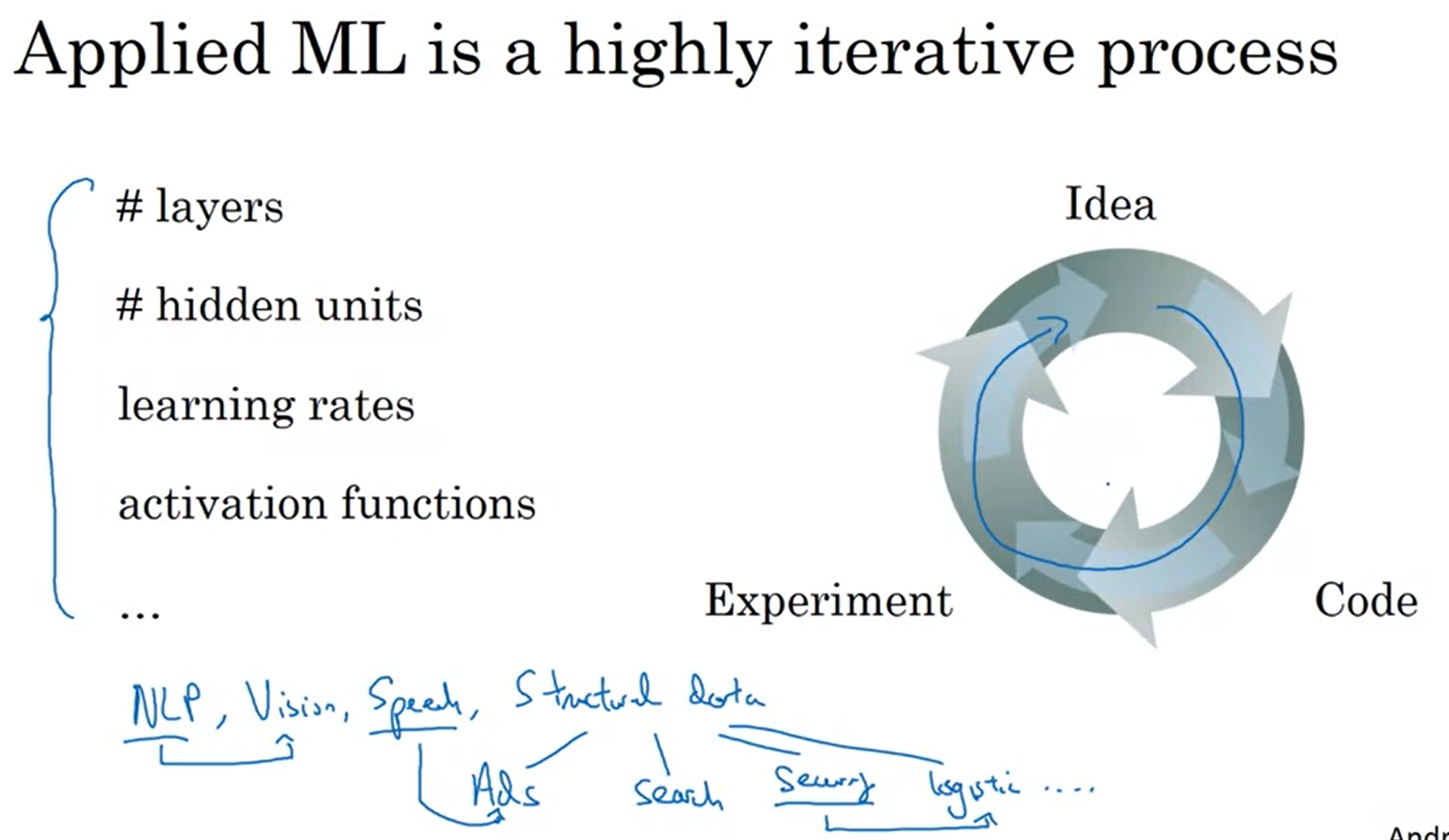
# Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization

## Week 1: Practical Aspects of Deep Learning

### Setting up your Machine Learning Application

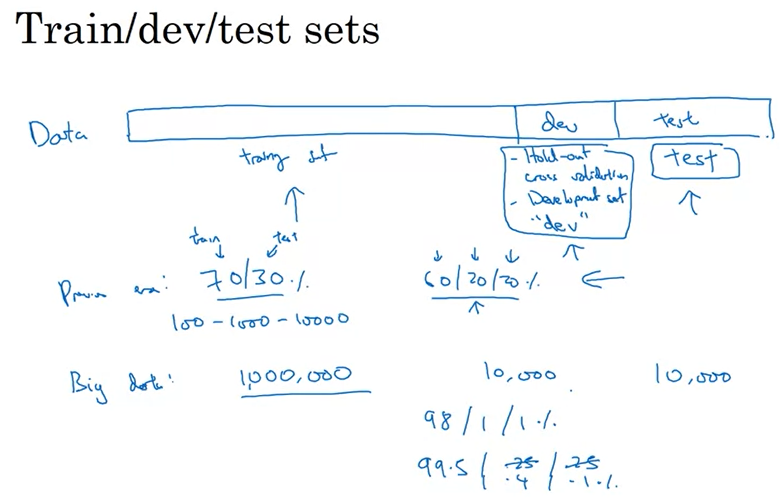
#### Train / Dev / Test sets

歡迎來到這個從實用觀點來看深度學習的課程 也許您已經學到如何建置一個神經網路 在這個禮拜您將學到用實用的觀點如何去讓您的神經網路可行 從像是超參數調整到如何設定您的資料 如何確認您的最佳化演算法跑得最快 使得您的學習演算法的學習時間合理 在第一個星期我們先談到如何設定機器學習問題 然後我們會談到正規化 然後我們會談到一些技巧來確認您的神經網路建置是正確的 這樣，讓我們開始吧

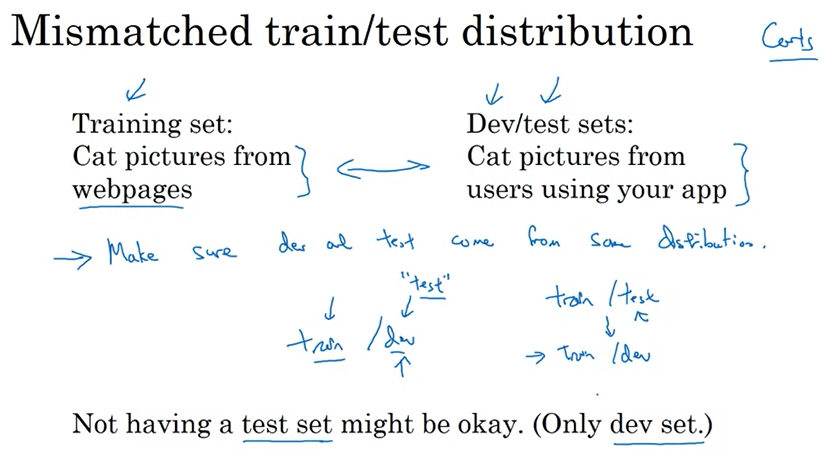


選擇好您如何設定訓練,開發及測試集可能會造成很大的不同在幫助您很快的找到一個高效能的神經網路 當訓練一個神經網路您必須做很多決定 像是您的神經網路要幾層?您每一層要幾個隱藏單元？學習率是多少?您在不同層要用哪些啟動函數？當您開始一個新的應用時 幾乎不可能正確的猜出所有這些正確值 還有其他超參數的選擇在您第一次嘗試時 實際上應用機器學習是高度的循環程序 您通常開始於一個想法 像是您想建立一個神經網路有特定的層數 特定的隱藏單元 也許特定的資料集等等 然後您就開始寫程式 試著去跑它 跑完以後 您得到結果告訴您 這個特定的網路作用得如何 或者說這個特定的設定作用如何 基於成果 您也許改進您的想法 改變您的選擇 也許繼續循環為了試著找到更好的神經網路

在今天, 深度學習已經在很多領域獲得很大的成功 從自然語言處理 到電腦視覺 到語音辨識 到很多應用在結構式資料 而結構性資料包含了 從廣告到網路搜尋 不只是網路搜尋引擎 同時, 舉個例子, 購物網站 任何網站想要送出好的搜尋結果 當您在搜尋欄中鍵入術語時 到電腦安全, 到物流 像是找出司機到取貨點及送貨點 到其他更多的應用 我見到的是有時候 有很多NLP經驗的研究人員 也許試著做電腦視覺 或者也許有很多語音辨識經驗的研究人員 也許進入試圖做廣告有關的東西 或者從保全也許進入做起物流相關 我見到的是 一個領域的直覺或者從一個應用領域通常並不能轉換到另一個應用領域上 最佳的選擇則是依靠在您有的資料量多寡 您有多少輸入特徵 您的電腦設定 是否訓練在GPU或CPU上 如果有，真正的GPU跟CPU的配置等等很多事 對於很多的應用, 我想那是不可能 即使有經驗的深度學習達人 也不可能正確猜出最佳的超參數選擇在第一次時 所以在今天 應用深度學習是很循環的程序您需要循環這個圈圈很多次 期望找出對於您的應用的一個好的選擇 如何快速獲得進展的其中一件事是 您如何用最有效的方式來循環這個圈圈 而設定您的資料集成為訓練,開發及測試集可以讓您更有效率



如果這是您的訓練集 我們畫一個框框 傳統上您也許拿所有的資料 切成一部分是您的訓練集 一部分是您的交叉驗證集有時候也稱為開發集 簡化起見我叫它為dev集 但所有這些名詞大約同義 然後您也許切剩下的部分成為您的測試集 所以程序是您一直訓練演算法在訓練集上 然後使用您的開發集或者說交叉驗證集看看 哪一種模型在您的開發集作用得最好 然後做完足夠長的時間 當您有最後的模型 您想要評估 您可以拿這個您找到的最佳模型 在測試集評估 為了得到您演算法 作用如何的無偏估計 在先前的機器學習時代 通常做法是拿所有您的資料 拆開成也許是70/30% 這方面，人們通常談到70/30訓練測試分拆 當您不用開發集的話, 或者60/20/20%分拆, 就是60%訓練 20%開發, 20%測試 幾年前這是廣泛使用的機器學習最佳典範 如果您也許有100個例子 也許1000個例子 也許10,000個例子 這樣的比率是完美的經驗法則 但在現代大數據時代 舉個例子 您也許有數百萬個例子 趨勢是您的開發跟測試集已經變得小一點的比率了 因為記得, 開發集的目標是 您用不同的演算法來測試看看哪一種演算法作用最好 所以開發集只需要夠大來讓您評估兩個不同的演算法或者十個不同演算法選擇 很快決定哪一個最好 而您不需要整個20%的資料來做這件事 舉個例子, 如果您有百萬個訓練例子, 您也許決定只需要用10,000個例子在開發集已經夠了 來評估哪一個或兩個演算法作用最好 同樣地，您測試集的目的是給您最終的分類器，給您相當自信的評估它作用的如何?再次，如果您有百萬個例子 您也許決定用10,000個例子已經夠了為了要評估一個單一分類器而給您一個好的評估 對於它作用得如何?所以在這個例子 當您有百萬個例子 如果您只需要用10,000給開發集,10,000給測試集 您的比率會是10,000是1%的一百萬 您有98%訓練，1%開發, 1%測試 我也看過一些應用 如果您有甚至比百萬個例子還多, 您或許用99.5%訓練, 0.25%開發, 0.25%測試 或者0.4%開發, 0.1%測試 總結一下，當設定您的機器學習問題時 我通常設定成訓練,開發,跟測試集 如果您有相對小的資料集 傳統的比率也許可用 但如果您有很大的資料集 設定您的開發跟測試集小一點像是比20%小甚至10%也都可行 我們會給予特定的準則在開發集及測試集 在晚一點這個學程中



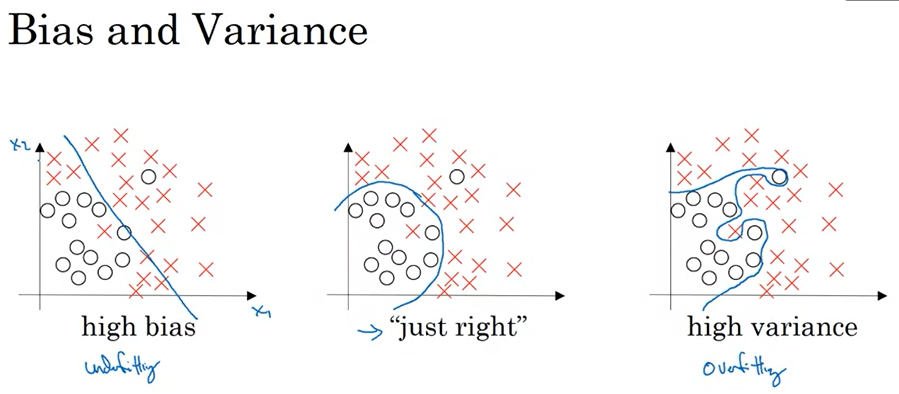
另一個趨勢我們在現代深度學習時代是越來越多人們訓練在不同的訓練及測試分佈 假釋您設置一個應用 讓使用者上傳很多的照片 您的目標是找出貓的照片來給您的使用者 也許您所有個使用者都是愛貓者 也許您的訓練集是從網路下載的貓的照片 您的開發及測試集也許包括了從我們app的使用者來的貓照片 也許您的訓練集很多 來自網路爬下來的 但是開發跟測試集是從使用者上傳來的 結果是很多網頁有很高解析度專業很棒的貓的照片 但或許您的使用者上傳模糊 低解析度影像從手機照相或是比較休閒的狀況下 所以這兩種分佈的資料是不同的 按照經驗準則是鼓勵您在這情況下 確定您的開發及測試集來自相同的分佈

我們將會談到這個準則，但因為您會使用開發集來評估很多不同的模型 努力試著增進在開發集的表現 如果您的開發集跟測試集來自同樣的分佈會很好 但因為深度學習演算法對訓練資料極度飢渴 一種趨勢我看到的是您或許使用種種創意的策略 像是爬網路 為了獲得比原先更大的訓練集 即使部分的成本會是您的訓練集 資料也許不是來自於您開發及測試集同樣的分佈 但您會發現只要您遵循這個準則 您機器學習演算法的進度會更快 我會給更多的解釋對於這種特別的準則 在往後的學程中

最後, 也許沒有測試集是可行的 記得測試集的目的是給您一種無偏估計在您最終網路的表現您所選擇的網路 但如果您不需要無偏估計 那也許不需要 測試集是可行的 如果您只有開發集 而沒有測試集您會做的是 您在訓練集上訓練然後您試著不同的模型架構在開發集評估它們 然後運用它們用循環的方式來試著得到好的模型 因為您用您的開發集來配模型 這不再能給您模型表現的無偏估計 但如果您不需要這個 也許完全可行 在機器學習世界裡 當您只有訓練跟開發集沒有另外的測試集 大部分的人會稱這個為訓練集而稱開發集為測試集 但其實他們是用測試集來作為交叉驗證集用 也許不是好的 專業術語使用方式 因為他們會對測試集過適 所以當一個團隊告訴您他們只有訓練及測試集 我會謹慎地想他們真的有訓練開發集？因為他們會對測試集過適 文化上也許要改變一些團隊的術語很難 要他們稱之為訓練開發集 而不是訓練測試集 即使我想稱之為訓練跟開發集是比較正確的術語 而這實際上是可行的 如果您完全不需要您演算法表現的無偏估計 所以設定好訓練,開發,測試集 會讓您更快循環 也會讓您更有效地測量變異及偏差對於您的演算法, 所以您可以有效地選擇改進您演算法的方式 我們在下一段影片來開始談這個

