的可能性，為了能有效解決這個問題，可以在每次生成新資料時加入些微的隨機擾動。

Undersampling是從種類較多的樣本剔除些許，或是從種類較多的樣本中選取部分，其缺點是訓練集失去一些數據，可能讓模型只學到了總體模式的一部分。有三種方法能減少信息的損失，第一種方法是利用模型融合的方法（Ensemble）：多次Undersampling放回採樣，如此一來產生的訓練集才相互獨立，且產生多個不同的訓練集，進而訓練多個不同的分類器，透過組合多個分類器的結果得到最終結果。第二種方法是BalanceCascade，利用增量訓練的思想（Boosting）：先透過一次Undersampling產生訓練集，訓練一個分類器，對於分類正確的大眾樣本不會放回，再對更小的大眾樣本Undersampling產生訓練集，訓練第二個分類器，以此類推，最終組合所有分類器的結果得到最終結果。第三種方法是利用KNN挑選最具代表性的大眾樣本，稱NearMiss，這類方法計算量很大。

### 加權

除了以上兩種的方法以外，還可以透過加權來解決資料不平衡的問題，也就是說對不同類別依照其數量產生不同的比例。此方法需要根據實際情況來設定權重值，然而，難點在於如何設置合理的權重及實際應用中讓各個分類之間的加權損失值近似相等。

### 單分類(One Class Learning)

對於正負樣本不平衡的狀況，可以換不同的角度來看待這個問題：把它看作單分類（One Class Learning）或異常檢測（Novelty Detection）問題。重點是把其中一類樣本建模，以下簡單介紹最常見的One-class SVM。

使用One-class SVM的方式非常簡單，尋找一個超平面將樣本中的正例挑選出來，預測的方法是用這個超平面做決策，在挑選出的樣本即認為是正樣本。

## 績效準則

* 1. **綜合評價指標(F-Score)**

*F*-Score亦稱作*F*-Measure，因為準確率及召回率可能會有矛盾的情形發生，為了解決這個問題，常用*F*-Score，如公式(1)，來作為指標，若*F*-Score越高證明模型越有效。而F-Score是準確率及召回率的加權調和平均。

(1)

若*F*-Score沒有特別定義時，則是準確率及召回率的權重皆設為1(公式2)，可以說是準確率及召回率的調和平均，公式如下:

(2)

## 相關研究與系統

* 1. **程式錯誤定位(Program Error Location)**

在程式撰寫的過程中，程式中一定有些錯誤，但是要解決程式中的錯誤，必須先找到錯誤的位置。要找出錯誤位置的過程十分不易且費時，目前已經研究出數種錯誤定位技術來幫助程式開發人員來追蹤錯誤位置，但可以自動修改錯誤問題的技術及研究卻很少。例如:Wong等人用徑相基函數(Radial Basis Function)神經網路 [13]，其特點為不用檢查太多的程式碼就能發現錯誤，這樣來做也比較有效率。

* 1. **程式錯誤分類(Program Error Classification)**

在程式碼的自然語言中，變數和函數的名稱可以傳達有效的訊息，但是有些除錯的工具會無視這些訊息，導致錯過一些錯誤分類。因此Pradel 和Sen提出DeepBugs[12]，是一種偏重變數或函數名稱的除錯學習方法，並自動學習錯誤檢驗。把錯誤問題的檢驗制定為二元分類，分別為正確程式碼和錯誤程式碼。為了解決讓其有效率的學習，需要正確和錯誤程式碼範例，從已存在的正確程式碼中，建立不正確的程式碼範例。從人工建立的錯誤程式碼，學習錯誤檢驗，產生錯誤檢驗器，使檢測器得以將實際程式碼中的錯誤解決，並獲得具有高準確度的錯誤檢驗器。

* 1. **程式結構分類**

程式語言的處理在人工智慧的領域日漸重要。因而，Mou等人提出一種樹結構的卷積神經網路(TBCNN)用於處理程式語言[10]，最重要的是以樹的結構進入類神經網路。研究顯示其在兩種不同程式分析的有效性，對程式進行分類並檢驗部分程式碼。

* 1. **程式錯誤預測**

為提升軟體穩定性，常會利用程式錯誤預測協助開發人員找出潛在的程式碼錯誤，但較早的錯誤預測是利用人工的方式來標記特徵，再將特徵輸入到機器學習分類器中來辨識錯誤程式碼。Li等人提出一個透過卷積神經網路(CNN)的錯誤預測[9]，並透過程式的抽象語法樹(AST)擷取具代表的向量，再藉由映射(Mapping)和嵌入(Embedding)把擷取的向量轉化為數值向量，再把數值向量輸入到卷積神經網路中讓它可以自動學習語意，最後將學習的特徵與先前人工標記的特徵結合，達到更準確、可信任的錯誤預測。

此外，許多開發的預測模型並無考慮成本問題，Zheng認為對程式錯誤的預測把程式模型分為較容易出現的錯誤與不容易出現的錯誤可以維持程式預測的高品質，容易出現錯誤的模型在成本上通常較高，因此提出了成本導向(Cost-sensitive)Boosting神經網路對於程式錯誤的預測 [7]，把錯誤分類的成本算進程式錯誤預測模型中。由於臨界值移動 (Threshold-moving)的算法在錯誤分類成本產生改變時，基本神經網路分類器不需要重新訓練，所以錯誤分類的成本相對較低，此算法是使用增強型神經網路建構成本導向的程式錯誤預測模型的好選擇。

