การบ้านการเขียนโปรแกรม 4:
Neural Networks Learning

Thanks Andrew Ng for this beautiful programming exercise

ในการบ้านนี้เราจะทดลองเขียนโปรแกรมตามอัลกอริทึม backward propagation สำหรับสอน Neural networks โดยใช้ matrix operation ด้วย Octave/Matlab โดยทดลองทายตัวเลขอารบิกที่เขียนด้วยลายมือ เช่นเดียวกับการบ้านครั้งก่อน

ในpackage ประกอบด้วยไฟล์

- ex4.m - สคริปต์เพื่อรันโปรแกรมส่วนแรก
- ex4data1.mat - ชุดข้อมูลสำหรับสอนระบบส่วนแรก
- ex4weights.mat - ค่าน้ำหนักเริ่มต้นของ neural networks
- displayData.m - ฟังก์ชันเพื่อสร้างกราฟ
- fmincg.m - ฟังก์ชันเพื่อหาค่าพารามิเตอร์ที่ดีที่สุด (เช่นเดียวกับ fminunc())
- sigmoid.m - sigmoid function
- computeNumbericalGradient.m - สำหรับคำนวณ gradient แบบ numeric
- checkNNGradient.m - ฟังก์ชันสำหรับตรวจสอบ gradient
- debugInitializeWeights.m - ฟังก์ชันสำหรับกำหนดค่าเริ่มต้นของ weights
- predict.m - Neural networks Prediction Function
- sigmoidGradient.m* - ฟังก์ชันคำนวณ sigmoidgradient
- randInitializeWeights.m* - ฟังก์ชันทำหน้าที่สร้างการกำหนดค่าเริ่มต้นแบบสุ่ม
- nnCostFunction.m* - cost function ของ Neural Networks

* คือ ไฟล์ที่ต้องแก้ไขและส่ง

ตลอดการทดสอบโค้ดของการบ้านครั้งนี้ให้สั่งรัน ex4.m เท่านั้น

1. Neural Networks

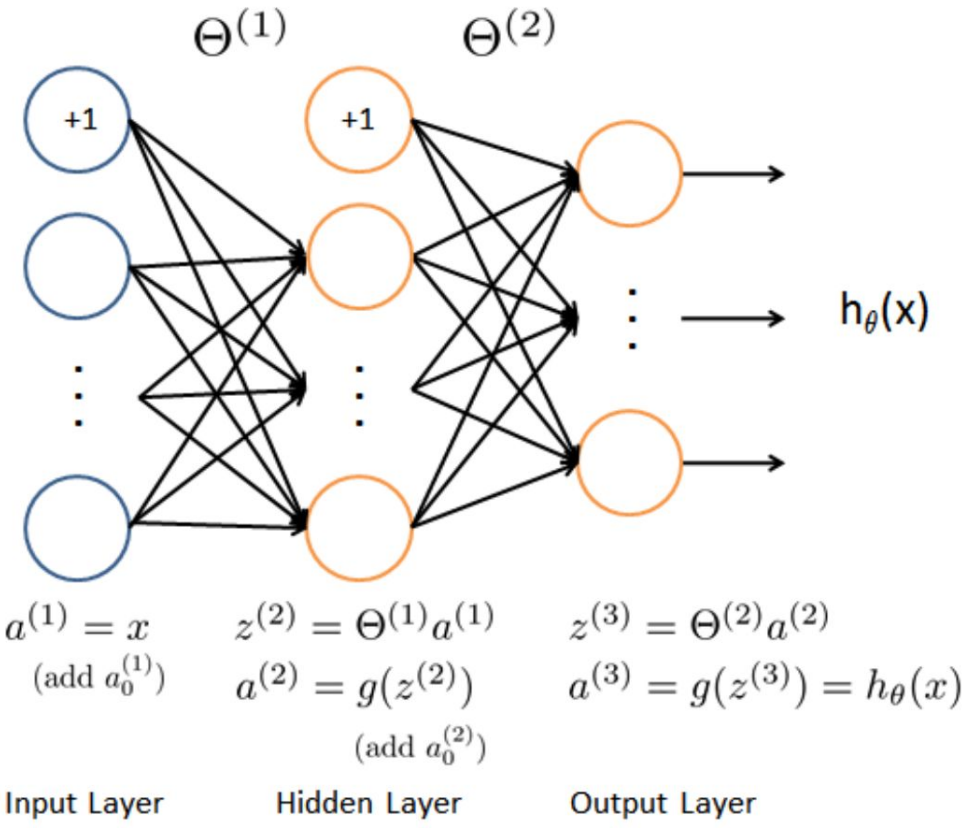
การบ้านนี้เน้นการเขียนตามอัลกอริทึม feedforward-backpropagation เพื่อสอน Neural Networks

