智慧化企業整合

Intelligent Integration of Enterprise

長晶製程參數智能化解析——斷線率改善與控制

指導教授：邱銘傳 教授

學生：108034541 周郁淇

一、緒論

(一)研究背景

長晶現多採用柴式長晶法(Czochralski process)，以下簡稱之為 CZ 長晶法並做簡單介紹。CZ 長晶法又稱為直拉法，由波蘭科學家 Jan Czochralski 於 1916 年提出，可簡單分為以下幾個步驟：填料(Stacking Charge)、熔融(Meltdown)、晶頸生長(Neck Growth)、晶冠生長(Crown Growth)、晶身生長(Body Growth)、晶尾生長(Tail Growth)。而長晶製程中，業界對於晶棒的不同生長部位給予別稱，如圖 1.2 所示。

長晶製程中，影響晶棒生長環境的因子眾多，若各項因子沒有控管在一定的範圍內，容易造成晶棒內部的晶體結構出現差排，破壞單晶結構。在單晶結構下，晶棒外觀會出現明顯的結晶稜線，稱作「晶線」，差排隨著晶棒逐漸成晶會延伸至晶棒表面，使晶線消失或不連續，即為俗稱「斷線」之現象。作業員觀察到斷線發生時，會判定晶棒已不符合規格，對晶棒重新進行熔融、生長等步驟，造成極大的成本消耗。而晶棒若發生斷線現象，自重融到拉起需要耗費約 1 天的生產時間，過程中將額外消耗許多設備、人力等資源。

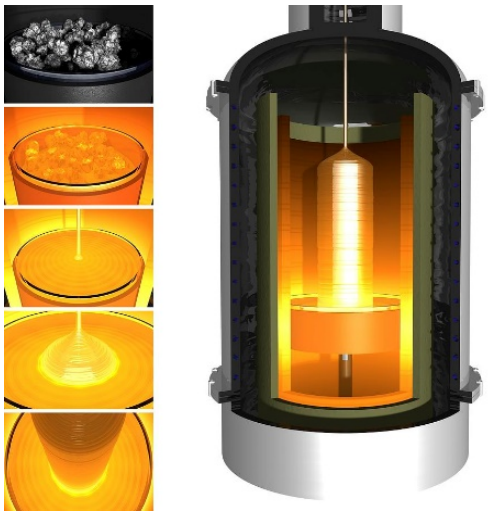

圖 1.1 長晶製程示意圖

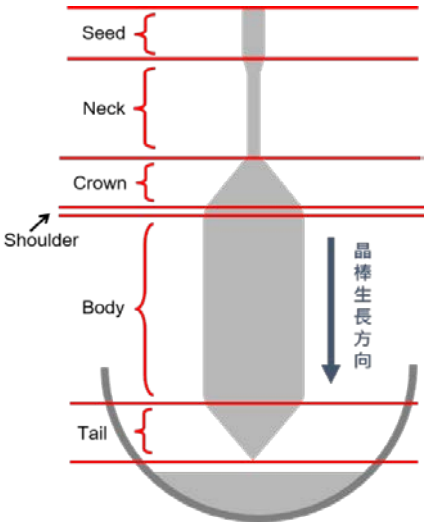

圖 1.2 晶棒部位名稱

(二)問題定義—5W1H

在晶圓廠現行製程中，晶棒是由控制器調整參數自動生長。但若控制器參數設定不佳，多數晶棒皆無法順利自動拉起，經常需要透過作業員手動介入長晶。因此本研究將以製程控制為主，利用機器學習分析過往的生長資料，並判別控制邏輯與斷線之間的關聯，找出造成斷線的原因，解決長晶製程中斷線率高的問題，進而提升產能與良率。

What?	長晶製程斷線率
When?	晶棒生長的過程
Who?	晶圓廠作業員、半導體公司
Where?	晶圓廠長晶爐
Why?	晶棒斷線造成企業成本增加
How?	資料分析、深度學習、機器學習

(三)專題文獻回顧與研究動機

此報告題目為 2018 年工業工程展專題的延伸。2018 年專題 陳任軒 、蔡萱霓 和 周郁淇 透過統計迴歸分析參數數據，將資料做平移、加乘等智能化轉換，提出拉速控制的參數模型；2019 年專題 林品君 、楊朵 與 陳柏言 則以晶棒生長影像資料，進行卷積類神經網路的分析判斷斷線，並進一步改善拉速迴歸模型。而本報告則以機器學習分析參數數據，以時間點差分等邏輯，進行資料前處理，來達到判別斷線的效果。