* 1. **程式錯誤自動更正**

程式除錯時，程式設計師經常在程式碼中尋找錯誤的地方並檢查更正程式碼能否將解決錯誤。即使有許多工具可以用在檢驗程式錯誤和錯誤位置的追蹤，不過能自動修正錯誤問題的技術及研究都非常少，所以，Kern和Esparza提出Java程式碼的自動錯誤更正[4]，經過語法分析來找尋程式碼錯誤的位置，並對每個錯誤提出替代方案，再透過迭代法產生不同的變更集。測試程式輸入，搜索透過變更集程式碼替換的組合所得到的程式，選擇具備所有條件的變體。此研究假設程式錯誤是因為差一錯誤(Off-by-one error)引起，將有40%的錯誤可以達到錯誤自動更正。程式錯誤的更正，eddy系統[5]也有類似的功能，是IntelliJ提供的系統，能把錯誤的Java程式碼自動修正為符合規則的Java語法，同時也能糾正一些常見的語意錯誤。因為eddy系統理解許多常見的語言構造，所以若只習慣撰寫C++的語法，eddy系統也能將此問題解決。eddy系統僅在Java編輯視窗中有效，並且只在程式碼編輯區內部有效。

# 研究方法

本研究以「Java程式設計」的課程歷屆學生寫的程式碼當成測試樣本進行類神經模型訓練，此研究將以Performer建立程式碼的錯誤¬¬分類。由於Python提供相關的下載套件，錯誤分類模型搭配Keras套件，因此我們將運用Python做為程式開發工具。

## 資料前處理

* + 1. **資料不平衡(Imbalanced data)**

**1. 採樣(OverSampling)**

因為上述三種作法皆為案例只有一項錯誤時，用起來較為方便，但是在我們的研究中，案例往往不只一種錯誤，故最後決定使用平均犯錯比例(R’，公式3)來算出擴增比例。利用案例中有錯的類別()及此種類別在所有案例中的比例()來做計算。計算後得到每一個案例的R’，因為要將每一個類別的數量擴增到大約相等，故會將R’倒數後計算其比例，並利用此比例擴增。

(3)

**2. 資料擴增**

本研究採用王豐緒(2019)[6]的原始碼擴增程式，分成兩個部份來做，第一個部分是符合javalang語法的規則的原始碼，先將該程式正規化，目的是為了方便計算行數，第二步是將程式碼的順序調換，並產生出P為指令排列的結果，第三步是更改變數名稱，並產生出R為替換名單，做完以上步驟即是完成擴增程式。

第二部分是針對不符合javalang語法的規則，首先要用人工的方式將不符合javalang語法的地方改掉，第二步是將程式正規化，在此做正規化的目的同樣是為了方便計算行數，並計算和一開始的程式做比較回推算出替換規則()，並將其記錄在替換規則(S)的字典裡，第三步是將程式碼的順序調換，並產生出P為指令排列的結果，第四步是更改變數名稱，並產生出R為替換名單，做完以上步驟會產生出擴增的程式，因為此程式是從正規化擴增而來，故我們須回到正規化的部分並還原程式的位置，下一步是將人工修改的部分還原，再將其做位置的替換，即完成擴增程式的所有步驟。

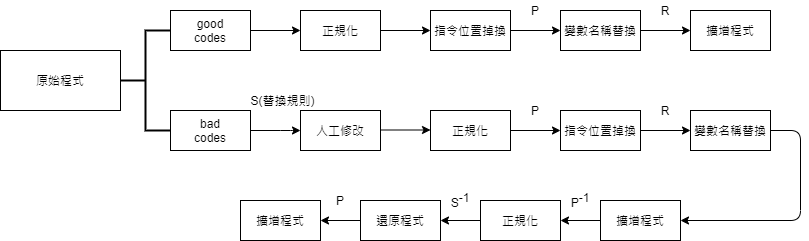


圖 3-1擴增原理

* + 1. **檔案分割**

將程式擴增後，因為資料筆數過於龐大，導致機器無法將資料完全讀入，故本研究決定將檔案做切割，並採用資料緩衝區的技術，讓深度學習模型得以完整學習(如圖3-2)。步驟是先將擴增後的資料份數，依照上述的擴增方法算出的比例()乘以500(份)，並取出該案例中的前*r*份，依序放入特定大小的硬碟Block，再將模型所需的資料放入兩記憶體的Data Buffer中。若模型需要的ID在Data Buffer中找不到，則會回到硬碟Block中找到其ID所在的Block並放入Data Buffer，如果Data Buffer裡的兩個Block已經放滿，且沒有模型所需的ID，則要計算每一個Block未被抽取的機率，機率越低的表示Block已經被抽取的次數越多，應該優先被替換。

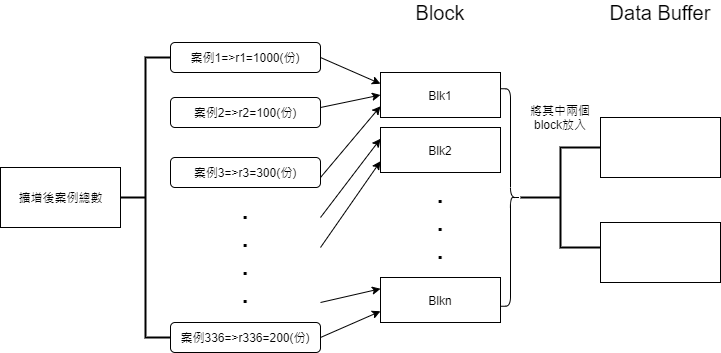


圖 3-2檔案分割與緩衝處理

## 訓練樣本取得與分類

研究人員先對訓練樣本進行檢閱，再根據老師提供的參考答案、執行結果再加上我們對於Java的概念進行錯誤分類的歸類，產生錯誤分類表(如表3-1)。我們依據學長姐分類出的錯誤列表，加上一些之前沒有被提出的錯誤類型於下表中，目的是為了讓類神經模型學習得更加準確，我們最終統計出了36種錯誤類別:

表 3‑1錯誤列表

|  |
| --- |
| **錯誤列表** |
| 正確:正確 |
| 變數:重複宣告變數 |
| 變數:變數未定義無法使用 |
| 變數:不了解局部變數的概念 |
| 運算:運算錯誤:型態不符無法運算 |
| 函數:未傳遞參數或傳回值 |
| 函數:函數庫使用錯誤 |
| 函數:函數結構不清楚 |
| 函數:函數概念不清楚 |
| 字串:字串比較錯誤(沒有使用equals函數) |
| IO:不會使用輸入scanner類別 |
| IO:重複宣告scanner物件 |
| 物件:物件動作概念(建構子) |
| 物件:物件資料/物件動作概念 |
| 物件:正確的產生物件 |
| 物件:無法分辨物件&類別的資料/動作 |
| 物件:建立物件之間的關係 |
| 物件:不會使用物件之間的關係 |
| 物件:不會使用物件資料 |
| 陣列:不會宣告陣列變數 |
| 陣列:陣列索引值 |
| 陣列:陣列溢位 |
| 其他:撰寫main函式 |
| 其他:關鍵字錯誤 |
| 其他:筆誤 |
| 其他:不熟悉Java程式結構 |
| 繼承:使用繼承 |
| 繼承:使用super產生繼承物件 |
| 抽象:繼承資料或動作 |
| 抽象:抽象動作 |
| 抽象:不會分離變跟不變的部分 |
| 介面:Comparable介面 |
| 介面:Comparable介面的排序應用 |
| 迴圈:迴圈概念錯誤 |

參考學長姐分類及我們新增的錯誤列表

### Java標記程式編輯器

下圖3-3為說明Java標記程式編輯器的類別圖，Java標記程式編輯器是設計用來建置訓練資料，利用Java Swing提供的套件設計出一個介面化程式，讓小組成員可以透過這個介面化程式的功能更有效率地標註每個Java程式的錯誤類型與錯誤出現的起始位置，且生成一份xml格式的檔案提供之後的訓練資料的建置。

Java標記程式編輯器是我們使用Blue J這套軟體所編寫的介面化程式我們引用了JFrame的套件來架構出整體視覺化程式的框架。Java標記程式編輯器包含了如下功能:

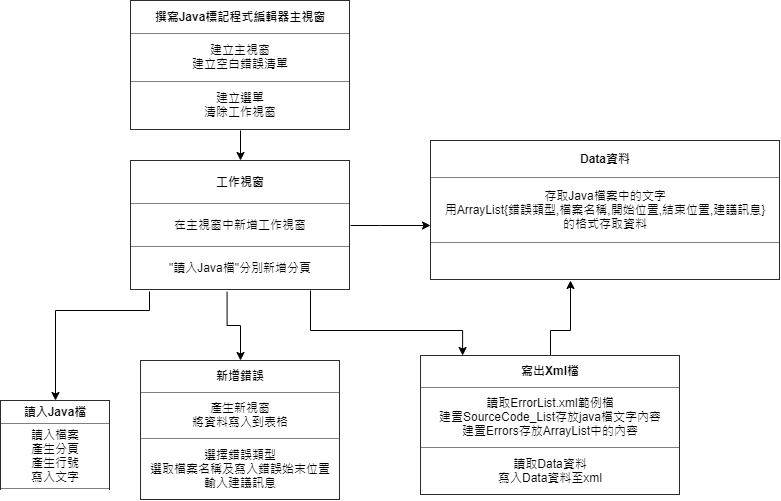


圖3‑3 Java標記程式編輯器的類別圖

**讀入Java檔**

可以選取數個要標記的Java程式(如圖3-4)，並將每個Java檔以分頁的形式呈現，為了方便標註，我們會在程式旁邊標示行號

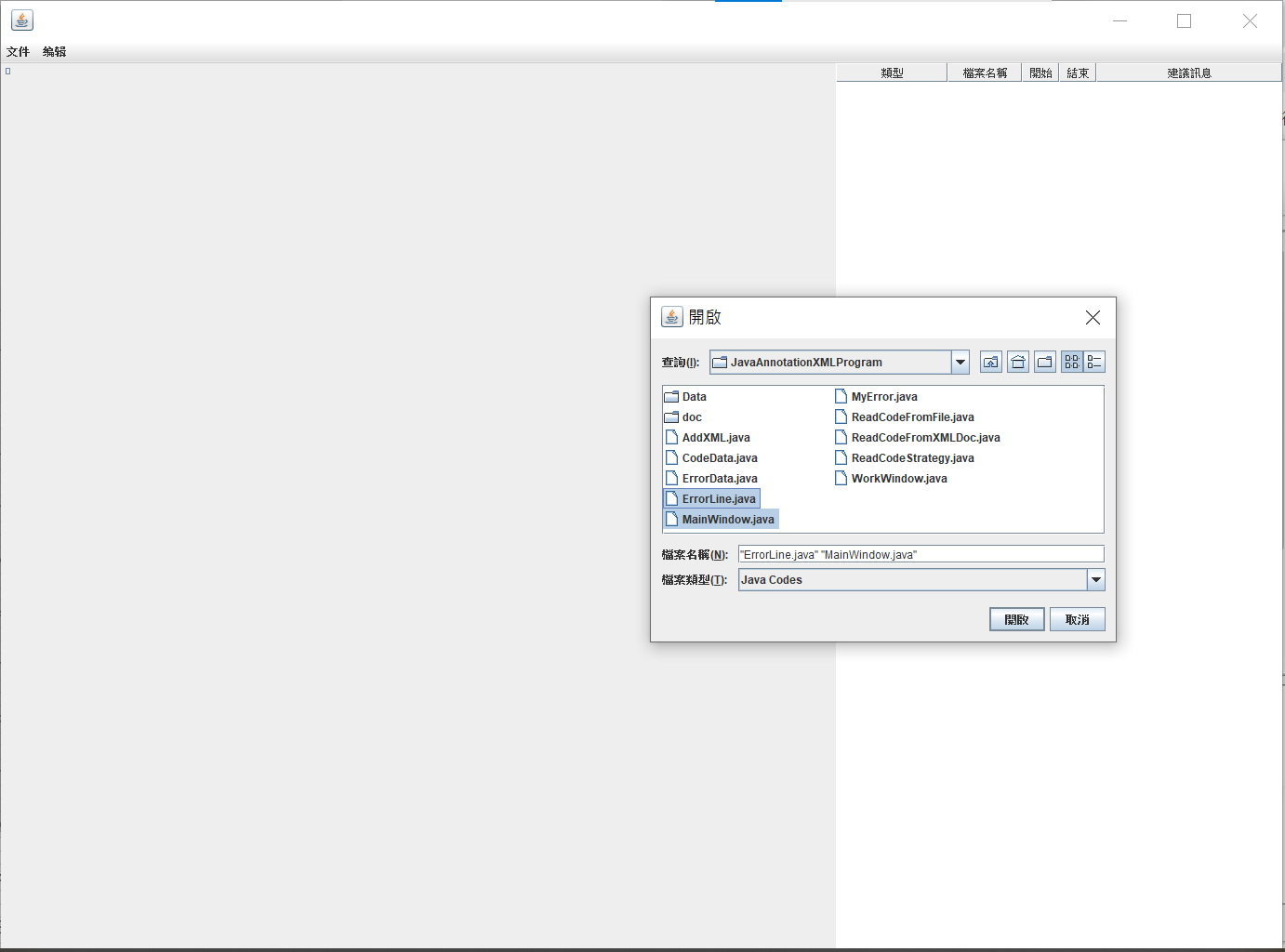


圖 3‑4 讀入Java檔

**新增錯誤**

在編輯的地方新增一個新視窗，其中包含：錯誤類型的下拉式選單、開始與結束行號的文字方塊、輸入建議訊息的文字區域、將錯誤以表格的形式呈現(如圖3-5)。

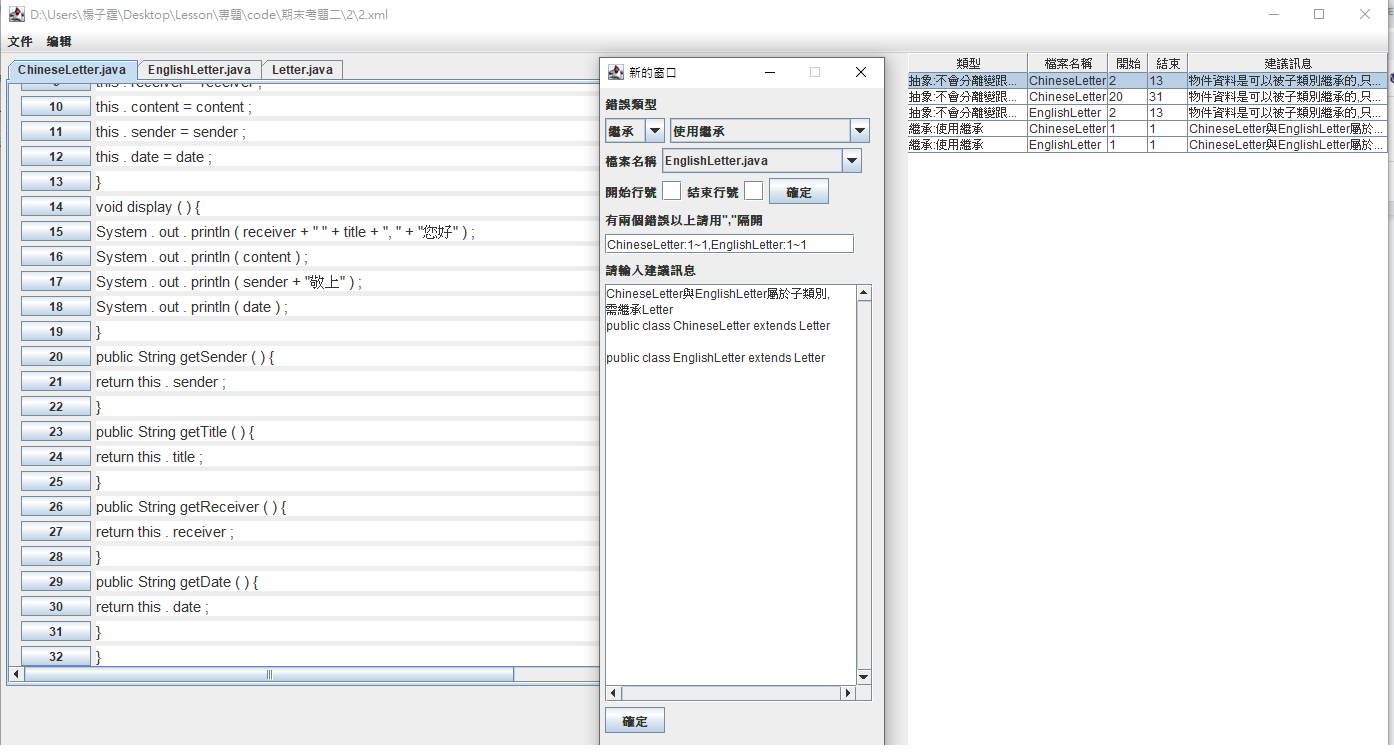


圖 3‑5新增錯誤類型

**寫出Xml檔**

我們運用Xml標記式語言的結構，使資料明確分類，下表3-2為Xml的結構：

表 3‑2 說明產生出的Xml結構與屬性的意義

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> |
| <ErrorList id=""> |
| <SourceCode\_List> |
| <SourceCode name="" lines=""> |
| <CDATA[]]> |
| </SourceCode> |
| </SourceCode\_List> |
| <Errors> |
| <Error tpye=""> |
| <Message> |
| <![CDATA[]]> |
| </Message> |
| <Linelist> |
| <line src="" Begin="" End=""/> |
| </Linelist> |
| </Error> |
| </Errors> |
| </ErrorList> |

