DSP Final Project -

Deep Neural Networks

R02922036 施詠翔

前言

最近 Deep Neural Networks 的話題似乎愈來愈熱門，連網路媒體上都時有所聞。而且一直以來，我恰好也對人腦的議題有很大興趣，看過不少相關科普書籍，因此對受神經組織啟發所產生的 Neural Networks 存有一點憧憬。於是便決定要以閱讀 Deep Neural Networks 的相關文章作為這次期末報告的主軸。

在這份報告裡，我將簡述 DNNs 的發展。另外，我將特別針對 Restricted Boltzmann Machines 做介紹，並且實作一個簡單的程式。最後則以對數位語音未來的樂觀期待作為結束。

Deep Learning 簡介

在 [1] 中指出，許多舊有的 machine learning 和 signal processing 的方法都是奠基在比較淺的結構上，她們通常只有一層結構可以將輸入的資料轉換到跟問題相關的 feature space。然而從人類的資訊處理方法看來，其實整合了各種不同的輸入，並且有多層次的概念和結構。因此人們相信，如果能找出學習多層次的方法，就可能得到比較強大的資訊處理系統。

在一個擁有深層結構的 signal processing 系統中，通常有多個非線性轉換的階段，每個階段將輸入轉換成輸出，然後再將輸出當成下個階段的輸入。而 deep learning 指的則是訓練這麼多個階段轉換的方法。Deep learning 的概念最早是從 neural networks 領域發展出來的，擁有多層 hidden layers 的 perceptron 就是一個擁有深層結構的模型。

在 [2] 中提到，傳統使用 Gaussian mixture models 的語音辨識系統很難去 model 接近 nonlinear manifold 的資料分佈，但 artificial neural networks 有潛力去學習這樣的 model。只是早期的時候，沒有好的演算法，所以非常難以訓練擁有多層結構的 DNNs，很容易就會陷入 local optima 而無法自拔。

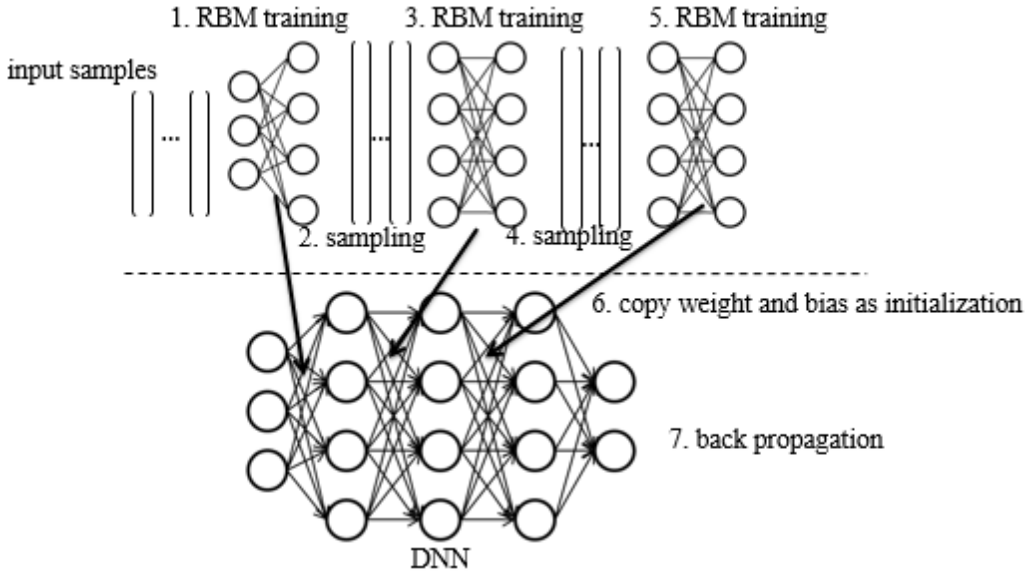
到了 2006 年 Hinton et al.[3] 提出了一個 deep belief networks 可行的學習演算法，才讓整件事改觀，而在那以後，不同的 deep learning 方法也被陸續提出。

其中，這類演算法的主要架構通常分為兩個階段，首先是 generative pretraining，讓 model 盡可能的和資料的分佈變得相像。接著再以這作為起點，進入 fine-tuning 的階段，分出想識別的 classes。