會選擇這個題目，也是因為當初在做專題時還不甚了解機器學習，因此即使有資料卻仍不會分析，而後來也未用到機器學習來呈現專題成果，是當年專題的一個小遺憾，因此透過本次課程的機會，重新審視長晶製程數據，應用所學之分析方法，期望得到不一樣的結果。這份報告除了感謝 邱銘傳 老師教授我們學習機器學習，也特別要感謝 桑慧敏 老師讓我有這個機會接觸到這個題目與資料，讓我學習到許多平常無法接觸到的實務應用。

二、模型架構

本報告根據晶圓廠提供之長晶生長參數資料，依可控條件篩選關鍵因子，並針對資料時間序列特性進行差分運算，再以類神經網路進行訓練，藉此判斷調整邏輯與斷線之關聯性。

(一)　資料前處理

1. 資料來源

本報告使用之原始資料，係為合法取得竹科某晶圓廠之機密資料，並取得公司方許可作為學術研究使用，且在保密協定的約束下，恕不予公開。而企業提供之實際資料為近兩個月單一長晶爐生產單一規格晶棒、由長晶爐控制機台自動輸出之數值，包括多隻晶棒在同一機台，不同生長時間點之參數數值。

2. 資料混合

根據斷線統計資料，依照圖 1.2 的部位區分，我們統計出 98%的斷線現象皆發生在晶身(見圖 2.1)，因此本報告將以 Body 段的生長參數做為主要研究範圍進行分析。

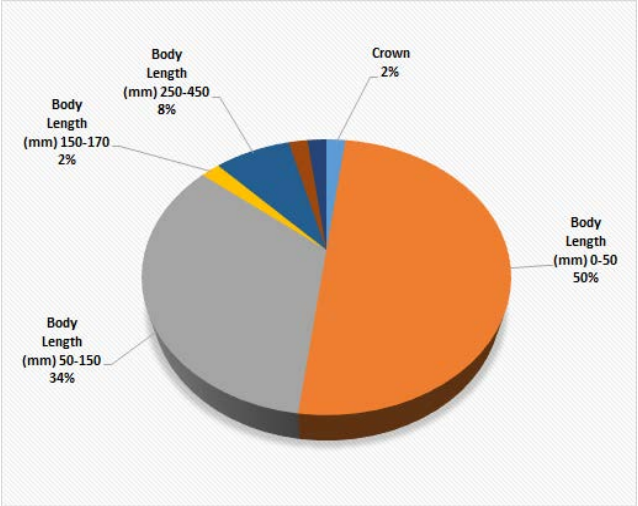

圖 2.1 斷線位置統計圖

因所得晶棒資料參差，有製程中經歷多次斷線或經人工調整的晶棒，我們以一次就長成功及第一次就斷線來做為樣本抽取的標準，最後篩選的晶棒樣本非斷線與斷線比率為 3:1，並以 outcome 欄位做標籤，標示 1 為斷線、0 為非斷線。且因斷線樣本之斷線位置為晶身 55 公分處，我們皆取晶棒樣本生長資料到晶身 55 公分處，混合成訓練資料。

3. 篩選關鍵因子

製程輸出之參數眾多，為達製程及時控制之最終目標，本研究針對可於製程進行當中調控、會直接影響長晶之參數進行討論，並假設除此之外之參數設定皆相同。

原始的長晶製程資料中，共包含下述 19 種製程參數：實際拉速、目標拉速、實際晶棒直徑、目標晶棒直徑、實際加熱器溫度、目標加熱器溫度、實際融湯溫度、實際晶轉、實際堝轉、加熱器實際功率、加熱器實際電壓、加熱器實際電流、坩堝上升速度、實際堝升比、實際氬氣流量、上爐腔爐壓、下爐腔爐壓、節流閥開度、CCD 量測數值。

討論參數於製程當中可控制與否，並根據各參數的性質與生成方式，我們篩選了：目標拉速、目標晶棒直徑、實際晶轉、實際堝升比、實際氬氣流量、上爐腔爐壓及下爐腔爐壓等七個參數來進行訓練探討。