ErrorList標籤內的id屬性表示此檔案的路徑。

SourceCode\_List為存放所有Java程式碼的標籤。

每個Source標籤存放一個Java類別程式碼，其中name屬性表示此類別的名稱，lines屬性表示此類別的行數。

SourceCode的CDATA內為存放整個Java類別原始程式碼的地方，之所以使用CDATA是因為Java程式碼內有很多會令XML判斷錯誤的特殊符號，而將程式碼放入CDATA內時XML就會忽略CDATA內的內容。

Errors標籤為存放所有錯誤類別跟錯誤訊息的地方。

Error標籤內的type屬性表示此錯誤的分類，錯誤訊息則會存放在Message標籤中的CDATA內。

Linelist標籤為存放錯誤位置的地方。

line標籤內的src屬性表示錯誤的類別的名稱，Begin屬性表示錯誤的起始位置，End表示錯誤的結束位置。

![一張含有 文字 的圖片

自動產生的描述

圖 3‑6 Xml的結構

## 類神經架構

本節將提到本計畫規劃採用的類神經架構及深度學習模型之中的實作過程。

### 類神經架構

本研究使用的類神經模型由一組Performer編碼器(Encoder)及兩個分類器組成，如圖3-7。首先，先將每份Java程式碼經過前處理合成一份文字檔，並經過tokenize及embeding轉換為Encoder輸入。輸入資料經由Encoder中的矩陣運算，會得到的輸出是一組包含原始碼及個別資料的編碼序列資料。Encoder的輸出會分別連接到錯誤類型分類器(Error Type Classifier)及錯誤位置分類器(Line Block Classifier)。

本研究設計兩種分類器，第一種為錯誤類型的分類器，輸入為經過自注意力機制的序列，利用其中的向量權重讓類神經學習判斷出這份資料出現了哪些錯誤類型。在輸出層對應的類別神經元產生36個節點，在這36個節點中，有出現此錯誤輸出為1、沒有出現此錯誤輸出為0。

第二種為錯誤位置的分類器輸入為經過自注意力機制的編碼器(Encoder)輸出序列與錯誤類型分類器的輸出向量。加入錯誤類型分類器的輸出向量是為了提高錯誤位置分類器的準確率，讓定位分類器能在分辨錯誤位置也能與錯誤類型分類器輸出的錯誤類型相對呼應。本研究使用陣列的方式來呈現起始與結束的位置，程式會將第一次偵測到的1作為開始位置(行號)，下一次偵測到的1作為結束位置(行號)，以此類推，直到陣列裡的160個輸出判斷完為止(最多偵測80個錯誤區段)。

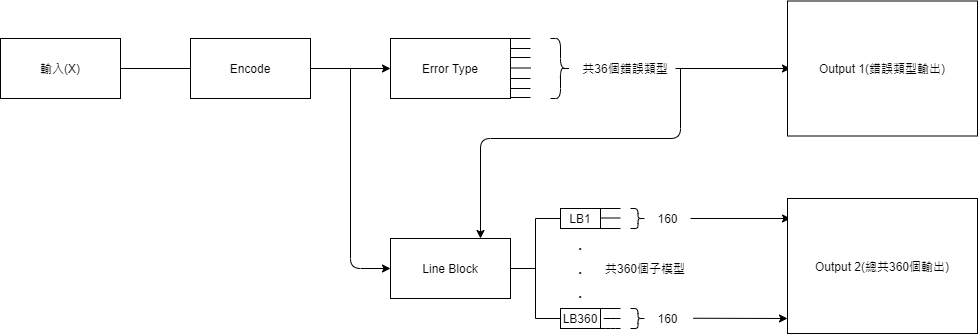


圖 3-7類神經架構示意圖

### FAVOR+(Fast Attention Via positive Orthogonal Random features)

下圖3-8為說明自注意力機制示意圖，自注意層將輸入原始程式碼。因為原本Transformer會花較多的時間和空間，其中*L*表示程式中的序列token的長度，還要乘上embed\_dim的大小，其時間複雜度為O，然而為解決時間及空間的問題，本研究使用Google團隊發表的Performer其注意力層能更有效的利用時間及空間(如圖3-8)，將*A*分解成*Q*’及*K*’並照虛線框的指示進行矩陣乘法，依照此方法我們得到的時間複雜度大大縮減為*O*(*Lmd*)，其中*m*為常數，是隨機特徵函數的個數。

一張含有 文字, 時鐘 的圖片

自動產生的描述

圖 3-8 FAVOR+(資料來源: https://arxiv.org/pdf/2009.14794.pdf)

## 損失函式

損失函式是機器學習的重要概念之一，透過計算損失函式的大小，是機器學習的過程中的重要依據及判斷演算法優劣的重要依據。

* + 1. **交叉熵損失函式(Binary Cross Entropy)**

交叉熵損失函式常用於二元(0或1)分類的應用程序，但要用在多分類的應用程序也不會有問題。因為本研究中的錯誤類別只有對或錯兩種答案，因此我們使用交叉熵損失函式讓機器可以學習的更好以及計算此模型的優劣。其中target為我們輸入的正確答案，要特別注意的是target在使用前必須先進行one-hot編碼，output為模型預測的答案，先將output放進sigmoid函數(公式4)計算出預測數值的分數，再利用Binary Cross Entropy(公式5)計算出二元交叉熵，公式如下:

(4)

(5)