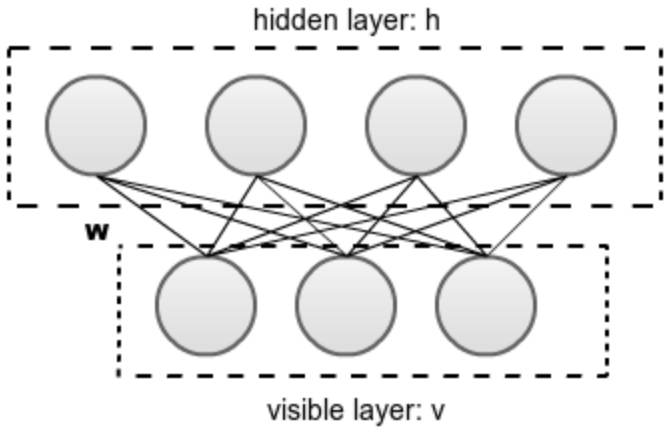
有趣的是，在我們課堂上曾提過，Gaussian mixture models + HMM 一開始是用 generative 的方法，讓 model 產生的分佈跟資料儘可能的相像，可是後來發現，再用 discriminative training 將想判斷的 class 分開，可以得到更好的結果。但 [2] 中提到，neural nets 剛發展出來時，是使用 discriminative training，直到後來才發現用 generative 的方法做 pretraining 可以得到較好的效果。

Restricted Boltzmann Machine (RBM)

為了要訓練 deep belief networks，在一開始 generative 的初始化中，可以使用 RBM training，並一層一層慢慢推展下去，正如講義中的圖所示：



所謂的 Restricted Boltzmann Machine，是一種特別的 Markov Random Field，如下所示，可以分為一層 hidden layer 和一層 visible layer：



其中，每個 hidden layer 的 node 都和全部的 visible layer nodes 相連，而每個 visible layer node 也和全部的 hidden layer nodes 相連。但同一個 layer 彼此不相連（所以才稱為 restricted）。

在 [2] 中提到，我們可以找到一個方法去訓練這個 RBM，首先定義整個系統的 energy 為：

$$E(v,h) = -a^T v - b^T h - v^T W h$$

其中 a 為 visible layer 的 bias， b 為 hidden layer 的 bias，而 v、h 分別為以 vector 表示的 nodes 數值，W 則是用來敘述兩層 layers 關係的 weight matrix。

這樣的話，我們就可以說任意一組 v, h 的機率如下：

$$p(v,h)=\frac{e^{-E(v,h)}}{Z},\quad Z=\sum_{v,h} e^{-E(v,h)}$$

其中 Z 被稱作是 partition function，是一項 normalizing factor。

如果要算一組 v 的機率，則可以將所有的 h 下的機率加起來：

$$p(v)=\frac{1}{Z}\sum_{h} e^{-E(v,h)}$$

這樣就可以算出 training set 對應 weight 的 log probability 的偏微分：

$$\frac{1}{N}\sum_{n=1}^{n=N}\frac{\partial \log p(v^n)}{\partial w_{ij}} = E[v_i h_j]_{data} - E[v_i h_j]_{model}$$

其中，N 為 training set 的大小， v^n 則是第 n 組資料。而透過這個結果，讓我們得到更新 w_{ij} 的方法：

$$\Delta w_{ij} = \eta(E[v_i h_j]_{data} - E[v_i h_j]_{model})$$

其中 η 為 learning rate。

假設資料只有 binary 的數值，則同一 layer 沒有相連的特性使我們很容易可以找到 $E[v_i h_j]_{data}$ 的 unbiased sample：

$$p(h_j=1|v)=sigmoid(b_j+\sum_i v_i w_{ij})$$

$$p(v_i=1|h)=sigmoid(a_i+\sum_j h_j w_{ij})$$

而 $E[v_i h_j]_{model}$ 由於沒有簡單的方法求得，所以採用 Gibbs sampling 的方法來計算。

RBM learning 實作

雖然讀過以上的說明，還是很難弄懂實做的方法，所以我又另外參考了 [4][5][6][7][8] 等教學，慢慢拼湊出一個簡單的 RBM 實做。

環境的設定上是使用 Python 配合 numpy 和 scipy 等 libraries。測試的資料則是用 mnist_all.mat [9] 當中的 train0 dataset，我們使用 784 個輸入，輸出則為 500 個。