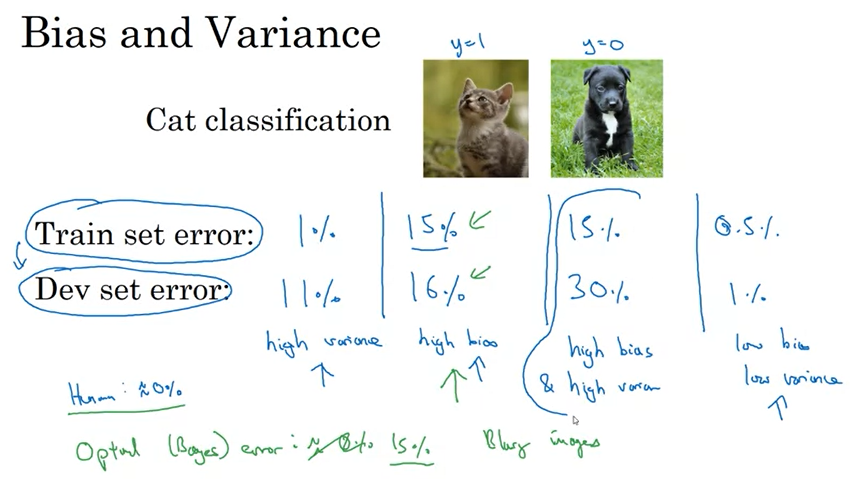
#### Bias / Variance

我注意到幾乎所有很棒的機器學習 達人往往是很精於理解偏差跟變異 偏差跟變異是一種很容易學習但很難駕馭的概念 即使您想您已經看了偏差及變異的基本概念 還會有一些新的您意想不到的概念出現 在深度學習時代 另一種趨勢比較 少討論到的是稱為偏差變異權衡 您或許聽過偏差變異權衡 但在深度學習時代比較少權衡 所以我們還是解決偏差 還是解決變異 但我們比較少有關偏差變異權衡 我們來看這是什麼意思



1. 我們看一個資料集像這樣 如果您用一條直線來配合這些資料 也許用羅吉斯迴歸分析來配合它 這不是很配這些資料 所以這是一種高偏差的類別 我們說這個模型低適這些資料
2. 相反地 如果您用極度複雜的分類器 也許深度神經網路 或者神經網路用很多的隱藏單元 也許您可以完美的配合這些資料 但這看起來也不像是好的配適 這是一種高變異的分類器而這樣會過適這些資料
3. 而也許有一些分類器介於其中 有中度的複雜性 也許是一條曲線像這樣 看起來更合理的配適這些資料 我們稱它為剛剛好，它是介於中間

所以在 2D 的例子像這樣 只有兩個特徵 X-1 跟 X-2 您可以畫這些資料來視覺化它們的偏差及變異 在高維度的問題中 您不能畫這些資料來視覺化這些邊界 取而代之, 有一些不同的量測 我們用來試著理解偏差及變異 繼續我們的貓圖片辨識例子

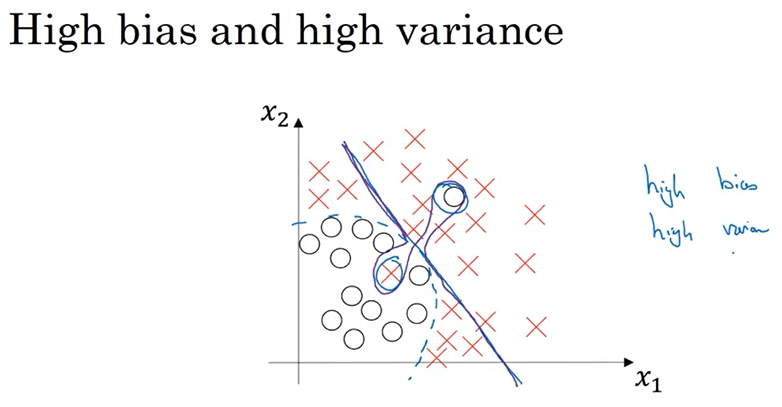


|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 過適 | 低適 |  |  |
| Train set error | 1% | 15% | 15% | 0.5% |
| Dev set error | 11% | 16% | 30% | 1% |
| (誤差) | High Variance | High Bias | High Bias &  High Variance | Low Bias &  Low Variance |

這是一個正面的例子，這是一個負面的例子 兩個重要的數字來理解偏差跟變異會是 訓練集誤差及開發集誤差 為了便於討論 假設您能夠辨識貓的相片 這是一般人都可以完全做到的, 對吧?

* 假設說您的訓練集誤差是1%, 而您的開發集誤差 為了便於討論 假設說是11% 在這個例子中 您在訓練集做得很好 但您在開發集就相對做得比較差 看起來您也許過適於訓練集 或許您沒能一般化得很好到整個交叉驗證集, 也就是開發集 所以如果您有這樣的例子 我們會說這個有高的變異 所以看著訓練集誤差跟開發集誤差 您能夠診斷您的演算法有高的變異
* 假設您測量您的訓練集誤差跟開發集誤差 而得到不同的結果 假設您的訓練集誤差是 15% 我將訓練集誤差寫在上面的列上 而您的開發集誤差是 16% 在這個狀況下, 假設人類大致能達到 0% 誤差 人類可以看看這些照片，直接判斷是不是貓 然後看到這個演算法甚至在 訓練集做得不是很好 所以如果甚至在訓練集 對資料配合得不是很好 那這是低適於這些資料 所以這個演算法有高的偏差 但相反的, 這個實際上 在開發集上一般化得很好 它在開發集的表現只有 比在訓練集差一個百分點 所以這個演算法有高偏差的問題 因為它甚至不配適在訓練集上 這是相當類似於我們 前一張投影片的最左邊
* 現在, 有另一個例子 假設您有 15% 的訓練集誤差 所以這是很高的偏差 但當您衡量開發集時, 情況更糟 也許是 30% 在這種情況下，我會診斷該演算法具有高的偏差 因為在訓練集做得不是很好, 而且有高的變異 所以這真的是兩個世界裡最差的狀況
* 最後一個例子 如果您有 0.5% 訓練集誤差 跟 1% 開發集誤差 也許我們的使用者很高興 您有一個貓咪分類器只有 1% 誤差 我們同時有低的偏差及低的變異

一個細微之處，我只簡要地提及 我們會留到以後的影片 再來詳細討論 這種分析是基於假設 人類幾乎可以達到 0% 誤差 或更一般來講, 最佳誤差 有時候也稱為基本誤差 所以最佳誤差將近 0% 我在這段影片 不想深入探討這個 但實際上最佳誤差或是 基本誤差高很多 假設說是 15%, 那當您看這個分類器 15% 實際上是很合理的在訓練集上 就不會當它是高的偏差同時變異也很低 所以如何分析偏差跟變異 當沒有分類器可以做得很好時， 舉個例子 如果您有很模糊的照片 即使人類或者 沒有一個系統可以做得很好時 也許基本誤差高很多 這時一些細節對於 如何做分析也會改變 但除此細節之外 重點是看著 您的訓練集誤差會讓您感覺到 資料配適做得如何 至少在訓練集上 所以可以告訴您是否有偏差的問題 然後看著您的誤差有多高 當您從訓練集到開發集時 這應該會給您一些感覺 有關變異問題有多糟 所以您從訓練集一般化到 開發集時做得好不好 給您一些有關 變異的感覺 所有這些都基於一個假設 就是基本誤差是相當小的而且您的訓練集跟您的開發集 都是從同一分佈抽樣出來的 如果違反這些假設 您可以有一些更複雜的分析 我們會在以後影片中談到 在前面的投影片中 您看到高偏差, 高變異的樣子 我猜您對於一個 好的分類器有一些感覺 同時高偏差跟高變異的樣子如何?

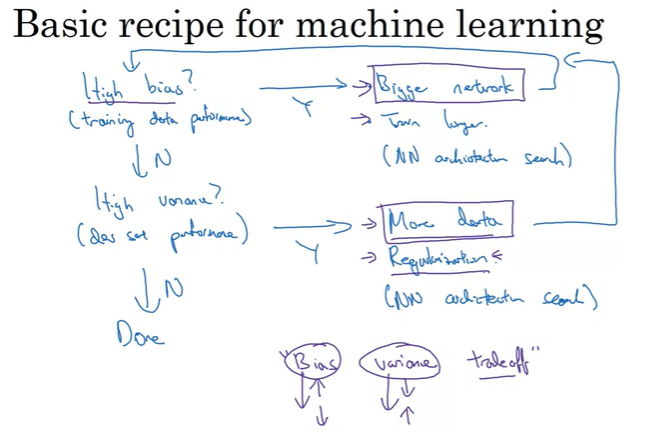


這是一種兩個世界裡 最糟的狀況 您記得, 我們說一個分類器像這樣 那您的分類器有高的偏差 因為它低適了這些資料 這會是一種將近線性的分類器 所以低適了這些資料 我們用紫色來畫 如果您的分類器做很奇怪的事 那這也其實過適了這些資料 所以我用紫色畫的 有高偏差同時也有高變異 有高的偏差因為 將近是一個線性分類器 並不是很配適這些資料 像是這個二次曲線的形狀 但有著相當彈性在中間 它為了包含這個例子 跟這個例子, 過適於這兩個例子 所以這個分類器有高的偏差 因為它幾乎是線性的 您需要的也許是 曲線函數或者二次函數 而那也有高的變異 因為它太彈性的 去配適這兩個誤標的例子 或者在中間的異常值 如果這看起來很做作 這個例子是有點做作 在二度空間 但在高維度的輸入時 您實際上會得到 高偏差在一些區域裡 跟高變異在一些區域裡 所以這樣的分類器是可能的 在高維度輸入時 似乎會比較不做作

總結一下, 您看到了 如何看待您演算法的誤差在訓練集上跟演算法誤差在開發集上 診斷是否有高偏差或高變異的問題 也許同時有，或者同時沒有 依據是否您的演算法 承受著偏差或者變異 實際上有不同的方式 來試著解決 在下一段影片， 我們將談到 所謂的機器學習的基本配方 讓您有系統化地改進您的演算法 依據是否有高偏差或者高變異問題 讓我們進入下一段影片

#### Basic Recipe for Machine Learning

在先前的影片中, 您看到訓練誤差及開發誤差如何幫助您 診斷是否您的演算法有偏差或是變異 或是兩者皆有的問題 實際上這些資訊讓您比較 有系統化的使用他們稱之為基本機器學習的配方(Basic recipe)，讓你更系統化 去提高你的演算法的性能。



我們來看看 當訓練神經網路時 這是我會使用的基本配方 經過訓練完一個初期的模型後 我首先會問 您的演算法有高的偏差嗎? 試著評估是否有高的偏差 您真的應該看看 在訓練集或者說訓練資料的表現 如果它真的有高的偏差 甚至不是那麼配適訓練集 您可以試著選一個網路 加上更多的隱藏層或者隱藏單元 或者訓練長一點的時間 也許梯度下降跑長一點 或者更進階的最佳化演算法 我們將在這堂課的後面談到 或者您也可以試 這是一種， 也許可行, 也許不可行的方法 但我們將來會看到很多不同的神經網路架構 也許您可以找到一種神經網路架構適合這個問題 將這個放上括號, 因為一件事 您就是要去試 也許您可以找到, 也許找不到 但是用大一點的網路幾乎都會有幫助 訓練長一點的時間並不一定有幫助 但也不會造成傷害 所以當訓練一個學習演算法時 我會試這些事情 直到至少去掉偏差問題 回頭試試這些東西 直到我可以配適 至少很配適訓練集 通常如果您有足夠大的網路 您應該可以能夠很配適這些訓練資料 只要這個問題可解的話, 對吧? 如果影像很模糊 也許沒辦法來配適它 但如果人類可以做得到 如果您覺得基本誤差不大 那用足夠大的網路訓練的話 您應該可以做到 希望至少在訓練集可以做得很好 至少配適或者過適訓練集 一旦您減低偏差到可以接受的數字後 就要問您有變異的問題嗎? 要做評估的話, 我會看開發集的表現 您可以從很棒的訓練集表現推廣 到很棒的開發集表現嗎？ 如果您有很高的變異 最佳解決變異問題的方法是 獲得更多資料 如果您可以得到的話 就會有幫助 但有時候您無法獲得更多資料 或者您可以試試正規化 我們在下一段影片會談到 試著去除過適問題 然後再說一次 有時候您只能試試 但如果您可以找到更合適的神經網路架構 有時候您可以減少您的變異問題 就像減低偏差問題一樣 但如何做? 要完全系統化做這個很難 但當我試這些時 我會一直回頭 直到希望找到可以同時低偏差跟低變異 那您就做到了 要注意幾個點

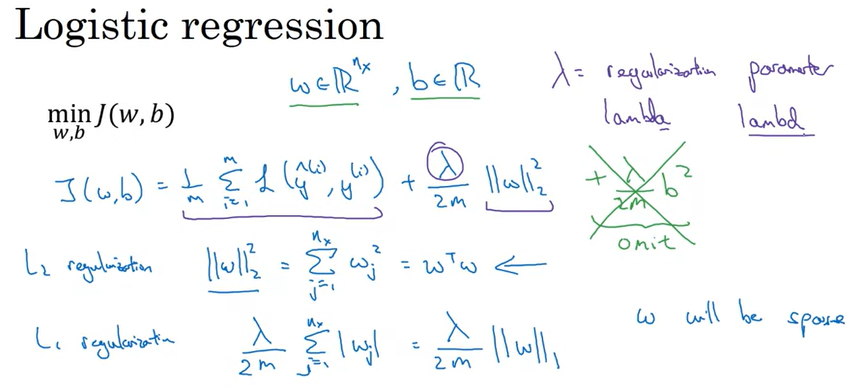
1. 首先, 根據是否您有高偏差或者高變異 您可以試的東西 可能非常不同 我通常會用訓練開發集試著 診斷是否您有偏差或者變異問題 然後選擇適當的子集來試 舉個例子, 如果您有高偏差問題 獲取更多的資料其實並沒有幫助 或者說並不是最有效的方法 所以確認是偏差問題 或者是變異問題 或者兩者皆是 可以幫助您專注於最有用的事情來試
2. 第二, 在早期機器學習時代 曾經有很多的討論 有關於稱為偏差變異權衡 而原因是 對很多事情你可以試試 您可以增加偏差跟減低變異 或者減低偏差增加變異

但回到深度學習前時代 我們沒有很多工具 我們並沒有很多工具來 只是減低偏差或者只是減低變異 而不影響另一個 但在現代深度學習, 大數據時代 只要您可以一直訓練大一點的網路 只要您可以獲得更多資料 並不總是可以辦到 但如果可以做到 那用大ㄧ點的網路幾乎總是可以 只減低您的偏差 並不一定傷害您的變異 只要您適當的正規化 而獲得更多的資料通常總是 減低您的變異 而不會傷害您的偏差 真正發生的是 使用這兩個步驟 能夠去訓練, 選擇一個網路 或者獲取更多資料 我們現在有工具來減低偏差 且只減低偏差 或者減低變異且只減低變異 而不會傷害其他的東西 我想這是一個很大的原因 深度學習對於監督式學習如此有用 當然或多或少這種權衡 您 必須仔細去平衡偏差跟變異 但有時候您就是有比較多的選項 來減低偏差 或者減低變異 而不一定增加另外一項 實際上只要您有好的正規化網路 我們會在下一段影片談到正規化 訓練大一點的網路 幾乎永遠不會有壞事 而主要的成本來訓練太大的神經網路 只是運算時間 只要您做了正規化 我希望這給您一點基本架構的感覺 如何 組織您的機器學習問題 去診斷偏差及變異 然後試著選擇正確的動作 來讓您的問題有所進展 我在這段影片一直提到正規化這件事 是非常有用的技巧來減低變異 這裡有一點點偏差變異權衡 當您使用正規化 它也許會增加一點點偏差 雖然通常不會太大 如果您有足夠大的網路 但讓我們在下一段影片深入探討這些細節 所以您可以 比較了解如何應用正規化到您的神經網路