* + 1. **分類交叉熵函式(Categorical Cross Entropy)**

分類交叉熵函式常用的定義是在多分類的概率分布上，而概率分布則要求兩個標籤或以上的值相加等於1，而Softmax變換就有這個作用，做完上述的動作後，再計算Categorical Cross Entropy(公式6)即為使用此函式的方法。在進行以上動作前，要先將所有的label進行one-hot編碼。因為本研究的錯誤段落之行數分別是開始及結束，因此使用分類交叉熵函式來學習及計算優劣。

(6)

## 績效準則

* + 1. **錯誤類型預測績效**

錯誤類型包括正確的類型總共有36個，本研究將目標結果及預測結果利用陣列的方式輸出，再將兩個陣列對照後，計算其預測為包含錯誤且實際含有錯誤的比率(TP)、預測為不包含錯誤且實際沒有錯誤的比率(TN)、預測為不包含錯誤但實際含有錯誤的比率(FP)、預測為包含錯誤但實際沒有錯誤的比率(FN)的數量，計算準確率(Accuracy，式7) 、召回率(Recall，公式8)以及綜合評價指標(F-value，公式9)做為模型預測績效的指標。因為召回率與準確率需要取捨，且我們認為召回率和準確率相比，召回率較重要，所以本研究著重於召回率權重較大的調和平均數，也就是綜合評價指標作為模型績效評估的參考指標，使用的公式如下:。

(7)

(8)

(9)

* + 1. **位置定位預測績效**

錯誤定位的部分，本研究使用陣列作為模型的輸出，陣列裡第一個標註1的地方為開始位置，然而遇到下一個1時即為結束位置。利用這些開始及結束的位置來計算上方提及的準確率及召回率。

* 1. **研究環境**
* Performer

為了保持開發環境與狀態的一致，本研究在開發performer時，利用anaconda3-2020.07版本，並利用該系統附帶的python3.8進行開發。

* Java

本研究開發標註程式與生產訓練樣本使用blue j5.0.0以上版本對平台開發

以確保擁有最穩定的環境與穩定性。

* 硬體實驗環境：

本組因206只有一台電腦供老師的小組使用，故自行組裝深度學習機進行實驗。

* 1. 硬體環境：
     1. CPU: AMD 5950x
     2. GPU: Nvidia RTX 3090
     3. RAM: DDR4 3600 64GB
     4. SSD: Seagate Firecuda 530 & WD SN 750
  2. 深度學習環境

本研究利用本機安裝Anaconda 後需執行Tensorflow 進行學習，在進行Tensorflow 深度學習前，我們會需要進行一些前置作業，我們需要安裝Microsoft Visual Studio 安裝C++ Compiler，確保有完整的C++ library，要利用Tensorflow做機器學習，我們還需要CUDA ，Tensorflow 目前支援至11.2版，我們選用CUDA 11.2.2(update 2)，以確保有最完善的運算環境，不過要利用GPU快速運算大量的樣本，我們還需要cuDnn的library 驅動顯卡，Tensorflow 支援至8.1，我們利用cuDnn 8.1.1(update 1)進行安裝。

* 1. Tensorflow &Keras 環境:
     1. Tensorflow 2.5.0rc0
     2. Tensorflow-GPU 2.5.0rc0
     3. Keras 2.4.3
     4. Keras-pos-embed
     5. Keras-multi-head
     6. Keras-embed-sim

1. **研究結果**

本章根據不同參數利用anaconda平台上做模型準確度比較，包括程式碼最大長度、學習速率等。紀錄結果比較後，本研究採取準確度較高的參數，做為系統實作的模型。

* 1. **實驗與比較**

本系統將所有樣本分為訓練資料90%，測試資料10%，並將訓練資料再分出10%作為驗證，最終將分割完的資料放入模型進行訓練。

本研究將對模型進行長度小於300及1000的訓練。

圖4-1至圖4-2可以看到長度小於300的模型訓練過程的loss變化，圖4-3至圖4-4可以看到模型訓練過程的accuracy變化。在圖中可以看出約在175圈時有機會得到較優良的模型，但此模型並無符合理想預測，目前造成此問題的原因尚未找到解答。

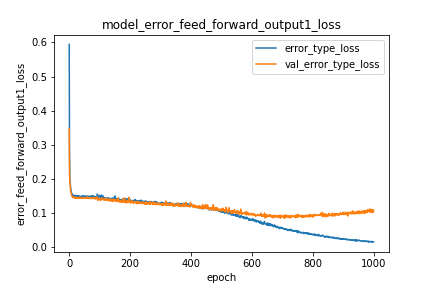


圖4- 1訓練階段的錯誤分類loss值

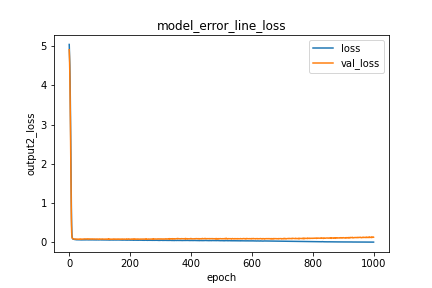


圖4- 2訓練階段的錯誤定位loss值

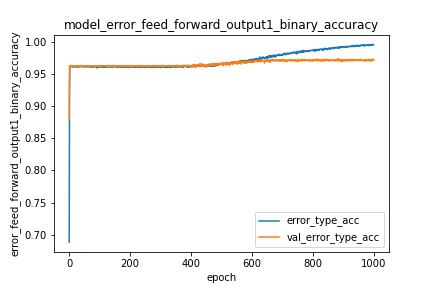


圖4- 3訓練階段的錯誤分類accuracy值

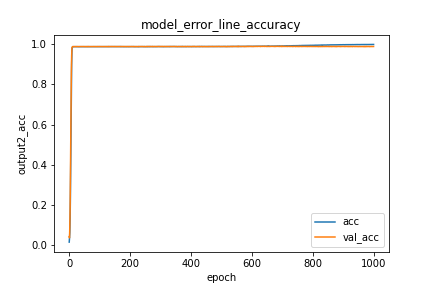


圖4- 4訓練階段的錯誤定位accuracy值

圖4-5至圖4-6可以看到長度小於1000的模型訓練過程的loss變化，圖4-7至圖4-8可以看到模型訓練過程的accuracy變化。在圖中可以看出約在175圈時有機會得到較優良的模型，但此模型並無符合理想預測，目前造成此問題的原因尚未找到解答。