這個模型中，我也是採用 binary 的模型（ $v_i, h_j \in \{0,1\}$ ），並且不實做像是 regularization 和 momentum 等等優化，只做簡單的學習。並且在每一個 iteration，我採用 [9] 的簡單公式，只做一輪迴的 Gibbs sampling：

$$p_j = sigmoid(hbias_j + \sum_{i=1}^{vnum} w_{ij} x_i)$$

$$y_j = \begin{cases} 1 & with\ \ probability\ \ p_j \\ 0 & otherwise \end{cases}$$

$$p_i = sigmoid(vbias_i + \sum_{j=1}^{hnum} w_{ij} y_j)$$

$$p'_j = sigmoid(hbias_j + \sum_{i=1}^{vnum} w_{ij} p_i)$$

然後再將結果用來更新模型的數值：

$$\Delta w_{ij} = \eta \frac{\sum_{k=1}^{m} (x_i p_j - p_i p'_j)}{m}$$

$$\Delta hbias_j = \eta \frac{\sum_{k=1}^{m} (p_j - p'_j)}{m}$$

$$\Delta vbias_i = \eta \frac{\sum_{k=1}^{m} (x_i - p_i)}{m}$$

最後寫完的簡單程式 rbm.py 附在最後，同時也在上傳的附檔中。

新發展

正如一開始所提到的，DNN 的研究非常熱門，因此進展想必也十分迅速。於是我挑選了[10] 來閱讀，檢視了近年的進展，結果真的有了許多新的發現。比如說，這篇文章提到，之前提過的 generative pre-training，如果在有大量 label 資料的情況下，其實不見得是必要的。除此之外，這篇文章也提出一些有趣的觀察，例如目前的 speaker-dependent DNNs 表現其實並沒有比 speaker-independent DNNs 好上太多，這情形跟上課時提到的，以前的作法 speaker-dependent 會好上很多完全不同。

在一開始的閱讀材料中，我常看到文中提到 DNN 相較於先前的系統有一個重大的限制，就是學習的演算法不容易平行化到多台機器上，如此一來，處理大量資料時就會非常緩慢。尤其在今日巨量資料的時代，scalability 成為一個相當重大的議題。

這篇文章中介紹了 Google 的 DistBelief [11] 系統，就是一個解決 scalibility 的嘗試。利用分散式的計算來完成 stochastic gradient descent 計算。他是將很多個 neural nets 拿給不同的機器針對不同的 training data 來計算 gradients，最後再想辦法整合起來。

這篇文章還提到了一些 recurrent neural networks 的新發展，還有是否可以用不一樣的 neuron activation function 來得到更好效果，以及在 DNNs 愈來愈成功的情況下，重新衡量什麼樣的語音預處理才是最好的，例如以 DNNs 結合 spectrograms 能夠得出勝過 MFCCs 為基礎的 recognizers 的結果。最後還有，DNNs 似乎對多語言辨識有很好的效果。

結語

機器在很多領域上都能輕鬆打敗人類，快速的數學計算使人類難以企及。然而也有一些事情是人類特別擅長的，在這當中，語言又是一個特別奇妙的能力。在地球上，似乎只有人類這個物種有如此複雜的語言，而且為了演化出這種能力，也花了好久的時間才達成。從小孩將 pidgins 轉換成 creoles 的能力可以看出人類學習語言不只是後天學習那麼簡單，還可能有一些能力是早就藏在大腦裡的。

透過觀察人類自身能學到的事情確實不少，研究人耳的知覺讓我們想出 mel-scale，而觀察人腦對資訊的處理，也啟發不同演算模型的研究。其中 Deep Neural Networks 確實是令人興奮的結果。

還記得以前在大學部時，曾聽別人說過 Artificial Neural Networks 是一個很多年都沒有進展的領域，直到現在聽老師的課，並看了資料後，才知道從 2006 年以來 deep learning 已經讓這領域蓬勃發展起來，值得好好研究。只是就像老師說的，每個時代最好的模型都不太一樣，未來 DNNs 也不見得會持續保持最好的結果。未來還得仔細關注發展，才能知道每個時代最尖端的科技是什麼。