### Regularizing your Neural Network

#### Regularization

如果您懷疑您的神經網路 過適於您的資料 也就是您有高變異問題 您第一件事情應該 要試著做的也許是正則化 另一種方式來解決高變異是得到更多的資料 這也是相當可靠的方式 但您不可能永遠獲得 更多的資料, 或者 獲得更多的資料可能很貴 但加入正則化通常會 幫助您避免過適問題 或者減低您網路的錯誤 所以讓我們來看正則化如何作用

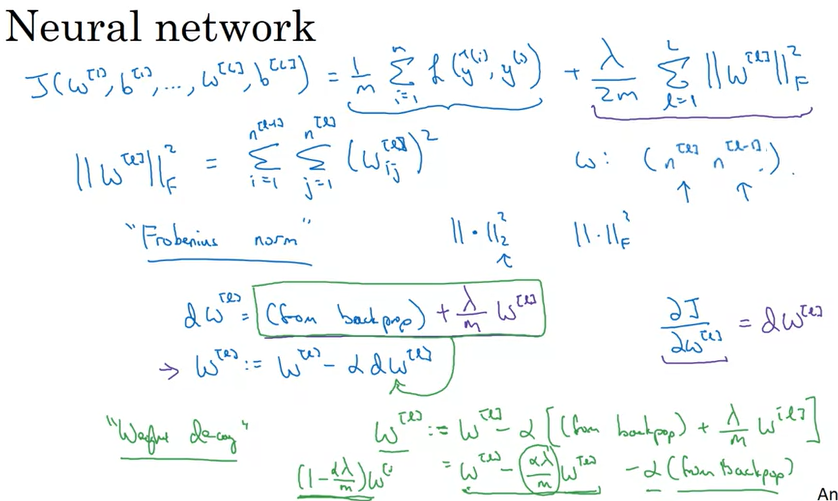


讓我們用羅吉斯迴歸分析來發展這個觀念 記得在羅吉斯迴歸分析中 您試著最小化成本函數J定義成這個成本函數 總和您的訓練例子的個別預估值損失在不同的例子上 您記得w跟b在羅吉斯迴歸分析中是參數 所以w是nx維度參數 向量b是一個實數 要加上正則化到羅吉斯迴歸分析, 您要做的是這件事, lambda, 稱為正則化參數 我們等一下會談到lambda/2m乘上w的範數平方 這裡w範數平方等於從j等於1到nx總和wj平方 或者這也可以寫成wTw, 這只是參數w向量的歐氏範數平方 而這稱為L2正則化

∥v∥1

因為這裡您使用的是歐氏範數, 也是稱為L2範數 對於參數向量w 為什麼您只有正則化參數w 為什麼我們不把b也加進去？實作時, 您可以這樣做 但我通常省略它 因為如果您看您的參數w 通常是很高維度參數向量 特別是高變異問題 也許w就是有很多參數 所以您無法配適所有的參數 而b只是一個數 幾乎所有的參數在w而不在b 而如果您加上最後一項 實作上它不會造成多大不同 因為b只是很大數目的參數之一 實作上我通常不去包含它 但如果您要加的話也可以 所以L2正則化是最常見的正則化 您也許也聽過人們談到L1正則化 也就是當您加的時候 與其加這個L2範數 您取而代之用這個項lambda/m總和這個 這也是稱為L1範數的參數向量w 所以這裡有小小下標1 對吧? 我猜不管您放m或者2m在分母 都只是一個比例常數 如果您用L1正則化 那w會變得比較稀疏 這個意思是w向量會有很多的0在裡面 有些人說這樣可以壓縮這個模型，因為很多參數為0時 您可以用比較少的記憶體來儲存這個模型 雖然我發現，實作時L1正規化讓您的模型稀疏 只會幫助一點點 所以我並不常用它 至少不會為了壓縮您的模型用它 而當人們訓練網路時 L2正則化就是用的多很多 抱歉, 更正一些記號 最後一點細節 這裡的lambda是稱為正則化參數

通常您使用開發集來設定這個 使用交叉驗證集 當您試著不同的值 看哪一個作用最好 權衡了在訓練集作用得很好 以及設定了您參數的L2範數較小 這樣會避免過適 所以lambda是另一個超參數 您必須調整 順便說一下 對於程式作業lambda是一個保留關鍵字在Python程式語言 所以在程式練習中 我們用lambd少了一個a, 為了不跟Python保留關鍵字衝突 所以我們用lambd來代表lambda正則化參數 這是您如何使用L2正則化在羅吉斯迴歸分析上



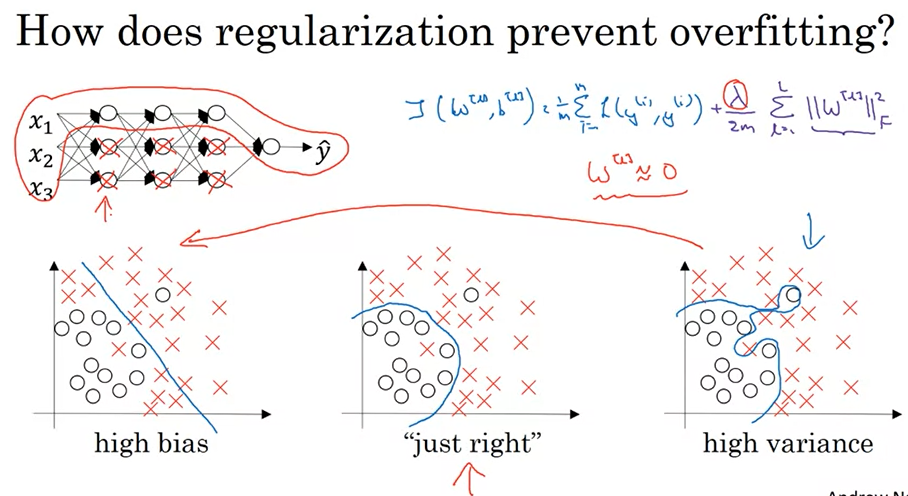
如何用在神經網路上？在神經網路上, 您有一個成本函數是一個函數 用所有的參數, w[1] b[1] 直到w[L], b[L] 而大寫L是您神經網路的層數 所以成本函數是這個 對於損失做總和 您m個訓練例子的總和 而要加入正規化 您加入lambda除以2m of 總和對於所有您個參數W 您的參數矩陣w我們也稱為範數平方 而矩陣的範數也就是範數平方定義為i總和j總和 所有矩陣的元素平方 如果您要這個總和的索引 這一個是總和從i=1到n[l-1] 總和從j=1到n[l] 因為w是一個n[l-1]\*n[l]維度向量 而這些數字是l-1層跟l層的單元數目 這個矩陣範數 實際上也稱為弗比尼斯範數在矩陣上記一個F在下標 為了神秘的線性代數 技術上的原因 這不稱為矩陣L2範數 而是稱為矩陣弗比尼斯範數 我知道聽起來稱它為矩陣L2範數比較自然 但為了神秘的理由 您不需要知道 按照慣例，這就稱為弗比尼斯範數 它只是對於矩陣元素平方的總和 所以您如何用它來建置梯度下降 之前, 我們會用反向傳播來完成dw 而反向傳播會給我們偏導數 of J 相對於w 或者應該說w對於任何[l] 然後您更新w[l]=w[l]-α\*dw[l] 所以這是加上額外的正則化項目之前的目標 現在我們加入這個正則化項目到這個目標 您要做的是拿dw + lambda/m\*w[l] 而然後您只要計算這個更新值 像之前一樣 實際上有了這個新的dw[l]定義 這個新的dw[l]仍然是正確的導數定義 對於您的成本函數 相對於您的參數 即使您現在在後面加入這個額外的正則化項目

而為了這個原因 L2正規化有時候也稱為權重衰減 如果我拿這個 dw[l] 的定義 然後代入這個公式 然後您會看到這個更新是 w[l] = w[l] -α\* (這個從反向傳播來的東西) + lambda/m \* w[l] 丟一個減號在這裡 所以這等於 w[l] - αλ/ m \* w[l] - α\* (您從反向傳播來的東西) 而這一項表示 不管w[l]矩陣為何 您將會讓它小一點點, 對吧? 這個實際上是如果您拿這個矩陣w 你將它乘以 1-αλ/m 您其實是拿這個矩陣w -α lambda /m 乘上這個 像是您用這個數字乘上矩陣w 這會是比1小一點點 所以這是為什麼 L2 範數正則化 也稱為權重衰減 因為它像一般 梯度下降, 而您更新 w 用減去 alpha 乘上 從反向傳播得到的一般梯度 但現在您也將 w 乘上這個東西 這個會比1小一點 所以另外一個名稱給L2正則化是權重衰減 我將不會使用這個名稱 但直觀上而言 為什麼稱為權重衰減是 因為第一個項目等於這個 所以您將這個權重矩陣 乘上一個數子比1稍小一點 所以這是您如何建置L2正則化在神經網路

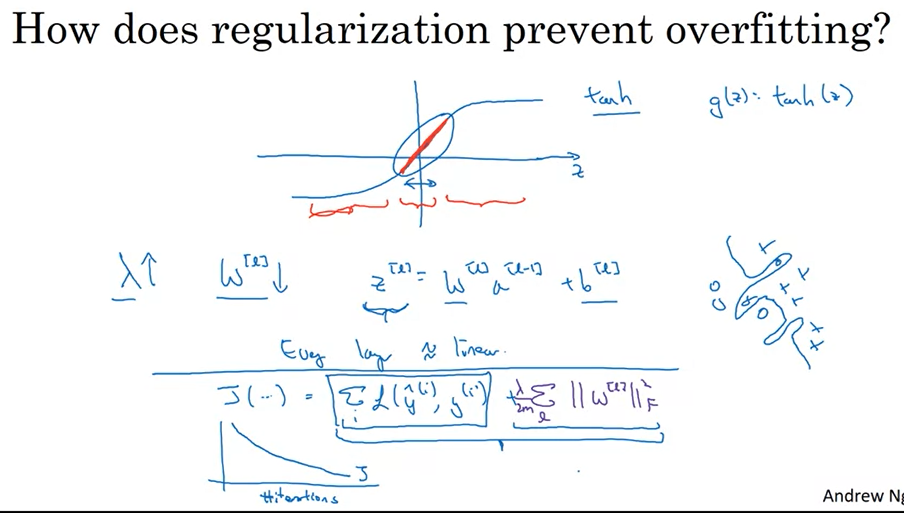
現在一個問題是 嗨, Andrew為什麼正則化能夠避免過適 讓我們來看下一段影片 來得到一些直觀 正則化如何避免過適

#### Why Regularization Reduces Overfitting?

為什麼正則化幫忙解決過適問題? 為什麼它有助於減低變異問題 ？ 讓我們用一些例子來 直觀上看它如何作用



記得高偏差, 高變異 我只是從我們之前影片中 的圖形重畫一遍像這樣 現在讓我們來看大一點的深度神經網路 我知道我不曾畫過 太大或太深的網路 但讓我們看一些過適的神經網路 您有一些成本函數像是 J of w b 等於損失的總和 我們在正則化作的是增加 這個額外的項目 去懲罰權重矩陣如果太大了 所以這是弗比尼斯範數 所以為什麼用 L2範數或 弗比尼斯範數縮減參數 會比較不過適呢？ 一點直觀是如果您 設想正則化 lambda 相當相當大 它們會真的激勵到 設權重矩陣 w 趨近於 0 所以一點直觀是或許設權重趨近於 0 對於很多的隱藏單元基本上 零化是去除掉這些隱藏單元的影響 如果是這樣 這會簡化神經網路 變成小一點的神經網路 實際上, 這幾乎是羅吉斯迴歸分析 疊成幾層深 所以這會將您從 這樣過適情況趨近於左邊到高偏差情況 但希望會有一個中間值的 lambda 結果會是到在中間 "剛好正確" 的情況 但直觀上設想 lambda 很大會使得 w 趨近於 0 實際上這不會真的發生 我們可以想成零化或至少減低 很多隱藏單元的影響 所以您最終會 也許像是簡單一點的網路 它們會越來越趨近於 只是使用羅吉斯迴歸分析 這種完全零化一大堆 隱藏單元的直觀並不正確 實際上發生的是它們 還是會使用所有隱藏單元 只是它們每一個都會 變得影響比較小 但您最終會得到一個 比較簡單的網路 就像您有小一點的網路 而這樣比較不會過適 不確定這樣的直觀是否幫助您理解 當您建置正則化 在程式練習中 您實際上自己會看到 一些變異減低的結果

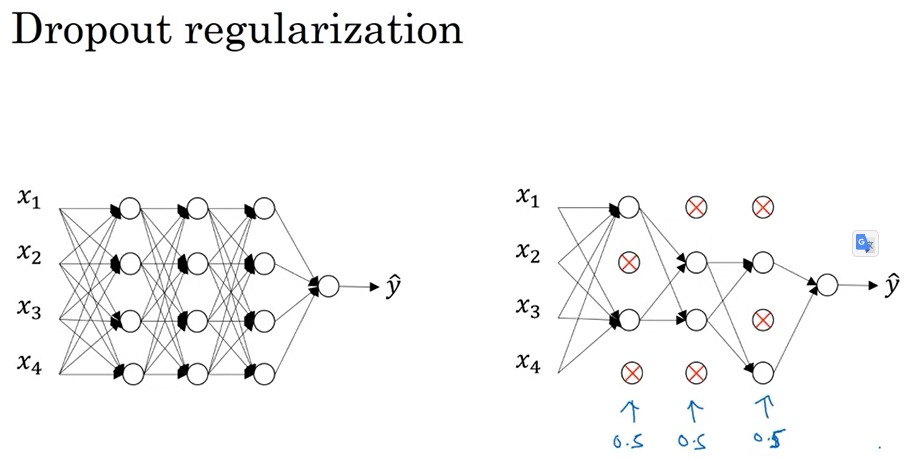


這是另一個嘗試用直觀來 看為什麼正則化 幫助避免過適 這裡, 我先假設我們使用 tanh 啟動函數 像這樣 這是 g(z)=tanh(z) 如果是這樣 注意到只要z很小 所以當z只取一個小範圍的參數 也許在這邊 那您只是使用tanh函數的線性區域 只要z允許到大一點或小一點的值像這樣 啟動函數會開始比較不線性 直觀上您可以學到的是 如果lambda正則化參數變大 您會讓您的參數變小 因為它們會被懲罰如果成本函數變大 而如果w變小因為z等於w 然後技術上是要加上 b 如果w變小 那z也會變小 特別是, 如果z最終以相對較小的值只在這一小段範圍 那g of z會大約是線性 就好像每一層都大約是線性 就像是線性迴歸分析 而我們在第一課程看過 如果每一層都是線性那您整個網路就只是線性網路 即使是很深的網路 在深度網路使用線性啟動函數 最後它們只能夠計算線性函數 所以不能夠來配適 很複雜的決策 很非線性的決策邊界 允許它很 過適於資料集 像是我們看到的 過適高變異情況 在前一張投影片