1.1 แสดงผลภาพอินพุต

ในแพ็กเกจมีไฟล์ displayData.m เพื่อทำหน้าที่แสดงภาพอินพุต ซึ่งเป็นชุดข้อมูลเดียวกันกับการบ้านครั้งก่อน ซึ่งเป็นภาพขนาด 20 x 20 พิกเซล มีทั้งหมด 5000 ตัวอย่าง แต่ละอินพุตมีเอาท์พุตกำกับคือ ตัวเลข 0-9 โดย 0 คือ class ที่ 10 เพราะ Octave/Matlab เริ่มที่ 1

1.2 Model representation

โครงข่ายประสาทเทียมมีโครงสร้างดังภาพ โดยประกอบด้วย 3 ชั้น อินพุต เอาท์พุตและ hidden layer เนื่องจากแต่ละภาพมีขนาดเท่ากับเวกเตอร์ 400 (20x20) เป็นตัวกำหนดขนาดอินพุต และ ขนาดเอาท์พุตกำหนดด้วยเอาท์พุตมี 10 แบบ ส่วนชั้น hidden โจทย์ได้กำหนดมาให้เท่ากับ 25 โหนด



ภาพที่ 1 โครงข่ายประสาทเทียม

1.3 Feedforward and cost function

ในส่วนนี้ คุณต้องทำการแก้ไขโค้ดในไฟล์ nnCostFunction.m เพื่อคำนวณค่า cost ตามสมการ

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left[-y_k^{(i)} \log((h_\theta(x^{(i)}))_k) - (1 - y_k^{(i)}) \log(1 - (h_\theta(x^{(i)}))_k) \right]$$

โดย h(x) คำนวณได้ตามภาพที่ 1 , K =10 จำนวนแบบของผลลัพธ์ที่เป็นไปได้ อย่าลืมว่าเราต้องเปลี่ยนเอาท์พุตที่ได้มาให้อยู่ในรูปเวกเตอร์ของ y ดังภาพที่ 2

$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \dots \quad \text{or} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

ภาพที่ 2 เอาท์พุตเวกเตอร์

ค่า cost (J) ที่คำนวณได้ควรเป็นค่าผลรวม error ของทุกชุดข้อมูล โดยที่จำนวนข้อมูลจะเท่ากับเท่าไรโปรแกรมก็ยังทำงานได้ และจำนวนแบบของผลลัพธ์ (K) เช่นกัน

เมื่อสั่งรัน ex4 โปรแกรมควรแสดงค่า cost ที่คาดหมายคือ 0.287629

1.4 Regularized cost function

จากค่า cost ก่อนหน้า ในขั้นตอนนี้เราจะปรับค่าโดยเพิ่มขั้นตอน regularization ดังสมการ

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\left[\sum_{j=1}^{25}\sum_{k=1}^{400}(\Theta_{j,k}^{(1)})^2 + \sum_{j=1}^{10}\sum_{k=1}^{25}(\Theta_{j,k}^{(2)})^2\right].$$

ในขั้นตอนนี้ ให้เขียนโค้ดสำหรับโครงข่ายประสาทเทียมตามภาพที่ 1 คือมี 3 ชั้น โดยไม่ต้องรองรับโครงสร้างอื่นใด แต่อย่างไรก็ตาม โค้ดที่เขียนไม่ควรยึดติดกับขนาดของ Theta1, Theta2 ควรรองรับ Theta ได้ทุกขนาด