回到語音系統，假設我們相信人的思考是純然的物理過程的話，我們就該相信，確實存在某種演算法，可以達成至少和人類做的一樣好的語音系統，因為那活生生的人腦正展示了語言的可能性。即使人腦使用了某種我們尚未可知的物理性質（例如量子物理的性質），我們也相信有一天我們會理解，並實作出可以使用同樣物理性質的系統。

而機器透過巨大的網路系統可以連結成千上萬的電腦系統，不眠不休針對特定工作不斷學習，這樣的學習所花的心血，或許終會有一天能趕上數萬年演化刻印在基因裡的記憶。會有那麼一天，機器的語音辨識能力將會超越人類。而當那樣的突破到來時，或許我們也會對自身更為了解吧。

參考資料

[1]. Yu, D., & Deng, L. (2011). Deep learning and its applications to signal and information processing [exploratory dsp]. Signal Processing Magazine, IEEE,28(1), 145-154.

[2]. Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A. R., Jaitly, N., ... & Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. Signal Processing Magazine, IEEE, 29(6), 82-97.

[3]. Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. Neural computation, 18(7), 1527-1554.

[4]. https://www.coursera.org/course/neuralnets

[5]. http://deeplearning.net/tutorial/rbm.html

[6]. https://gist.github.com/dwf/359323

[7]. http://pages.cs.wisc.edu/~andrzeje/research/rbm.pdf

[8]. http://halfbakedmaker.org/?p=11748

[9]. http://cs.nyu.edu/~roweis/data.html

[10]. Deng, L., Hinton, G., & Kingsbury, B. (2013, May). New types of deep neural network learning for speech recognition and related applications: An overview. In Proc. ICASSP.

[11]. Heigold, G., Vanhoucke, V., Senior, A., Nguyen, P., Ranzato, M., Devin, M., & Dean, J. (2013). Multilingual acoustic models using distributed deep neural networks. ICASSP.

rbm.py

```
import numpy
import scipy.io

class RBM(object):
    def __init__(self, vnum, hnum):
        self.vnum = vnum
        self.hnum = hnum

        self.weights = numpy.random.uniform(-0.01, 0.01, (vnum, hnum))
        self.vbias = numpy.zeros(vnum)
        self.hbias = numpy.zeros(hnum)

    def sample_binary(self, means):
        # samples[i] = 1 with prob: means[i]
        samples = numpy.zeros(means.shape)
        probs = numpy.random.uniform(size=means.shape)
        samples[probs < means] = 1.0
        return samples

    def ph(self, vis):
        ''' p(h|v) = sigmoid(hbias + Wv) '''
        return self.sigmoid(numpy.dot(vis, self.weights) + self.hbias)

    def pv(self, hid):
        ''' p(v|h) = sigmoid(vbias + W'h) '''
        return self.sigmoid(numpy.dot(hid, self.weights.T) + self.vbias)
```

```
    def sigmoid(self, x):
        ''' s(x) = 1/(1+e^(-x)) '''
        return 1/(1+numpy.exp(-x))


class Trainer(object):
    def __init__(self, rbm, rate=0.001):
        self.rbm = rbm
        self.rate = rate

    def train(self, training_data, iterations, batch_size=50):
        for it in range(iterations):
            for i in range(0, len(training_data), batch_size):
                x_i = training_data[i:i+batch_size]
                bsize = x_i.shape[0]

                # gibbs sampling
                p_j = self.rbm.ph(x_i)
                y_j = self.rbm.sample_binary(p_j)
                p_i = self.rbm.pv(y_j)
                p_j_ = self.rbm.ph(p_i)

                # update parameters
                self.rbm.weights += self.rate * (numpy.dot(x_i.T, p_j) - numpy.dot(p_i.T, p_j_)) / bsize
                self.rbm.hbias += self.rate * (p_j.sum(axis=0) - p_j_.sum(axis=0)) / bsize
                self.rbm.vbias += self.rate * ((x_i.sum(axis=0) - p_i.sum(axis=0)) / bsize)

            print 'Iteration #%d finished' % (it + 1)

class Tester(object):
    ISIZE, OSIZE = 784, 500

    def __init__(self):
        mfile = scipy.io.loadmat('mnist_all.mat')
        self.data = mfile['train0']
        self.rbm = RBM(Tester.ISIZE, Tester.OSIZE)
        self.trainer = Trainer(self.rbm)

    def test(self):
        self.trainer.train(self.data, 10)

if __name__ == '__main__':
    tester = Tester()
    tester.test()
```