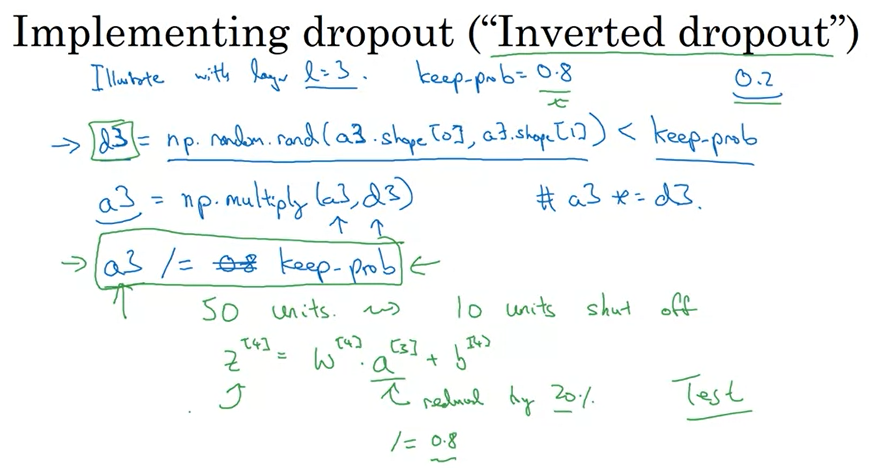
總結一下 如果正則化變得很大 參數w很小 所以z相對變小 先忽略對b的影響 所以z相對變小 真的，我應該說它只取一小段範圍的值 所以啟動函數如果是tanh會將近線性 所以您整個神經網路計算的 不會離一個大的線性函數很遠 也就是相當簡單函數而不會是很複雜非線性函數 那種比較會過適 再提醒一次，當您自己進入程式練習的正則化 您會自己看到這些影響 在結束我們的正則化討論之前 我只是要給您一個建置的提示 也就是當建置正則化時 我們拿我們的成本函數 J 的定義 我們修改它 增加這個額外的項目 如果這個權重太大懲罰它 所以如果您建置梯度下降 一個梯度下降除錯的步驟是 畫成本函數 J 是一個函數 對於梯度下降的迭代數目 而您要看到 成本函數 J 單調下降 經過每一次的梯度下降 如果您建置正則化 請記得 J 現在有新的定義 如果您畫舊的 J 的定義 只有第一項 那您可能看不到單調下降 所以在梯度下降除錯時 請確認您畫的是 新的 J 的定義 包含了第二項 否則您也許看不到 J 單調下降 在每一個單一評估後 所以這是 L2 正則化 實際上 是一種正則化技巧 我用在大部分的深度學習模型中 在深度學習還有另外一種 有時候用來正則化的技巧 稱為 dropout 正則化 讓我們在下一段影片看看它

#### ★Dropout Regularization

除了 L2 正則化 有一種有力的正則化技巧稱為 "dropout" 我們來看它如何作用

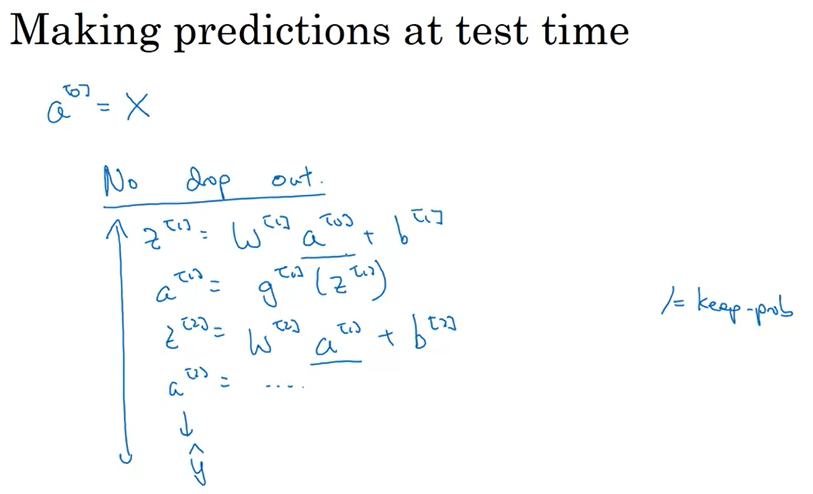


假設您訓練一個神經網路 像左邊這個是過適的 這是 dropout 的做法 讓我複製一下神經網路 使用 dropout, 我們要做的是通過網路每一層 設定一些機率來消除神經網路的節點 假設對於每一層 我們將對每一個節點 擲一枚硬幣，有 0.5 的機會 保留這個節點, 0.5 機會刪除每一個節點 經過擲硬幣後 我們會決定消除這些節點 我們做的是實際 從這些節點移除所有輸入及輸出 所以我們最終會得到一個小很多 真的裁減很多的網路 然後您做反向傳播訓練 使用一個例子在這裁減很多的網路上 而在不同的例子上 您會再次擲硬幣來用 不同集合的節點然後 丟棄或者去除不同集合的節點 所以對於每一個訓練例子 您會訓練它使用這些縮減網路之一 也許這似乎是一個有點瘋狂的技術 它只是四處隨機剔除節點 但這實際上可行 但您可以想像因為您訓練每一個例子 在比較小的網路上 或者也許給一些感覺 為什麼您最終能夠正則化這個網路 因為這些比較小的網路已經通過訓練了 讓我們來看如何建置 dropout



有一些方式來建置dropout 我將告訴您最常用的這種技巧稱為 翻轉dropout 為了完整描述起見 假設我們用l=3層來描述 所以在程式裡我將寫一堆3 這只是說明如何去表示dropout在單一層裡 我們要做的是設定一個向量d 而d3將是dropout向量在第三層 這就是3的意思, np.random.rand 而這將跟a3同樣的shape 而我來看看這個是否小於某個數字 我將稱它為keep\_prob 所以, keep\_prob是一個數字 之前是用0.5 也許我現在用0.8 在這個例子上 而這是對於一個隱藏單元要不要保留的機率 所以keep\_prob=0.8意思是有0.2的機會去除掉任一個隱藏單元 它的作用是產生一個隨機矩陣 如果您向量化的話也管用 所以d3是一個矩陣 因此，每個示例都有一個隱藏單元，相應的d3的機率是0.8，而零的機率是20％。 因此，此隨機數小於0.8，則有0.8的可能性為1或為真，而20％或0.2的可能性為假，為0。然後您要做的是 拿您第三層的啟動值 讓我稱之為a3 在下面這個例子 所以 a3 是您計算的啟動值 您可以設a3=舊的a3 \* 這是ㄧ個逐元素的乘積 或者您可以也寫成 a3 \*= d3 這樣做是對於d3每個等於0的元素 每個元素有20%機會為0 只是乘法運算最終將d3的對應元素歸零 如果您在python做這個 技術上d3會是一個布林值矩陣 其值會是真或偽 而不是1或0 但用乘法運算一樣可行, 它會將真跟偽的值換成1跟0 如果您自己在python試一試, 您會了解

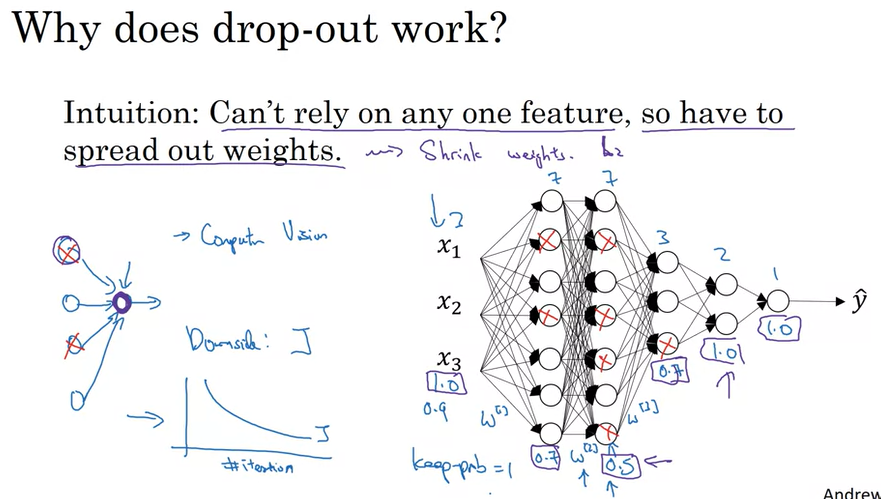
最後, 我們將a3按比例放大, 利用除以0.8, 實際上是除以我們的keep\_prob參數 讓我來解釋 最後這個步驟在做什麼 假設為了方便解釋起見 您有50個單元或者50個神經元在第三個隱藏層 所以也許a3是一個50乘1維度或者如果您向量化, 或許是50乘m維度 如果您有80%機會保留它們，20%機會去掉它們 這意思是平均來講 您最終會關閉10個單元 或者10個單元歸零 現在如果您看z[4]的值 z[4]將等於w[4]\*a[3]+b[4] 所以,期望值這個會減少20% 這個的意思是a3 20%的元素會被歸零 為了不減低z[4]的期望值 您要做的是拿這個 將它除以0.8. 因為這會更正它或者說 將它提升回您需要的額外的20% 這樣就不會改變a3的期望值 這一行就是稱為 翻轉dropout 技巧 而它的影響是 不管您設keep\_prob多少 不管是0.8,0.9甚至是1 如果設成1 就不會有dropout 因為這會保留所有的 不管是0.5或其他 這種翻轉dropout技巧 是除以keep\_prob來確保a3的期望值不變 實際上在測試時 當您試著評估神經網路 我們將在下一張投影片談到 這種翻轉dropout 這一行用綠色框框起來 這會讓測試時間容易些 因為您減低了比例問題 至今最常用的dropout建置 就我所知是翻轉dropout 我建議您用這種方式建置 但有些早期dropout的迭代 少了除以keep\_prob這一行 在測試時演算法變得比較複雜 但人們傾向於不再用其他版本 您做的是您使用d向量 您注意到對於不同的訓練例子 您將不同的隱藏單元歸零 實際上, 如果您使用多次(循環)處理在同一個訓練集上 那在不同的處理資料集時循環時 您應該隨機將不同的隱藏單元歸零 所以並不是對於一個例子 您一直將同樣的隱藏單元歸零 在第一次梯度下降循環時 您也許將一些隱藏單元歸零 而在第二次梯度下降循環時 當您歷經訓練集第二次時 也許您會將不同樣子的 隱藏單元歸零 而向量d或者d3對於第三層 是用來決定哪些該歸零 同樣在正向傳播跟反向傳播時 我們這裡只展示正向傳播



現在如果您要在測試時訓練演算法, 您要這樣做 測試時，您會給予一些x來做預測 使用我們標準的符號 我將使用a[0]第零層的啟動值來記測試例子x 我們要做的是不使用dropout 在測試時, 特別是這是合理的z[1]=w[1]a[0] + b[1], a[1] = g[1](z[1]) ; z[2] = w[2]a[1] + b[2], a[2] = ... 等等, 直到您到最後一層您做預估y-hat 但請注意，測試時不使用dropout, 您不用隨機擲硬幣 您不用擲硬幣來決定哪一個隱藏單元來去除 而這是因為當您在測試時做預估 您真的不希望您的輸出是隨機的 如果您在測試時使用dropout 這只會在您的預估中增加雜訊 理論上, 您可以做的是跑預估流程很多次使用隨機丟棄不同的隱藏單元 然後跨過所有的隱藏單元 但這真的是效率不高的運算 且給您大約類似的結果 相當類似的結果對於這個不同的流程 還有要提醒一下 這種翻轉dropout 您記得前面投影片我們除以keep\_prob 那個的影響是確認即使當您不做dropout在測試時, 比例上這些啟動值的期望值不會改變 所以您不需要加入額外的比例參數在測試時 這是跟訓練時不同的地方 所以這是dropout 當您做這個禮拜的程式練習時 您會得到第一手的經驗 但為什麼它真的可行? 我想在下一段影片中給您 一些比較好的直觀有關於 dropout 真正在做些什麼 讓我們進入下一段影片

#### Understanding Dropout

Dropout 做著隨機敲掉您網路上的單元 看似瘋狂的事 為什麼它可以做到正則化? 讓我們來得到一些比較好的直覺

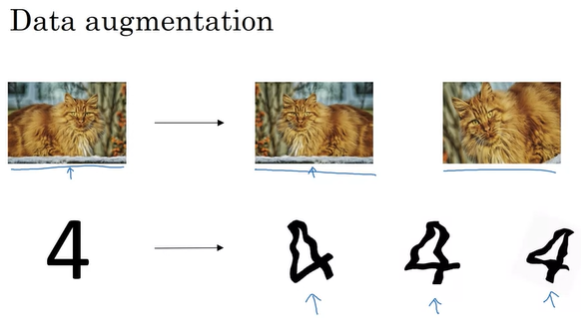


在先前的影片中, 我用dropout隨機敲掉您網路上的單元的這種直覺 這樣講好像是在每個循環中 您作用在小一點的神經網路 所以使用小一點的神經網路似乎像是有正則化的影響 第二種直覺是 讓我們從一個單一的單元來看 假設是這一個 對於這個單元它的工作是對於輸入 它需要產生有意義的輸出 現在因為dropout 這些輸入有可能隨機被去掉 有時候這兩個單元被消除 有時候不同的單元被消除 這個意思是這個單元 我用紫色來畫一個圈 它不能依賴任何一個特徵 因為任一個特徵都有可能隨機跑走或者任一個它的輸入都可能隨機跑掉 特別是會造成將所有的賭注放在這些輸入上的遲疑 假設說這個輸入權重來說, 我們會遲疑放太多的權重 在任一個輸入因為它會跑掉 所以這個單元會主動地分散權重 會給一點點權重到這四項每一個的輸入 而分散所有的權重 這會傾向縮減權重的平方範數的影響 所以類似於我們看到在L2正則化 建置dropout的影響是它會縮減權重來做類似於L2正則化的動作 來幫助避免過適 但實際上dropout可以正式地證明是一種自我調適的形式而不需要正則化 但L2懲罰在不同的權重是有所不同 它依據啟動值的大小來用它的方法相乘 但總結來說，是可能證明dropout跟L2正則化有類似的影響 只是L2正則化應用不同的方式可能些微不同, 甚至於能適應於不同比例的輸入 另一個細節當您建置dropout 這裡有一個網路有三個輸入特徵 這裡有七個隱藏單元 7,3,2,1 我們需要選擇參數 其中之一是keep\_prob, 也就是保留每一層的單元的機會 所以, 改變keep\_prob在每一層也是可行的 所以對於第一層 您的矩陣會是3乘7 您的第二個權重矩陣會是7乘7 w[3]會是7乘3 等等 所以w2實際上是最大一個權重矩陣 因為它實際上是最大的參數集 在w[2]是7乘7 所以為了避免, 為了減低那個矩陣過適 也許對於這一層 我猜這是第二層 您也許讓keep\_prob相對低一點 假設說是0.5 但對於不同層您或許比較不擔心過適 您可以有高一點的keep\_prob 也許就是0.7 如果一些層我們一點都不擔心過適 您可以將keep\_prob設為1.0 為了清楚起見, 我用紫色框框畫起來的數字 這些可以是不同的keep\_prob 在不同層 請注意設keep\_prob為1.0 意思是您將保留所有的單元 您實際上並不使用dropout在那一層 但對於那些您比較擔心會過適的層級 像是有很多參數的層級 您可以設keep\_prob小一點來應用dropout強而又力的形式 有點像是調整音量似的 正則化參數lambda在L2正則化您試著正則化一些層比某些層多 技術上而言, 您可以應用dropout到輸入層 您可以有一些機會來丟掉一些輸入特徵 雖然實作上,通常不這麼做 所以，在輸入層通常設keep\_prob為1.0 您也可以用很高的數字像是0.9 但比較少看到您想去除一半的輸入特徵 所以keep\_prob, 如果您用這個法則 即使您應用dropout在輸入層 也會是一個接近於1的數字