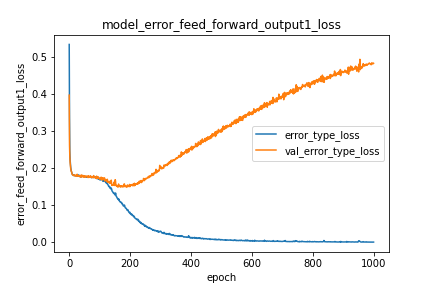


圖4- 5訓練階段的錯誤分類loss值

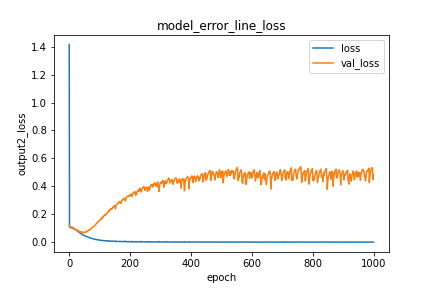


圖4- 6訓練階段的錯誤定位loss值

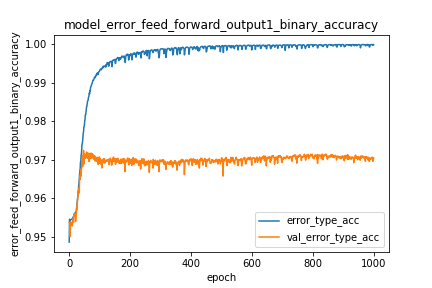


圖4- 7訓練階段的錯誤分類accuracy值

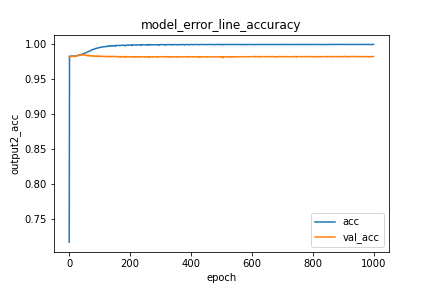


圖4- 8訓練階段的錯誤定位accuracy值

本研究主要採用準確率、召回率及綜合評價指標來做為績效結果，將其訓練及測試結果的平均分數製作成表格(如表4-1)所示。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | 最長<300 | | | 最長<1000 | | |
| Train | Validation | Test | Train | Validation | Test |
| 錯誤分類 | Precision | 0.759 | 0.756 | 0.716 | 0.997 | 0.654 | 0.646 |
| Recall | 0.585 | 0.567 | 0.522 | 0.997 | 0.657 | 0.655 |
| F2 | 0.633 | 0.618 | 0.574 | 0.997 | 0.656 | 0.653 |
| 錯誤定位 | Precision | 0.990 | 0.989 | 0.988 | 0.999 | 0.983 | 0.984 |
| Recall | 0.992 | 0.991 | 0.990 | 0.999 | 0.985 | 0.986 |
| F2 | 0.991 | 0.990 | 0.989 | 0.999 | 0.984 | 0.986 |

表 4-1 績效結果

* 1. **系統實作**

本系統架構如圖4-9所示。我們使用Python中的Flask套件來架設網站，並將結果利用網頁展示。Flask是Python中用來撰寫Web應用程式的框架，然而我們訓練類神經時也是使用Python語言，如此能減少在統合上造成的不便。

我們先將先前訓練完的模型載入，接著啟動網頁伺服器，使用者可以選擇檔案上傳或是程式碼輸入，再利用模型預測結果，並將預測結果呈現在網頁上，除了預測結果之外，我們還會將使用著的錯誤行號標示出來，以便使用者了解自己哪部分的觀念不足，藉此可以上網搜尋相關文獻來補足自己的缺失。

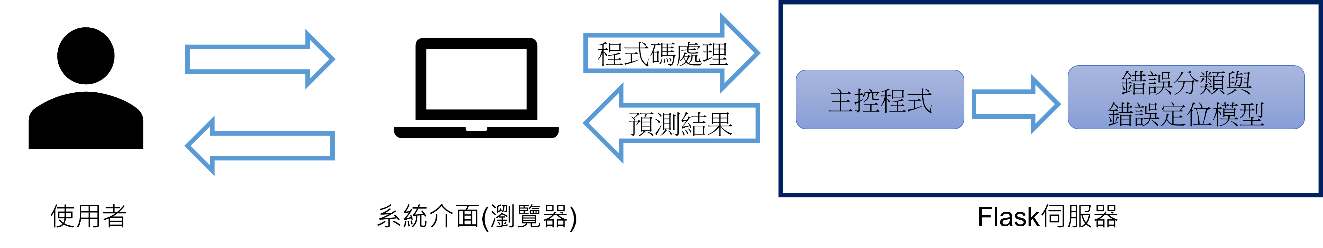


圖4- 9系統架構圖

本研究的錯誤辨識系統首頁，如圖4-10，學生可以利用選擇檔案上傳的方式來使用這個網頁，圖中顯示的是選擇檔案上傳，當選擇想要辨識的檔案後，中間的介面會顯示所選擇的程式碼。