4. 差分

因長晶製程參數多維持在一定值，但若過程發現異常才會進行調整，因此變異不大，而為放大差異，且本報告目的在於以「調整邏輯」，並非以「參數設定」來判斷斷線，因此我們取參數值的「變化量」來進行訓練。而原始資料為每十秒鐘輸出一筆的頻率，我們定義長晶製程時間點為 t，則參數資料處理後為 f(t)=f(t)-f(t-1)。

(二) 模型架構

我們透過簡單類神經網路(NN)來進行機器學習的訓練，模型結構之程式碼如下：

```
from keras.models import Sequential
from keras.layers import Dense

classifier = Sequential()

classifier.add(Dense(input_dim = d, output_dim = 32 , init = 'uniform', activation = 'relu'))

classifier.add(Dense(output_dim = 16, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'relu'))

classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

classifier.fit(train_set, y_train, batch_size=50, nb_epoch=50, validation_split = 0.1)
```

最開始之模型為簡單之三層結構，一個輸入層、一個隱藏層與一個輸出層，optimizer 為 adam，batch size 設定 50，epoch 為 50，準確度為 85%。

三、模型訓練與分析

根據簡單 NN 模型，我們進一步調適 optimizer、activation function、隱藏層、神經元數量、batch size 及 epoch 等設置來找到最佳分析模型。

(一) optimizer 調整

以最開始的 adam 運算，準確率為 85%，我們嘗試使用 Nadam、Adamax，準確率分別是 86%、84%，雖兩者和原本差異不大，我們仍選擇以結果較好的 Nadam 進行後續的訓練。

```
classifier = Sequential()

classifier.add(Dense(input_dim = d, output_dim = 32 , init = 'uniform', activation = 'relu'))

classifier.add(Dense(output_dim = 16, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'relu'))

classifier.compile(optimizer = 'Nadam', loss = 'binary_crossentropy', metrics = ['accuracy'])

classifier.fit(train_set, y_train, batch_size=50, nb_epoch=50, validation_split = 0.1)
```

(二) activation function 調整

```
classifier = Sequential()

classifier.add(Dense(input_dim = d, output_dim = 32 , init = 'uniform', activation = 'relu'))

classifier.add(Dense(output_dim = 16, init = 'uniform', activation = 'sigmoid'))

classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))

classifier.compile(optimizer = 'Nadam', loss = 'binary_crossentropy', metrics = ['accuracy'])

classifier.fit(train_set, y_train, batch_size=50, nb_epoch=50, validation_split = 0.1)
```

因資料 outcome 的標籤型態為 1 跟 0，使用 sigmoid 會較 relu 來的容易收斂，因此我們將隱藏層及輸出層的 activation function 改為 sigmoid，我們得到的準確度可以達到 90%。

(三) 隱藏層、神經元數量

當我們增加模型層數量至四層時，準確率可以提升至 91%，若再增加為五層則沒有明顯增加，因此我們維持總層數為四層(一輸入層、二隱藏層、一輸出層)，並依 2 的次方倍數調適神經元數量，得到神經元數量組合為：[512:256:256:1]時，可以得到較高的準確度 93%。

```
classifier = Sequential()                                                    Test accuracy: 0.930

classifier.add(Dense(input_dim = d, output_dim = 512, init = 'uniform', activation = 'relu'))

classifier.add(Dense(output_dim = 256, init = 'uniform', activation = 'sigmoid'))

classifier.add(Dense(output_dim = 256, init = 'uniform', activation = 'sigmoid'))

classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))

classifier.compile(optimizer = 'Nadam', loss = 'binary_crossentropy', metrics = ['accuracy'])

classifier.fit(train_set, y_train, batch_size=50, nb_epoch=50, validation_split = 0.1)
```

(四) batch size 及 epoch

最後我們調整 batch size 及 epoch，提高兩者的數量也未有顯著的效果，因此最後維持原樣。

四、結論與未來展望

經過訓練之後，模型準確率可以從原本的 85%最後達到 93%，我們可以利用深度學習有效的判斷參數的調整邏輯是否會影響斷線，進而應用在製程當中，作為參數調整的參考。

當然在此報告當中還有許多未完善的部分，實際資料相比複雜許多，訓練的資料樣本不夠多、資料前處理的方法不會是唯一的最佳解，而要將深度學習的結果應用在實務上，更需要多方的實驗與測試，這次報告只用到簡單的類神經模型，也是一時之選，相信若對機器學習有更深入的研究，解決這些實務問題可以更得心應手。