總結一下 如果您比較擔心一些層會過適 比起其他層而言 您可以設低一點的keep\_prob比起其他層 缺點是, 這會讓您需要搜尋更多的超參數使用在交叉驗證集 另外一種方式是一些層您應用dropout 而一些層您不用dropout 然後只用一個超參數 也就是keep\_prob 對於您想應用dropout的那些層 在結束之前, 一些建置的提示 很多第一個成功建置dropout是在電腦視覺 在電腦視覺裡輸入大小是如此之大 輸入所有這些像素值 資料幾乎從來都不夠 (因為輸入特徵太多,訓練例子會不夠) 所以dropout很常用於電腦視覺 有一些電腦視覺研究人員幾乎總是使用它 幾乎是預設 但請記得, dropout是一種正則化技巧來幫助避免過適 所以, 除非我們演算法過適 我不會麻煩用到dropout 所以在其他應用領域用的比較少 只是在電腦視覺裡您通常沒有足夠資料 所以您幾乎總是過適 這是為什麼一些電腦視覺研究者要用dropout 但這樣的直觀並不總是一般化到其他領域 一個重大的缺點是dropout的成本函數J不再完整定義在每一個循環中, 您隨機去掉一些節點 如果您重複檢查梯度下降的表現 實際上比較難檢驗您有定義良好的成本函數J 在每一次循環時望下坡走 因為您的成本函數J在做最佳化時實際上是比較小的 定義不明, 或者說很難去計算 您會失掉除錯工具來畫圖 像這樣圖形 我通常會將dropout關掉 您會設keep\_prob等於1 然後我跑程式，確定它是單調遞減的函數J 然後將dropout開啟, 希望我在使用dropout時 我的新程式不會有臭蟲 因為我想您需要其他方式(來除錯) 但不是用畫這圖來確定您的程式可行來讓程式可行 即使使用dropout 所以有了這個, 還有一些更多的正則化技巧值得您知道 讓我們在下一段影片中談更多的技巧

#### Other Regularization Methods

除了 L2 正則化以及 dropout 正則化 還有一些技術來減低 您神經網路過適的問題 讓我們看看



假設您配適一個貓的分類器 如果您有過適問題, 獲得更多的訓練資料會有所幫助，但獲取更多 訓練資料可能會很昂貴 有時候您就是沒辦法獲得更多資料 但您可以做的是 擴增您的訓練集像是拿一個影像 舉個例子 水平翻轉 將它也加入訓練集 現在不只是這一個例子 在您的訓練集 您可以加入這個到您的訓練例子 所以將影像水平翻轉 您可以將您的訓練集加倍 因為您的訓練集 現在有點多餘不像您 重新收集完全 新的獨立的例子 但您可以這樣做不需要 付任何費用為了 得到更多貓的照片 除了水平翻轉之外 您也可以隨機剪裁影像 這裡是我們旋轉 跟隨機放大這個影像 這還是像一隻貓 所以用隨機扭曲跟轉化影像 您可以 擴增您的資料集 增加假的新例子 再次, 這種額外假的例子 並不會增加 像是您去獲得全新的 獨立貓的例子的資訊 但因為您幾乎可以免費的做這個，除了 一些計算的成本 這個可以是最廉價的方式來 給予您更多的資料 因此也是一種正則化來減低過適 而透過綜合這些例子像這樣 您其實告訴您的演算法 如果一個東西是貓 水平翻轉後還是貓 請注意我並沒有垂直翻轉它 因為也許我們並不想要上下顛倒的貓, 對吧? 然後也許是隨機放大部份影像 這樣應該也還是一隻貓 對於光學文字辨識您也可以 將您的資料集將字元 實行隨機旋轉跟扭曲 所以如果您加入這些到您的訓練集 這些還是字元 4

為了解釋 我應用了很強的扭曲 所以這看起來很彎曲, 實作時您不需要將 4 扭曲 得這麼厲害，但只是些微的扭曲 就像我這邊做的 用這個例子讓您明白, 對吧? 但實作時通常用比較些微的扭曲 因為這看起來真的是很彎曲的 4 所以資料擴增可以用來 做正則化的技巧 實際上類似於正則化



還有一個常用的其他技術 通常稱為早期停止 您要做的是當您跑梯度下降時您會畫 您的不管是訓練誤差 您會使用 0 與 1 分類誤差在訓練集 或者畫成本函數 J 最佳化 那應該是單調遞減像這樣 是吧? 當您訓練時, 希望 您訓練時您的成本函數 J 應該要遞減 使用提前停止 您要做的是您畫這個圖 您也畫開發集誤差

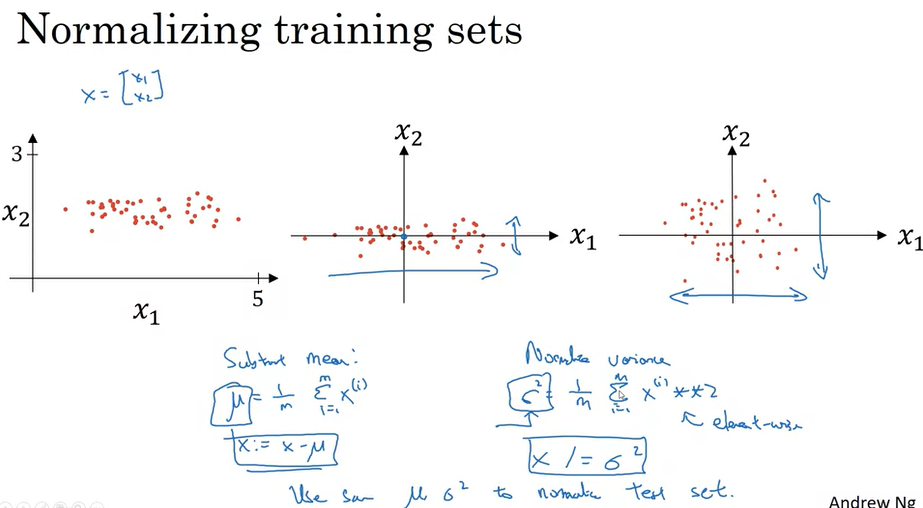
再次, 這可以是開發集上的分類錯誤, 或者一些 像是成本函數, 像是羅吉斯損失 或者對數損失在開發集上 您會發現到您的開發集誤差 通常會降低一會兒 然後會開始增加 而提前停止是 您會說 似乎您的神經網路 在這個循環做得最好 我們希望在訓練神經網路一半時停止 然後採用這個開發集誤差 不管到達何值 所以為什麼這樣可行 ？ 當您還沒跑很多次循環 對於您的 神經網路時 您的參數會接近於 0 因為隨機初始化也許 初始 w 為小的隨機值 在您訓練很久之前, w 還是很小 而當您開始循環, w 會越來越大, 直到 也許您有太大的參數 w 對於您的神經網路 而提早停止做的是 利用半途停止您會只有 中度大小的 w 而類似於 L2 正則化使用 選擇一個神經網路比較小的 範數對於您的參數 w, 希望您的神經網路不會過適 而這個提早停止術語 來自於您會 提前停止訓練您的神經網路 我有時候會使用提早停止 當訓練一個神經網路 但它有一個缺點 讓我來解釋 我想機器學習 包含了很多不同的步驟 其一是您要一個演算法 來最佳化成本函數 J 我們有不同的工具來使用 像是梯度下降 然後我們將會談到的其他的像是 momentum 跟 RMS prop 跟 Adam 等等 但經過最佳化以後的成本函數 J 您希望不要過適 我們有一些工具來使用 像是正則化 取得更多資料等等 在機器學習, 我們已經 有很多超參數像大浪襲來 從可能的演算法中來做選擇 已經很複雜的 所以我發現機器學習 比較容易想成 當您有一組工具來 最佳化成本函數 J 而當您注意在 最佳化成本函數 J 您要注意的是找到 w 跟 b 讓 J(w,b) 越小越好 您不去想其他的事 只是要減低這個 而不去過適它 完全是兩碼子的事情 換句話說 減低變異 而當您要處理它時 您有不同的工具來使用 而這個準則有時候 稱為正交法 而這種想法是您要 一次做一件事 我會談到正交法 在以後的影片中 如果您還沒完全搞明白這個觀念 不用擔心 但對我而言這種提前停止 的最主要缺點是 這會耦合這兩件事 所以您不再 將兩件事單獨作業 因為將梯度下降提前停止 您有點破壞了您正在做的 最佳化成本函數 J 因為您不再好好做 減低成本函數 J 的工作 您有點做不好那個 然後同時您又試著不要過適 所以與其使用不同的工具 解決兩個問題 您用了混合了這兩種在一起 而這會使得

您可以試的種種方式中 更加複雜去想用哪些 與其使用提早停止, 一種替代方式是使用 L2 正則化 然後您可以訓練神經網路越久越好 我發現這會使得搜尋 超參數的空間比較容易分解 比較容易搜尋 但缺點是您也許要試很多的 正規化參數 lambda 這會搜尋很多 lambda 的值 會產生很多 昂貴的計算 而提前停止的好處是 跑梯度下降 一次, 您試出小的 w 值, 中間的 w 值跟 大的 w 值, 不需要試很多 L2 正則化 超參數 lambda 如果這些觀念完全聽不明白 沒有關係 我們會談到正交的細節 在以後的影片 我想到時會更有感覺 儘管有它的缺點，很多人還是使用它 我個人比較喜歡 L2 正則化 試著不同的 lambda 值 這前提是您可以提供 必要的計算成本 但提前停止的確 讓您獲得類似效果 不需要明顯地試 很多不同的 lambda 值 您現在看過如何做資料擴增 以及如果您想提前 停止為了減低變異或者 避免過適在您的神經網路 下一段讓我們談談一些關於 設定您的最佳化問題來 讓您的訓練加快的技巧

### Setting Up your Optimization Problem

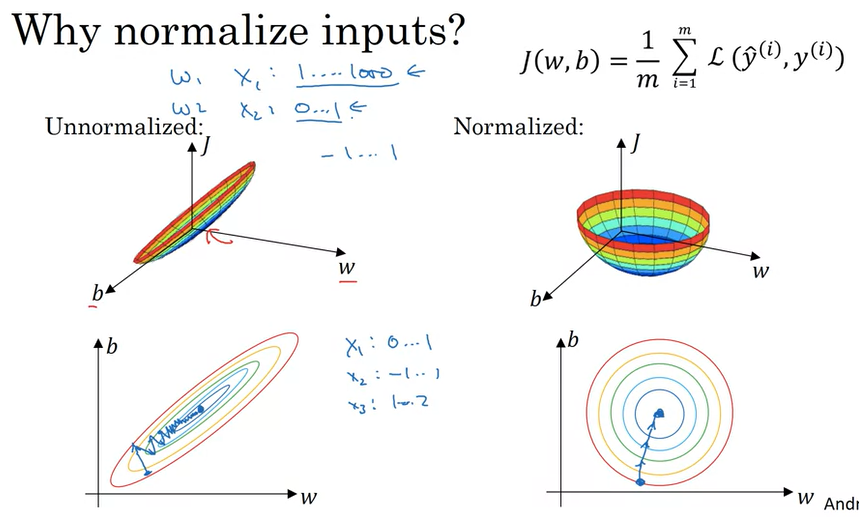
#### Normalizing Inputs

當訓練神經網路時, 加快您的訓練的技巧其中之一 是正規化您的輸入 讓我們來看看這是什麼意思



假設訓練集有兩個輸入特徵 所以輸入特徵 x 是二維 這個是您訓練集的散佈圖 正規化您的輸入相當於兩個步驟 第一步是減去 或者讓平均值為零 所以您設 mu = 1 除以 m 總和於 i of X(i) 所以這是一個向量, 然後 X 設為 X - mu 對於每一個訓練例子 這個意思是您移動 這個訓練集到 0 平均值 第二步驟是正規化它們的變異 請注意這裡的 x1 特徵 有比較大的變異 比起X2特徵來說 所以我們要做的是設 sigma = 1 除以 m 總和 X(i)\*\*2 這個是逐元素的平方 現在 sigma 平方是一個向量 是每一個特徵的變異數 請注意我們已經減去平均值 所以 X(i) 平方 逐元素平方只是變異數 然後拿每一個例子 除以這個向量 sigma 平方 在圖形上, 您最終會變成這樣 現在 X1 跟 X2 的變異都會等於 1

一個提示, 如果您用這樣來 按比例增減您的訓練資料, 那使用相同的 mu 跟 sigma 平方來正規化您測試集, 是吧? 特別是, 您不希望 用不同的正規化在訓練集跟 在測試集中 不管這個值是什麼跟 不管這個值是什麼, 用這這兩個 公式, 所以您用一樣的方式來 按比例縮放您的測試集, 而不是 計算 mu 跟 sigma 平方 分別在訓練集跟測試集 因為您要您的資料 不管是訓練或是測試例子 經歷同樣的轉換 用相同的 mu 跟 sigma 平方從訓練集計算出來的 為什麼做這些事 ？ 為什麼我們要對輸入特徵做正規化

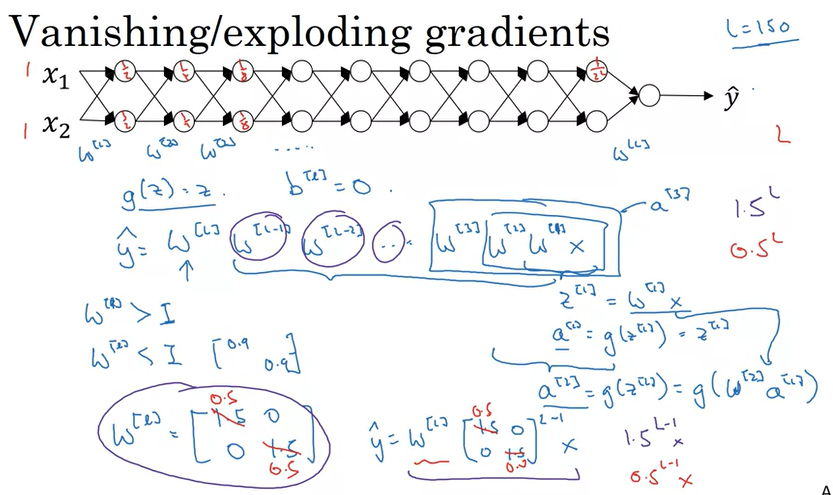


記得成本函數是 像在右上角定義的 實際上如果您用非正規化 輸入特徵, 它會 您個成本函數會像這樣 像是被壓扁的碗, 拉得很長的成本函數 而您試著要找的做小值會在這裡 但如果您的特徵是很不同的比例 假設 x1 特徵 從1 到 1000, 而 x2特徵從 0 到 1 那會變成對於 參數 w1 跟 w2 的比例或是值的範圍 會非常不同 也許這兩個軸應該是 w1 跟 w2 但我畫成 w 跟 b 那您的成本函數會是 細長的碗狀像那樣 如果您畫這個函數的等高線圖 您會得到一個 細長的函數像那樣 但如果您正規化這些特徵 那您的成本函數 平均來說或比較對稱 如果您跑梯度下降 在左邊的成本函數 那您會使用很小的學習率 因為如果您 在這裡，梯度下降會 需要很多步來回擺動 在最後終於找到到最小值的路徑 而如果您的是球型的等高線圖 不管您從何開始 梯度下降都可以 直接到最小值 您可以用比較大的步伐在梯度下降 而不需要 來回擺動像在左邊一樣 當然, 實務上 w 是高維度的向量 試著畫成 2D 並不能正確表示所有直觀 但大約的直觀是 您的成本函數會比較圓 比較容易最佳化當您的特徵 都在類似的比例時 不是從 1 到 1000, 從 0 到 1, 但 大部分從 -1 到 1 或者彼此類似的變異 這樣會讓您的成本函數 J 比較容易跟快速來最佳化 實際上如果一個特徵假設 X1 範圍從0 到 1 X2範圍 從 -1 到 1 X3 範圍從 1 到 2 這些都是類似的範圍 這個會作用得很好 就是對於那些 很大不同的範圍像是 一個從 1 到 1000, 另一個從 0 到 1, 這個 真的會傷害您的最佳化演算法 但只要設定它們都是 平均值 0 跟變異數 1, 像是我們 上個投影片做的, 保證所有的特徵在類似的比例上 通常會幫助您的 學習演算法跑得更快速 如果您個輸入特徵 有不同的比例 也許一些特徵是從 0 到 1 有些從 1 到 1000, 那正規化您的特徵是很重要的 如果您的特徵來自於類似的比例 那這個步驟比較不重要 雖然執行這個 正規化的步驟幾乎不會有任何 傷害, 所以我總是會執行它 如果我不確定是否 會幫助我加快 訓練我的演算法