一張含有 文字 的圖片

自動產生的描述

圖4- 10錯誤辨識系統首頁畫面

圖4-11的辨識結果中有顯示出錯誤類別及行號，在行號後面的數字表示的是回報模型輸出的信賴度，讓使用者可以參考這個行號部分的對與否。

一張含有 文字 的圖片

自動產生的描述

圖4- 11辨識結果為部份正確的辨識畫面

1. **結論與未來研究方向**

本研究是運用深度學習類神經網路，結合程式碼的錯誤分類與標註段落資料訓練程式碼的辨識能力，讓使用者了解自己Java程式架構的某些段落出現了問題，告知使用者這屬於物件導向結構中的錯誤類型，並完成「物件導向程式設計」課程專屬的輔助教學媒體。

因為程式碼有各式各樣的錯誤，導致很多的錯誤樣式範例過少。因此資料錯誤分類完成時，透過運算每個樣本所含錯誤分類的權重去調整擴增的比例使得模型能更有效地學習到每個錯誤類型，讓每個類型有一定的資料量，以確保系統學習的成功。

根據研究結果顯示，訓練數據與測試數據十分懸殊，發現有過度訓練的問題，我們嘗試透使用Validation來提高數據，透過Tensor Flow的model.fit\_generator將模型每一個圈數的loss值與binary\_accurancy值產生一張圖表，在圖表中看到loss值最低與binary\_accurancy值最高的地方就能得知這是模型訓練的最佳圈數，這樣能改善過度訓練的問題。

本研究讓我們學習到了類神經網路的功能以及如何有效訓練模型，在學習的過程中利用Python語言做為程式開發工具，並且使用Flask架設網站，來讓使用者能透過這個網站來運用本組的研究成果協助使用者分辨出Java程式碼是否存在結構問題，並把本組預測的段落與錯誤分類透過網站讓使用者得知。然而，受限於硬體裝備，本研究尚無法針對更大量較大的程式碼進行實驗。期望在未來能夠更新設備，完成更大量資料的實測，將此部分的精準度再次提高。最後，應用本研究所開發的系統於教學現場，探究其教學助益，也是未來的研究方向。

**參考文獻**

**中文部分**

[1]Lee Meng，淺談神經機器翻譯&用Transformer與TensorFlow2英翻中 檢自 url:

<https://leemeng.tw/neural-machine-translation-with-transformer-and-tensorflow2.html#%E7%B5%B1%E8%A8%88%E6%A9%9F%E5%99%A8%E7%BF%BB%E8%AD%AF%EF%BC%9A%E5%9F%BA%E6%96%BC%E7%9F%AD%E8%AA%9E%E7%9A%84%E7%BF%BB%E8%AD%AF> Retrieved 2020-12-24.

[2]Transformer\_YouTube影片 李宏毅教授 檢自 url:

<https://www.youtube.com/watch?v=ugWDIIOHtPA>

Retrieved 2021-2-26.

[3]施威銘研究室著，tf.Keras技術者們必讀!深度學習攻略手冊，初版，臺北市，旗標出版社，民國一零九年二月。

**英文部分**

[4]C. Kern and J. Esparza, “Automatic Error Correction of Java Programs,” in Technische Universit¨at M¨unchen.

[5]eddy systems, LLC. “eddy autocorrect for Java”. url:[https://eddy.systems/.](https://eddy.systems/)

[6] F. H. Wang, “A Semantics-Preserving Approach to Augmentation of Java Codes for Training Deep Learning Classification Models,” The 25th TANET, 2019

[7]J. Zheng. “Cost-sensitive boosting neural networks for software defect prediction”. Expert Systems with Applications, vol.37, no.6, pp. 4537-4543, June 2010.

[8]K. Choromanski , V. Likhoshersto , D. Dohan , X. Song , A. Gane , T. Sarlos , P. Hawkins , J. Davis , A. Mohiuddin1 , L. Kaiser1 , D. Belanger1 , L. Colwell , A. Weller , “Rethinking Attention with Performers”, Published as a conference paper at ICLR , 2021

[9] Li, J.; He, P.; Zhu, J.; Lyu, M.R. “Software Defect Prediction via Convolutional Neural Network” IEEE International Conference on Software Quality, Reliability and Security, 2017

[10] Mou, L.; Li, G.; Zhang, L.;Wang, T.; Jin, Z. “Convolutional Neural Networks over Tree Structures for Programming Language Processing”. in Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

[11]Polosukhin. I, Kaiser. L,Gomez. N, Jones. L,Uszkoreit. J, Parmar. N, Shazeer. N, Vaswani. A,"Attention Is All You Need".2017.06.12 arXiv:1706.03762

[12] Pradel, M.; Sen, K. “DeepBugs: a learning approach to name-based bug  detection”. in *Proceedings of the ACM on Programming Languages archive* Volume 2 Issue OOPSLA, November 2018 Article no. 147.

[13] Wong, W.E.; Debroy,Y.; Golden, R.; Xu,X; Thuraisingham,B. “Effective  Software Fault Localization Using an RBF Neural Network”. *IEEE  Transactions on Reliability*, vol.61, no.1, March, 2012.

附錄

本次專研實做的工作內容及人員分配如下表所示。

|  |  |  |
| --- | --- | --- |
| 主項目 | 子項目 | 人員分配 |
| 程式碼分類 |  | 全組 |
| 程式撰寫 | 標註程式 | 楊子霆 |
| 陳維嶸 |
| 訓練架構程式 | 楊子霆 |
| 陳維嶸 |
| 張淯雯 |
| 游騰緯 |
| 介面程式 | 楊子霆 |
| 游騰緯 |
| 模型訓練 |  | 全組 |
| 海報、專刊、影片製作 | 海報 | 張淯雯 |
| 專刊 | 張淯雯 |
| 影片 | 楊子霆 |
| 陳維嶸 |
| 游騰緯 |
| 初、總審撰寫 | 初審 | 全組 |
| 總審 | 張淯雯 |

甘特圖:

藉由以下甘特圖如下圖，來說明整體實作的流程及進度。