所以這是正規化您的輸入特徵 下一段，讓我們繼續談論一些方式 來加快訓鍊您的神經網路

#### Vanishing / Exploding Gradients

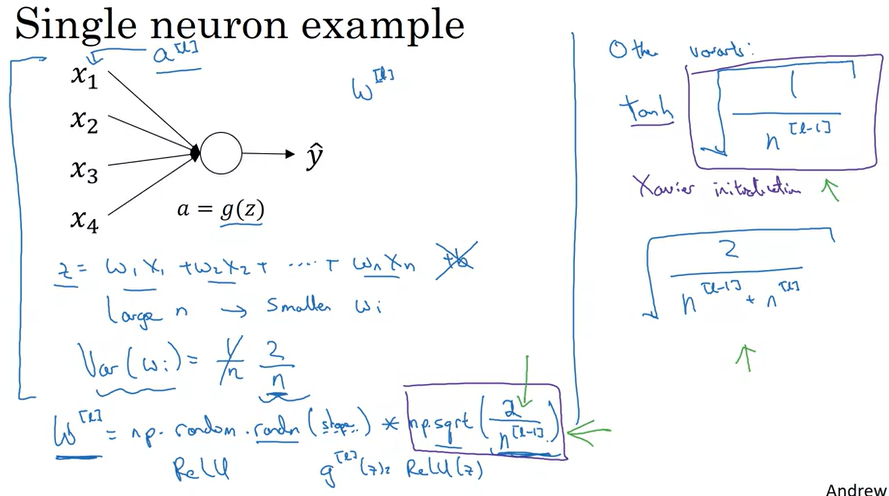
訓練神經網路的問題之一是 特別是很深的神經網路時 是資料消失以及梯度爆炸 這個的意思是當您訓練 很深的網路您的導數 或者您的斜率有時候會變得 很大很大 或者很小很小 也許是指數般的小 而這會讓訓練變得很難 在這段影片您會看到 梯度爆炸跟消失 的問題真正的含義 同時您如何可以謹慎選擇 隨機權重初始可以 明顯地減低這個問題



假設您正在訓練一個 很深的神經網路像這樣 為了節省投影片的空間 我畫成您只有 兩個隱藏單元在每一層 但它可以更多 這個神經網路會有參數 w[1], w[2], w[3] 等等到 w[L] 為了簡單起見 假設我們使用啟動函數 g of z 等於 z 也就是線性啟動函數 讓我們先忽略 b, 假設 b[l] 等於 0 這種情況下您可以證明輸出 y 會是 w[L] 乘 w[L-1] 乘 w[L-2] ...直到 w[3], w[2], w[1] 乘 x 但如果您要檢查我的數學 w[1] 乘 x 會是 z1 因為 b 等於 0 所以 z[1] 等於 w[1] 乘 x 然後加 b 但為 0 但然後 a[1] 等於 g of z[1] 但因為我們使用線性啟動函數 這個就等於 z[1] 所以這第一個項目 w[1]x 等於 a[1] 然後用類似的方式我們可以發現 w[2] 乘 w[1] 乘 x 等於 a[2] 因為那將會是 g of z[2] 將會是 g of w[2] 乘 a[1], 我們可以代入這裡 所以這個東西會等於 a[2] 然後這個東西將會是 a[3], 直到所有這些 矩陣的乘積會給您 y-hat, 不是 y 現在假設每個您個權重矩陣 w[L] 只有比 1 乘單元矩陣 大一點 所以像是 1.5, 1.5, 0, 0 技術上而言最後一個維度有所不同 也許這只是其餘這些權重矩陣 那 y-hat 會是 先忽略最後一個不同維度 這個 1.5, 0, 0, 1.5 矩陣的 L - 1 次方乘上 x 因為我們假設每一個 這些矩陣等於這個東西 這其實是 1.5 乘上單元矩陣， 那您最終會有這個計算 所以 y-hat 基本上會是 1.5 的 L 次方 L - 1 次方乘上 x 如果 L 很大, 是很深的神經網路 y-hat 會非常大 實際上, 這個是指數成長 它會長成 1.5 的層數次方 如果您有很深的神經網路 y 的值會爆炸 現在，相反地，如果我們取代這個為 0.5 是一種小於 1 的數字 那這個會變成 0.5 的 L 次方 這個矩陣會變成 0.5 的 L - 1 次方乘 x, 再次忽略 w[L] 所以每一個矩陣小於 1 假設 x1, x2 是 1, 1 那啟動值會是 一半 一半，四分之一 四分之ㄧ, 八分之ㄧ, 八分之一 直到這個變成 1/2 的 L 次方 啟動值會指數般減少 像一個網路深度的函數 也就是像函數對於 網路層的數目 所以在很深的網路時, 啟動值會指數般減少 這裡學到的直覺是從這個權重 w 如果他們只是比 1 大一點點 或者說把單元矩陣大一點點 那在很深的網路, 啟動值會爆炸 如果 w 比單元矩陣小一點點 那也許是 0.9, 0.9 在您用很深的網路時 啟動值會指數般下降 即使我歷經這個討論是 啟動值指數般增加或減少 是一個對於 L 的函數 類似的討論也可以用在證明 計算導數或者梯度在梯度下降 時也會指數般增加 或指數般減少 為一個對於層數的函數 一些現代的神經網路, L 等於 150 微軟最近用 152 層的神經網路 獲得很好的成果 但在這樣深的神經網路 如果您的啟動值或者梯度 指數般增加或減少為 L 的函數 那這些值可以很大或很小 而這會讓訓練更困難 特別是如果您的梯度是指數般小於 L 那梯度下降會用非常小的步伐 這樣會讓梯度下降 花很長的時間來學習 總結一下, 您看到深度網路如何承受 梯度消失或爆炸的問題 實際上, 長久以來這個問題是 訓練深度神經網路的很大障礙 實際上有部份解決方案 不能完全解決 這個問題但幫助很大 也就是 仔細選擇您的初始權重 讓我們在下一段影片 看這個方案

#### Weight Initialization for Deep Networks

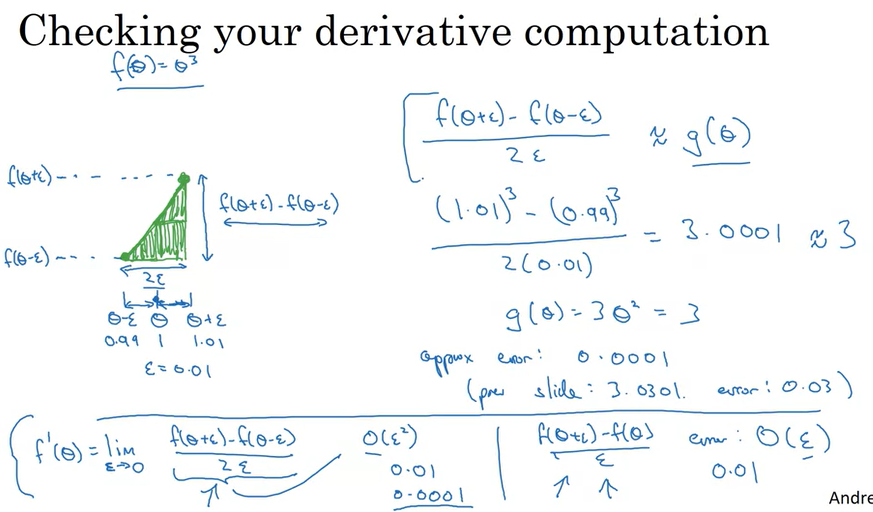
在上一段影片中, 您看到了很深的神經網路 可能有梯度消失或爆炸的問題 實際上有部分解決方案 無法完全解決但幫助很大 也就是小心選擇您 神經網路的隨機初始化 要理解這個讓我們從一個 初始一個單一神經元的例子開始 然後我們在一般化到深度網路去 讓我們用一個例子開始



只用一個單一神經元 然後我們等一下再來談深度網路 所以一個單一神經元您也許 輸入四個特徵 x1 到 x4, 然後您有一些 a = g(z) 最後會是一些 y 等一下在深度網路 這些輸入會是 一些層 a[l], 但現在我們先稱這些為 x 所以 z 會是等於 w1x1 + w2x2 + ... 直到 wnxn 假設 b = 0, 所以讓我們先忽略 b 為了讓 z 不會膨脹 也不會變得 太小, 您注意最大的 n 是 在您要的最小的 wi, 是吧? 因為 z 是 wixi 的和 如果您要將很多這些項目相加, 您要每一個項目都小一點 一種合理的方式是設 wi 變異為 1 / n 而 n 是輸入特徵的數目 進入一個神經元 實作上, 您可以做的是設權重矩陣 w 對於特定層 為 np.random.randn 然後無論這個矩陣的維度多少, 放進參數 然後乘上 平方根 of 1 除以特徵數目 餵入每一個神經元 也就是 n[l-1] 因為這是單元數目 我餵進每一個單元 實際上如果您使用 一個 ReLU 啟動函數, 那與其用 1/n, 實際上 設變異數為 2/n 會作用得好一點 所以您經常會看到這種初始化, 特別是如果您使用 ReLU 啟動函數所以如果 g(z) 是 ReLU(z) 取決於您多瞭解隨機變數 實際上 高斯隨機變數然後 乘上這個平方根 也就是說變異數 為 2/n, 我從 n 變成 n 上標 l-1 是 在這個例子用羅吉斯迴歸分析 所以是 輸入特徵, 但在一般情況下 l 層會有 n[l-1] 個輸入對於 這一層的每一個單元 如果輸入特徵或者啟動值大約是平均值 0 跟標準變異數 變異數 1, 那這會讓 z 用類似的比例, 而這個並不能解決 但應該會減低梯度消失, 梯度爆炸的問題, 因為這試著設定 每一個權重矩陣 w 不會太 大於 1, 也不會太小於 1, 所以不會太快爆炸或者消失 我剛提到一些變形 我剛提到的是假設 用 ReLU 啟動函數在 [聽不清](可能是 He's and Glorot's) 的論文中提到 其他的變形 如果您使用 tanh 啟動函數 有一篇論文顯示與其用常數 2 用 1 比較好，所以是 1 除以這個 而不是 2, 所以您乘上這個然後取平方根 所以這個平方根裡的項目會由 這個項目取代, 您用這個 如果您使用 tanh 啟動函數 這是稱為 Xavier 初始化 另一個版本我們會教到是 Yoshua Bengio 跟他的同事 您也許看過一些論文 但使用這樣的公式 有其他一些理論上的理由 但我會說 如果您使用 ReLU 啟動函數 也是最常用到的啟動函數 我會使用這個公式 如果您用 tanh, 您可以試這一版本, 一些作者也會 使用這個, 但實作上, 所有這些公式只是給您一個起點 它給您一個預設值來使用對於 您的各種不同的變異數的權重矩陣初始化 如果您要的話, 這個變異數 這個變異數參數也可以是 另一種您可以 調整的超參數, 所以您可以有 另一個參數乘上這個公式然後 調整這個乘數是 您超參數的一部份 有時候調整超參數有 中等程度的作用 它不會是我通常要 優先調整的超參數 但我也看過一些 調整這個超參數的問題 這個會有一些合理的幫助 但這通常對我而言比較低的優先權 對於其相對於 其他需要調整的超參數而言 所以我希望這會給您一些直觀有關於 梯度消失及爆炸的問題 以及如何去選擇 合理的比例對於如何來初始化權重 希望這樣做會讓您的權重 不要太快爆炸 也不要太快衰退 所以您可以 訓練合理深度的網路而不會 造成權重或者梯度太快爆炸或者消失 當您訓練深度網路 這是另一個技巧來幫助您 更快訓練您的神經網路

#### Numerical Approximation of Gradients

當您建置反向傳播時 您會發現有一種測試稱為 梯度檢查可以 真的幫助您確認 您的反向傳播建置 是正確的 因為有時候您寫了所有這些 方程式您並不能 100% 確定 您是否弄對所有細節 當您建置反向傳播時 為了要建立梯度檢查 讓我們先來談如何用 數值趨近來計算梯度 在下一段影片 我們會談到如何建置 梯度檢查來確定 反向傳播的建置是正確的



讓我們拿一個函數 f 這裡重新畫一次, 記得這個 f of θ 等於 θ 三次方， 讓我們開始一些 θ 的值 假設 θ 等於 1 現在與其將 θ 往右微調 加上 ε 我們要同時往右微調跟 往左微調得到 θ 減 ε 跟 θ 加 ε 所以這是 1, 這是 1.01, 這是 0.99 ε 跟之前一樣 0.01 實際上與其用這個小三角形來 計算高度除以寬度 您可以得到更加的梯度估計 如果您在這一點 f of θ - ε 跟這一點 您取而代之計算這個高度 除以這個大三角形的寬度 為了技術上的原因, 我不去深入談到 這個高度除以這個 大的綠色三角形的寬度會給您 比較好的在 θ 的導數估計 您自己也可以看到 用右上角的這個三角形 就像您有兩個三角形 一個在右上 一個在左下 而您有點像將兩個同時考慮 用這個大一點的三角形 所以與其用一邊的差異 您用了雙邊的差異 所以讓我們來做一些數學 這一點是 f(θ + ε) 這一點是 f(θ - ε) 所以這裡大綠色三角形的高是 f(θ + ε) 減去 f(θ - ε) 而寬度，這裡是一個 ε, 這是兩個 ε 所以這個大綠色三角形個寬是 2ε 所以高度除以寬度是 首先是高度 也就是 f(θ + ε) 減去 f(θ - ε) 除以寬度 也就是 2ε 我們寫在這裡

這希望靠近於 g of θ 所以代入這個值 記得 f of θ 是 θ 三次方 所以這個是 θ + ε 是 1.01 所以我取它的三次方減去 0.99的三次方 除以 2 乘 0.01 歡迎暫停影片 用計算機練習算一下 您應該會得到 3.0001 而從前面的投影片中 我們看到了 g of θ 是 3 θ 平方 而 θ 等於 1 這兩個數字實際上 非常接近 近似誤差是 0.0001 而在前面的投影片中 我們取單邊 的差異為 θ 跟 θ + ε, 我們得到 3.0301 近似誤差為 0.03 而不是 0.0001 所以這種雙邊差異的方式 來近似您要找的導數 是非常接近 3 所以這給您比較多的信心 來確定 g of θ 或許是正確的建置 f 的導數

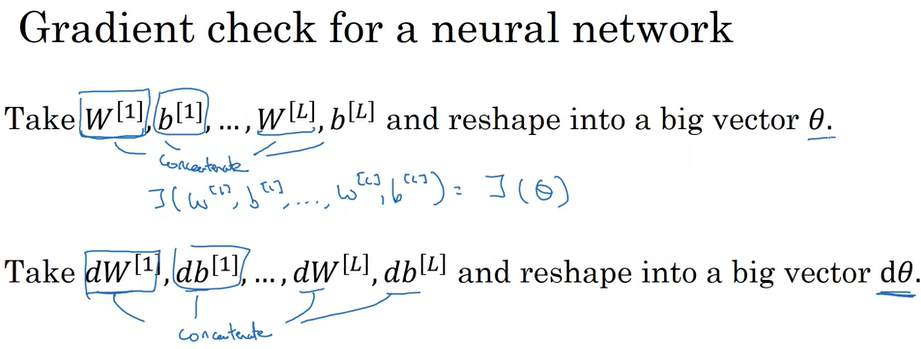
當您使用這種方式來作 梯度檢查跟反向傳播 這實際上會跑得兩倍慢 相對於您只使用單邊差異 實際上在實作中我覺得 用這種方式是值得的因為 這比較精確 再來一點可以選讀的理論 如果您比較熟悉微積分 實際上 如果您不懂我將要說的也沒關係 但實際上導數的正規定義是 對於很小的 ε 是 f(θ + ε) 減去 f(θ- ε) 除以 2ε 而正規的導數定義是在這個 右邊公式的極限值當 ε 趨近於 0 而極限的定義 是您學過的如果您 上過微積分課 我不會在這裡談到 而實際上對於一個 非 0 的 ε 值 您可以證明這個 趨近值的誤差是在 ε 平方等級的 記得 ε 是非常小的數 所以如果 ε 是 0.01 那這裡 ε 平方會是 0.0001 這個大 O 的意思是 誤差實際上是一個常數乘上這個項目 但這的確正是我們個近似誤差 所以這個大 O 常數剛好為 1 而相對的如果 我們使用這個公式, 另一個公式 那誤差會是 ε 等級 再次, 當 ε 是小於 1 的數字時, 那 ε 確實 比ε 平方大多了 這是為什麼這裡確實 比較不像左邊這個公式 那麼精確 這也是為什麼當作梯度檢查時 我們會使用雙邊差異 當您計算 f(θ + ε) 減去 f(θ - ε) 然後 除以 2ε, 而不用一邊的差異 因為他比較不精確

如果您不懂我最後這些意見, 這裡的東西 不用擔心 這是對於那些比較熟悉微積分 跟數值近似的人談的 但重點是雙邊差異公式 是比較精確的 這也是我們將在梯度檢查時 使用的公式, 在下一段影片中

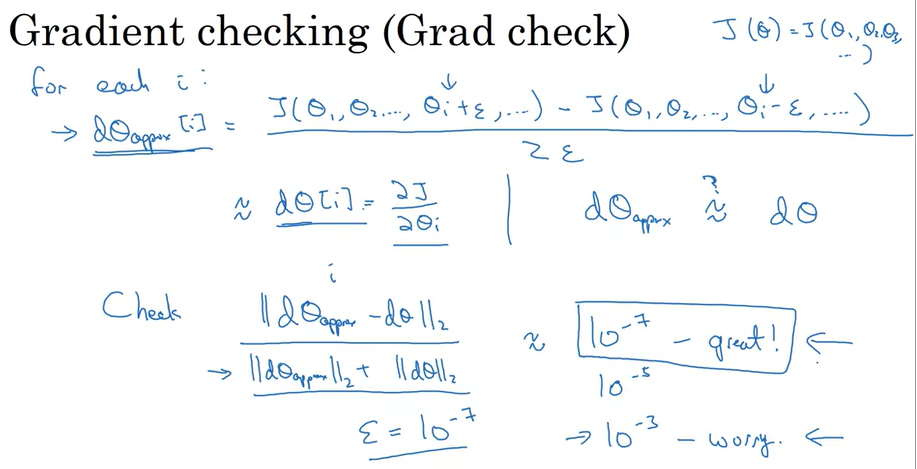
所以您看到如何使用雙邊差異 您可以數值驗證是否 有人給您的一個函數 g, g of θ 是正確的 f 函數的導數 讓我們看如何使用這個來驗證 是否您的反向傳播建置是正確或者 也許有錯誤您需要去挑出來

#### Gradient Checking

梯度檢查是一種 幫助我節省了大量時間的技術 多次幫助我發現在我的反向傳播 建置時的錯誤 讓我們來看 您也可以使用它來除錯 或者驗證您的反向傳播建置是正確的



您的神經網路有一些參數 像是 W[1], b[1], 等等到 W[L], b[L] 為了要建立梯度檢查, 第一件事是您應該要做的是拿所有您的 參數然後 重新建立一個大的向量 θ 您應該做的是拿 W 是一個矩陣 把它重塑成一個向量 您必須將所有這些 W 然後 重塑他們成為一個向量, 然後結合 所有這些東西, 變成一個大的向量 θ 大的向量發音為 theta 我們說成本函數 J 是 W 跟 b 的函數 您現在的成本函數 J 會變成 θ 的函數 接下來, 跟 W 跟 b 用相同方式排列 您也一樣將 dw[1], db[1], 等等 重塑它們變成一個 巨大向量 dθ 跟 θ 同維度 跟之前一樣, 我們重塑 dw[1] 這個矩陣, db[1] 已經是向量 重塑 dw[L], 所有 dw 的矩陣 記得 dw[1] 跟 w[1] 同維度 db[1] 跟 b[1] 同維度 用同樣的重塑及結合的運算 您可以重塑所有這些導數 成為一個大的向量 dθ 跟 θ 同維度 現在問題是 這個 dθ 是 成本函數 J 的梯度或者說是斜率嗎 ?

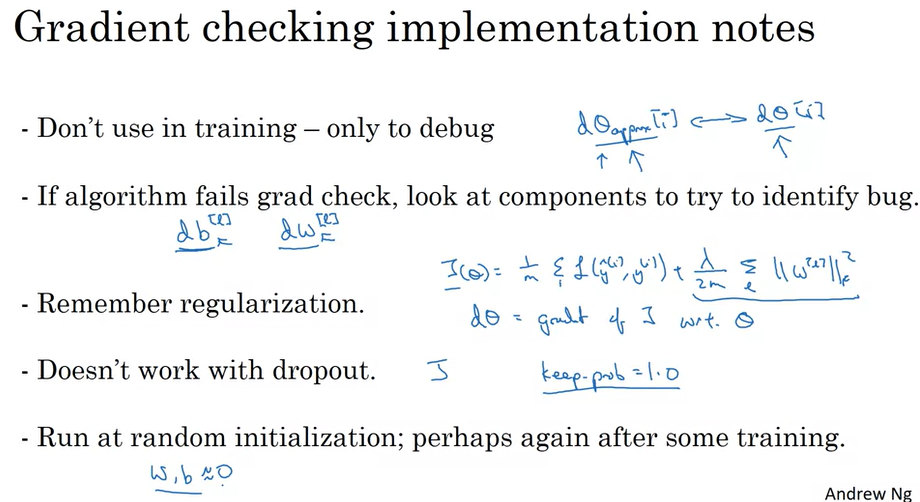


這個就是您如何來建立 梯度檢查 通常梯度檢查縮寫為 grad check 首先我們記住 J 現在是 一個大的參數 θ 的函數 對吧? 所以展開 J 是 θ1, θ2, θ3 等等的函數

無論這個大的參數向量 θ 的維度多少 要建立梯度檢查， 您要做的是建立一個迴圈 對於每一個 i, 對於每一個 θ 計算 dθ approx[i] 為 讓我們用雙邊差異 所以我拿 J of θ θ1, θ2, 直到 θi 而我們將微調 θi 加上 ε 只要將 θi 增加 ε, 其他保持一樣 因為我們取雙邊差異 我們同樣取 θi 的另一邊 減去 ε 所以其他元素都不去動它們 然後我們拿這個來除以 2ε (口誤) 我們從前面的投影片看到 這個應該會趨近等於 dθ[i] 也就是 J 的偏導數 相對於 我想是 θi, 如果 dθi 是成本函數 J 的導數 您要做的是計算每一個 i 最後 您會有兩個向量 您最終有這個 dθ approx 跟 這會是同維度的 dθ 而這兩個的維度都跟 θ 一樣 而您要檢查的是 是否這兩個向量幾乎相等 是相同的 詳細的說 您如何定義 兩個向量是否真的 互相彼此接近 我會這樣做 我會計算這兩個向量的 歐幾里德距離 dθ approx 減去 dθ 所以就只是 L2 範數 注意這裡上面沒有平方 這是總和於逐元素的差異的平方 然後取平方根 就像是歐幾里德距離 然後用這些向量的長度 來做正規化 除以 dθ approx 加 dθ 取這兩個向量的 歐幾里德長度 分母這一行就只是萬一任何一個向量很小 或者很大, 您的分母會 將這個公式變成比率 所以當實作時 我用 ε 等於也許是 10 的負7 次方, 所以 -7 使用這個範圍的 ε 如果您發現這個公式給您 的值類似於 10 的負 7 次方或更小 那很棒 這表示您的導數 近似值可能是對的 這真的是很小的數字 如果也許在 10 的 -5 次方範圍 我會小心的看看 也許沒事 但我會再次檢查 這個向量的分量 確定沒有一個分量是太大的 而如果一些分量的差異很大 或許某個地方您有一個臭蟲 (錯誤) 而如果左邊這個公式是 -3 次方等級 那我會擔心 會更擔心 也許某個地方有錯誤 但您應該得到的值會 遠小於 10 的 -3 次方 如果比 10 的 -3 次方大 那我會很擔心 我會嚴重的擔心 是否真的有錯誤 我會 您應該看每一個 分量的資料是否有特定的 i 使得 dθ approx i 非常不同於 dθi 使用它試著來追蹤是否 您的一些導數計算是錯誤的 而經過一定的除錯 最後它終於 到這個相當小的值 那您或許有一個正確的建置 所以當建立一個神經網路時 通常我會建立 正向傳播, 建立反向傳播 然後我或許會發現這個 梯度檢查相對大的價值 而當我懷疑有錯誤時 就一直除錯, 除錯, 除錯 經過一段除錯的時間後 我發現以小的值通過梯度檢查時 那我可以比較有信心, 它是正確的 所以您現在知道梯度檢查如何作用 這曾經幫助我找到很多的錯誤 在我建立神經網路時 我希望它也會幫到您 在下一段影片 我想分享您一些技巧 或一些筆記如何 建立梯度檢查 讓我們進入下一段影片

#### Gradient Checking Implementation Notes

上一段影片中 您學到了梯度檢查 在這段影片, 我希望分享您 一些實作的技巧或 一些筆記在如何真正 建立梯度檢查在您的神經網路上



首先，不要使用梯度檢查來做訓練 只用在除錯時 我的意思是 當計算 dθ approx i 時 對於所有 i 這是很緩慢的計算 所以當建立梯度下降時 您會用反向傳播來計算 dθ 然後用反向傳播來計算導數 而只有當您除錯時 您會計算這個 來確定它接近於 dθ 但當您做完檢查後 您要關掉梯度檢查 不要在每個梯度下降 的迴圈中執行它 因為它就是很慢 第二, 如果演算法在梯度檢查時失敗了 看一下分量 看每一個分量 試著找出錯誤 我的意思是如果 dθ approx 遠離於 dθ 我會看哪一個 i 的值使得 dθ approx 是非常不同於 dθ 的值 舉個例子, 如果您發現 θ (口誤) 或者 dθ 的值 它們遠離於 相對於 db[l] 在一些層中 但 dw 的分量卻相當接近 記得, 不同 θ 的分量代表 不同分量的 b 跟 w 當您發現如此時 您或許發現這個錯誤是 在您計算 db 時發生 導數相對於參數 b 同樣, 反之亦然, 如果您發現值是非常遠的 dθ approx 的值遠離於 dθ 您發現所有這些分量來自於 dw 或者一些層的 dw 這或許幫助您趨近 錯誤的位置 這不一定總是讓您 馬上找出錯誤 但有時候幫助您給您一些猜測 有關哪裡去追這些錯誤

接下來, 當做梯度檢查時 記得您的正則化 (regularization) 項目 如果您使用正則化的話 所以如果您的成本函數是 J(θ) 等於 1 除以 m 總和於您的 損失而 加上這個正則化項目 然後總和於 l of wl 平方 那這是 J 的定義 您應該有 dθ 是 J 的梯度 相對於 θ 包含這個正則化項目 所以請記得包含這個項目 下一個, 梯度檢查對 dropout 沒有用, 因為在每次迴圈時 dropout 隨機消除了 不同子集的隱藏單元 要計算成本函數 J, 當 dropout 應用在梯度下降時 是很不容易的 實際上 dropout 可視為 一些成本函數 J 的最佳化 但因為成本函數 J 定義為 總和於所有指數大的 節點的子集他們可能 被任何一個迴圈消除 所以成本函數 J 是 很難去計算 因為您會每次取樣於成本函數 在您使用 dropout 時 您會消除不同隨機子集 所以使用梯度檢查 來重複檢查您的 應用 dropout 的計算 是很難的 我通常建立梯度檢查時 不使用 dropout 如果您要的話, 您可以設定 dropout 裡的 keep\_prob 為 1.0 然後開啟 dropout 希望 我的 dropout 建置是正確的

有一些方式您可以做 像是固定某種形式的節點消除 利用梯度檢查來測試 這種形式是正確的 但實作時 我通常不這樣用 所以我的建議是先關掉 dropout 使用梯度檢查來檢查 當不使用 dropout 時, 您的演算法是正確的 然後才打開 dropout 最後, 這很微妙 這不是不可能, 或者說 極少發生但不是不可能 您建立梯度下降是正確的 當 w 跟 b 接近於 0 在隨機初始化時 但當您跑梯度下降時 w 跟 b 越來越大 或許您的反向傳播建置在 w 跟 b 接近於 0 時是正確的 但當 w 跟 b 越來越大時 它變得不正確 您可以做一件事 我通常很少這樣做 但您可以做一件事是 在隨機初始時跑梯度檢查 然後訓練網路一段時間 當 w 跟 b 慢慢從 0 離開 從您小的初始值離開 然後訓練一些迴圈後 再跑一次梯度檢查 所以這是梯度檢查 恭喜您來到這個禮拜 最後的內容 在這一週, 您學習如何 設定您的訓練, 開發及測試集 如何分析偏差及變異 當如果您有高偏差 或高變異時跟 同時高偏差跟高變異時, 怎麼辦 您也學了如何 應用不同種類的正則化 (regularization) 像是 L2 正則化跟 dropout 在您的神經網路 然後一些技巧來 加快訓練您的神經網路 最後是, 梯度檢查 我想這個禮拜您看了很多內容 您將練習這些觀念在 這個禮拜的程式練習中 祝您幸運 我期望在第二週再遇見您

### Quiz：Practical aspects of Deep Learning

### Programming Assignments

### Heroes of Deep Learning (Optional)

#### Yoshua Bengio Interview

Hi, Yoshua, 今天真的很高興你能加入我們 >> 我也很高興 >> 你現在不僅是一個深度學習的研究者或工程師 你已經變成了深度學習中的名人、偶像 不過我很想聽聽這故事的起源 那麼，當初你是怎麼踏入這深度學習，然後持續這旅程呢？ >> 是的，其實呢，故事從小時候就開始了，少年時 讀了很多科幻小說，我猜我們很多人都這樣 當我1985年開始研究所的時候， 我開始讀類神經網路的論文 這讓我非常興奮，變得真正的熱情 >> 實際上，那感覺是怎麼樣的呢， 在80中期對吧，1985， 在讀那些論文的時候，你還記得嗎？ 有對

从 ::59 开始播放视频并学习脚本0:59

你知道的，本來我學的是專家系統那種古典AI的課程 然後突然發現有那麼一個新大陸 是去思考人類如何學習、人類的智慧 然後可能可以跟人工智慧、 跟電腦聯結起來。 當我發現這些文獻時，我真的很興奮 我開始閱讀 Connectionists，這當然， 還有 Geoffrey Hinton、Rumelhart 的論文，等等等 我研究 RNN (遞歸神經網絡)，我研究語音辨識 我研究 HMM (隱馬爾可夫模型)，所以有 圖模型 (graphical model) 然後很快地，我進入 AT&T 貝爾實驗室和麻省理工學院， 我在那邊做博士後研究 就是在那裡，我發現一些訓練神經網路會碰到的 長期相依性的問題 (long-term dependencies) 過了不久，蒙特婁大學招募我，回去蒙特婁 那也是我度過大半少年時期之處

>> 那麼，身為一個在那打滾數十年、閱歷無數 見多識廣，告訴我你的想法 關於深度學習、神經網路在這些年的演進

>> 一開始，我們從實驗、從直覺觀念出發， 算是之後再看理論 現在，我們了解的更多更深，例如 為什麼反向傳播這麼有用，為什麼深度這麼重要 而對這些見解，在當時我們講不出很確實的理由 在2000年初期，當我們開始研究深度的網路， 我們有個直覺， 更深的網路應該更強大，這很合理 但是，我們不知道要怎麼運用 怎麼證明，而當然我們的實驗在一開始並沒有成功

>> 實際上 回顧過去，你覺得有哪些最重要的事情， 最後被發現是正確的 而哪些事最終發現是錯的、最讓人訝異的 如果和三十年前我們所知的來比較？

>> 當然，我過去犯的最大的錯誤，是認為 — 就和90年代的每個人一樣 認為反向傳播要有效，就一定要有平滑非線性的性質 因為我以為，如果有像是整流般非線性 (rectifying nonlinearities) 其中有個平坦的部份，這樣會非常難訓練 因為有太多地方的導數會是零 而我們一開始嘗試 ReLU 的時候 大約在2010年，用在深度網路上，我很執著在 「喔我們要小心， 會不會神經元有很多在0那邊飽和啊？」 不過最後事實證明，ReLU 實際上表現遠比 S型函數 (sigmoid) 還要好，這讓人非常驚訝 我們會做這個探索，其實是因為生物學的關聯 並不是因為我們覺得這樣比較容易優化 不過最終他表現的比較好，而不是像我想的比較難訓練 >> 那麼讓我問你 深度學習和大腦之間有什麼關聯？ 有一個明顯的答案，但我很好奇你的答案是什麼？ >> 嗯，一開始讓我對神經網路 感到興奮的見解，是 Connectionists 上的某個意見： 資訊是分佈在眾多神經元的啟動 (activation) 上 而不是由某種「祖母細胞」(grandmother cell) 來表示 像那時他們稱呼的，一種符號性的表示法 那是古典人工智慧中的傳統觀點。 我現在仍然相信這真的很重要 甚至在最近，我看到也有人也發現了這個的重要性 因此那真的是個基礎。 「深度」這件事是之後出現的，在2000年初期 至少不是我在90年代在想的 >> 的確，我記得你當時建了很多比較淺 但是很分散式的表示法來做 word embedding (詞向量)， 在很早期的時候 >> 是的，沒錯 那是在90年代後期讓我很興奮的其中一件事 事實上，我和我的兄弟 Samy 當時在研究一個點子： 我們可以用神經網路 來解決 curse of dimensionality (高維詛咒)，這被認為 是統計學習裡其中一個主要的問題 而我們有了這些分布式表示 (distributed representation) 後 我們就可以 很有效率地表示多個隨機變數的聯合分布 這個最後非常有效，然後我把他擴展到 一串字的聯合分布，這個就是 word embedding 的由來 因為我認為，這可以讓許多那些 意義類似的字有更共同、廣義的表示 (generalization) >> 所以在過去幾十年，你的研究團隊發明了超級多的想法 多到沒有人可以用幾分鐘說完 所以我很好奇，你團隊的哪些發明或點子 是你最感到驕傲的？ >> 好，我想我曾經提到長期相依性 (long-term dependencies)，那個研究 我覺得人們還不夠了解這個。 然後我剛剛說的故事，關於高維詛咒， 用神經網路處理聯合分布 (joint distribution)， 這在最近演進成 Hugo Larochelle 研究的 NADE (Neural Autoregressive Distribution Estimation) 然後像我剛說的，這啟發了許多研究 訓練 word embedding 來探討字詞的聯合分布。 接下來我想，或許是深度學習中我們最有名的研究 利用堆疊的 autoencoders 和堆疊的 RBMs (自編碼器 和 受限玻爾茲曼機)。

然後是一個研究，對深度網路訓練的難處了解更多 和 Glorot 想出初始化的點子 還有深度網路的消失梯度問題 (vanishing gradient) 實際上就是這項研究，引發了大家的實驗來證明 分段的線性啟動函數有多麼重要。 然後我會說，我們最重要的研究，有一些和 非監督學習、denoising autoencoders、GANs 有關 (降噪自編碼器 ; 生成式對抗網路) GAN 最近非常流行，Generative Adversarial Network。 還有我們利用 attention 做神經機器翻譯的研究 後來發現這個對機器翻譯非常的重要 而且現在業界系統也在用，例如Google翻譯 不過這個 "attention" 機制真的改變我對神經網路的看法 我們曾經認為，神經網路就是把一個向量對應到另一個向量 但是有了 attention 機制，你可以處理任何的資料結構 這開創了很多條有趣的研究路線。 而和生物學有關的方向 近幾年我在做的一件事情是 我們怎麼弄出一套類似反向傳播的東西， 但是人類大腦可以實作 在這方向我們有幾篇論文，看起來神經科學領域的人 覺得這很有趣 當然我們也持續在這個方向進行研究。

>> 我知道有一個主題，你一直花了很多心力思考 是深度學習和人類大腦之間的關連 可以和我們多談一些嗎？ >> 這個生物學的問題，實際上我不斷想了很久 或者可以說，做了很多白日夢吧 因為，我覺得這就像個拼圖 我們有一些片段的證據，是關於大腦以及大腦怎麼學習 例如 Spike-timing-dependent plasticity 而另一方面，我們有這一堆機器學習的概念 例如對一個目標函數做整個、整體系統的訓練 還有反向傳播這個概念 其實，到底「反向傳播」是什麼意思？ 就像，"Credit Assignment" 到底是什麼意思？ 當我開始思考，人類大腦是如何做類似反向傳播的事 這讓我繼續思考，嗯，或許在反向傳播的背後 有個更一般的道理，讓反向傳播能如此有效 而或許有很多種類的方式可以解決 credit assignment 而這些和做 reinforcemnet learning (增強式學習) 的人 一直在問的問題有關 所以有趣的是，有時候問一個簡單的問題 能導引你去思考眾多不同方面，迫使你去思考 許多不同的元素，像拼一個很大的拼圖把他們都串起來 我這樣已經很多年了 而我必須說，這整個努力、我追逐的很多主題 一直以來都是受 Geoffrey Hinton 的強烈啟發 特別是，我猜是在2007年，他給了一個演講 在第一次的深度學習研討會上 他分享了他認為人腦怎麼運作，

類似像時序編碼 (temporal code) 可能怎麼 做到反向傳播所做的 這讓我在最近幾年有很多探索的想法

嗯，這算是個滿有趣的故事

算算也十年了

>> 還有一個主題我也聽你談過很多次，是關於 非監督學習 (unsupervised learning) 你可以分享一下你的觀點嗎？

>> 當然當然，非監督式學習相當重要 目前業界的系統是基於監督式的學習 (supervised learning) 這本質上需要人類去定義 在某個問題中，什麼是重要的概念 以及在資料中把那些概念標示出來 你知道的，我們藉此做出許多玩意和服務和系統 不過人類能做的比這還多 藉由對這世界的觀察和互動，我們能探索、 能發現新的概念。 一個兩歲的小孩，能理解直觀的物理現象 換句話說，他知道重力，他了解壓力 他知道慣性 他懂得液體和固體 當然，他的父母從來沒教他這些東西，對吧？ 那他是怎麼知道的？ 這種問題就是非監督式學習想要回答的 這不僅僅是我們有沒有資料的標籤的問題 實際上這是一種心靈的建構 藉由觀察，來了解、解釋世界的運作 而最近，我一直在結合 非監督學習和增強學習兩者的概念 因為我認為，有很強烈的跡象顯示 那些我們想要釐清的、想要區分的 那些重要的最根本的概念

可以由人類或機器透過和世界互動、 探索這個世界、嘗試事物、嘗試控制來獲得 所以我認為這些和非監督學習的原始概念息息相關 學習. 所以我對非監督學習的意見： 十五年前當我們開始做 autoencoder 和 RBM 之類的 我們非常專注在如何學習良好的表示法 現在我仍然認為這是必要的課題 不過我們不清楚的是，什麼叫一個「良好的表示」 例如我們怎麼知道目標函數是什麼? 所以這些年我們試過非常多的東西 實際上，這是研究非監督式學習很酷的一件事 有超多各式各樣的點子 太多不同的方法來解決問題 這實在是...搞不好明年我們會發現完全不一樣的東西 而或許人腦運作方式又是另一種不一樣的 這種研究不是慢慢疊加上去的 這領域本身就非常值得探索

對於正確的目標函數該是什麼，我們沒有好的定義 更不用說怎麼測量一個系統的好壞， 在非監督學習的領域中。 所以當然，這很有挑戰，但同時 這也意味著無窮開放的可能性 而研究人員很喜歡這樣，至少這很吸引我

>> 時至今日，深度學習有太多事在發生了 我覺得我們已經過了某個階段 一個人已經沒辦法讀過所有深度學習的論文了

所以我很好奇，深度學習的哪些事最讓你興奮呢？ >> 那麼，我很有野心，而我覺得現今 深度學習的科學面還遠不及我所期望的 我有一種印象是，我們系統犯的那樣子的錯誤 顯示了那些系統對這世界的了解還很膚淺

因此，最讓我興奮算是研究的方向 我們不是想建造系統讓他們做有用的事情 我們是回歸到基本原理：電腦如何觀察世界 和世界互動，以及發掘世界如何運作？ 就算那個「世界」是很單純的，例如做成電動遊戲的形式 我們仍然不知道如何表現得好 這很棒，因為我不需要和Google和Facebook競爭 以及百度等等，對吧 因為這算是基礎研究 任何人可以在他的車庫進行，然後改變全世界 因此，當然，有很多很多的方向來著手 不過我看到很多深度學習和增強學習的點子 彼此間有豐碩的交流，這非常重要 而且我非常興奮這方向的進展 其實可能對實際應用有巨大的影響 因為，如果你觀察我們在應用層面碰到的一些大挑戰 例如怎麼處理新的領域，或者 一個類別裡面我們只有非常稀少的資料 以及那些人類非常擅長的情況。 所以這類 transfer learning 和 generalization 的挑戰 (遷移學習 ; 一般化/泛化) 如果我們的系統可以更了解這世界如何運作 那麼這些挑戰就顯得輕而易舉 要有深入的了解，對吧 到底發生什麼事？ 造成我看到這些的原因為何？ 我要採取什麼行動來影響我所看到的？ 所以這一類的問題是我最近倍感興趣的 我也認為，隨著深度學習研究演進了幾十年 這些已經牽涉到了更古老的 AI 課題 因為現今深度學習大部分的成功 都和「感知」能力有關 那接下來呢？ 剩下來的是更高層次的 也就是在抽象的層面上，理解萬物之運作 我們要理解高層次的抽象，而我認為 還沒達到那麼高的境界，所以我們必須要達到 我們必須思考推理、思考時序性的資訊處理 我們必須思考因果關係怎麼運作 以及要怎麼樣讓機器自己發掘這些事情 或許需要人類引導，至少機器要越自主越好 >> 你剛剛說的某一段，聽起來像是 你很喜歡用作研究的方法，也就是 — 我要用「玩具問題」這個詞，並沒有貶低之意 >> 好。 >> 在小問題上實驗 而你很樂觀認為這小問題可以轉移、類推到更大的問題 >> 對對沒錯，在高層次上類似地轉移 當然我們要花些功夫把格局弄大 然後解決那些問題 不過追求「玩具問題」的主要動機是， 我們能更清楚了解失敗在哪 我們可以把問題簡化到 從直觀上就能處理，更容易理解 好比典型科學在做的 divide and conquer 各個擊破 還有，我覺得，有件事大家想的不透徹 研究的週期會變快的多，對吧 如果我可以幾小時就做完一個實驗，我的進展就更快 如果我要嘗試一個超巨大模型， 試著捕捉所有的常識和知識 — 其實總有一天可以的 只不過以現今的硬體，每一個實驗就等到天荒地老 所以當我們的硬體夥伴還在建造那些 千倍萬倍飛快的機器，在那之前我還是做玩具實驗 >> 我也聽你說過 深度學習的科學，不只是一門工程學科 更是多多研究，了解究竟是怎麼回事 你想分享一下嗎？ >> 好啊，當然 我擔心很多我們正在做的研究，恐怕像是 盲人尋路 如果你運氣夠好，這樣也能找到有趣的收穫 不過說真的，如果我們稍微停下腳步 用可以類推的方式，試著了解我們在做什麼 因為我們遵照原理、遵照理論 — 但是我說的「理論」不見得是「數學」 當然我喜歡數學啦，不過我覺得 不用把每件事都用數學公式化 — 但是要有邏輯 換句話說，我可以說服大家這個應該有用、 這個有沒有道理 這是最重要的一點 然後數學讓我們更強更穩固 不過真的，這其實和「理解」有關 也和做研究的道理有關 不是為了成為下一個目標、基準 也不是擊敗其他實驗室或公司的其他人 而是：我們要問什麼樣的問題，才能讓我們 更理解有趣的事物現象 舉個例子 是什麼導致訓練深層網路或 RNN 很難？ 我們有一些想法，但很多事情我們還不明白

所以或許我們能設計實驗，其目的不在得到更好的演算法 而僅是更加了解現有的演算法 或是某個演算法在什麼樣的情況下比較有效？為什麼？ 「為什麼」才是最重要的 這就是科學在做的 找到「為什麼」 >> 好的，現在有很多人想要 進入這個領域 我相信你已經在很多一對一的場合回答過 不過對於正在看這影片的大家，如果他們 想踏入 AI，想開始深度學習，你有什麼建議？ >> 好，首先呢，動機有很多種 你能做的也不一樣 要成為一個深度學習的研究者所需要的，可能和 使用深度學習開發產品的工程師，兩者需要的不一樣 這兩個狀況所要求的理解程度會不同 不過無論如何，練習、再練習 如果要真正掌握深度學習 當然你要讀的多 你要嘗試自己手刻程式做練習

我常常面試到一些學生，他們用一些軟體 而現今有太多太好的軟體，你只消裝來就用 完全不用知道自己在做什麼 或者只有很粗淺的了解，於是乎 就很難發現何時無效、哪兒出問題 所以，試著自己實作東西，即使這沒效率 這只是為了確保你真的理解發生了什麼事 這真的很有用，自己多嘗試 >> 所以不要光是拿某個程式框架， 做每件事只需要幾行程式，但你其實不了解背後發生的事 >> 沒錯、沒錯！甚至不只如此 試著靠自己從基本原理推導各種東西，如果可以的話 這真的很有用 不過呢，至少一些平常的要做，像是閱讀、 看別人的程式、自己寫程式、 做很多的實驗、確保你都理解你所做的 特別是科學的方面 試著問自己：為什麼我在做這個？為什麼大家在做這個？ 或許答案在某本書，而你要多多閱讀

不過如果你能自己想通，那樣更棒

>> 讚讚 事實上說到閱讀，你和 Ian Goodfellow 和 Aaron Courville 寫了一本很高評價的書 >> 謝謝、謝謝 嗯，這本賣得很好 有點誇張 我覺得在讀這本書的人比目前看得懂的 還要多 不過對，還有 ICLR 的論文 ICLR 大概是最佳的好論文集散地 當然 NIPS 和 ICML 和其他研討會也有非常棒的論文 不過如果你真的想看很多好的論文，就去讀 最近幾次的 ICLR 論文，會給你這領域的全貌 >> 太好了 還有其他的想法嗎？ 如果請你給個建議，要如何擅長深度學習 >> 嗯，這要看你的背景 不要害怕數學 先養成直覺，當你能從直觀上了解 熟悉事物的原理，那數學方面就很容易理解了 而且有個好消息是，你不需要去讀五年的博士 來變成深度學習的專家 事實上你可以很快學會 如果你有電腦科學和數學的良好背景 幾個月的學習你就可以使用、做出東西 開始研究、開始實驗 經過正確的訓練，大概約莫六個月 就算可能對機器學習一無所知 如果他們擅長數學和電腦科學，這可以很快 不過當然，這意味著你要受過正確的數學 和電腦科學的訓練 有時候只靠電腦課所教的並不夠 特別是你還需要 連續函數的數學 (相對於離散數學) 也就是例如機率、代數、最佳化 >> 了解。還有微積分 >> 還有微積分，對對 >> 非常感謝, Yoshua, 分享這些意見、洞見和忠告 就算我已經認識你很久了，有很多細節、 早期歷史我到現在才知道，謝謝你 >> 也謝謝你 Andrew 特別做這影片，還有你所作的這些 我希望這能被很多人看見