โค้ดนี้ให้แก้ไขใน nnCostFunction.m โดยเป็นการคำนวณเพิ่มจากค่า J เดิม เมื่อสั่งรัน ex4.m ควรได้ค่า cost จากการทำ regularization เป็น 0.383770

2. Backpropagation

ในขั้นตอนนี้ คุณต้องเขียนโค้ดเพื่อทำ parameter learning ตามอัลกอริทึม backpropagation ค่าที่ได้จากการคำนวณ คือ gradient ในตัวแปร grad เพื่อส่งต่อให้ฟังก์ชัน fmincg() ทำหน้าที่หาพารามิเตอร์ที่ดีที่สุด

โดยแบ่งเป็น 2 ขั้นตอนย่อยคือ คำนวณ gradient และตรวจสอบความถูกต้อง หลังจากนั้นจึงทำ regularized gradient

2.1 Sigmoid gradient

ฟังก์ชันย่อยที่จำเป็นต้องสร้างขึ้น เพื่อคำนวณ g'(z) ดังสมการ

$$g'(z) = \frac{d}{dz}g(z) = g(z)(1-g(z))$$

โดย g(z) คือ sigmoid function เดิม (sigmoid.m) ที่เคยทำในการบ้านครั้งก่อนๆ

ให้แก้ไขโค้ดลงในไฟล์ sigmoidGradient.m เพื่อหาค่า g'(z) โดยเรียกใช้ sigmoid() เมื่อทดลองเรียกใช้งานฟังก์ชันนี้ ที่หน้าจอ command line เช่น sigmoidGradient(0) ควรได้ค่าเท่ากับ 0.25

2.2 Random initialization

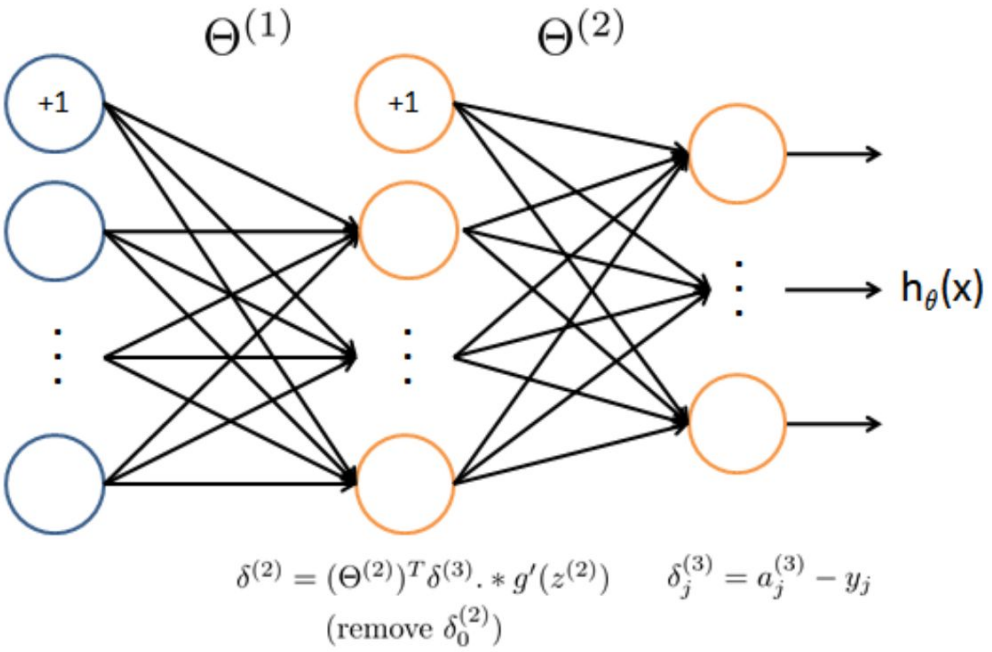
คุณต้องแก้ไขโค้ดในไฟล์ randInitializeWeights.m เพื่อกำหนดค่าเริ่มต้นแบบสุ่ม โดยพิมพ์โค้ดตามภาพที่ 3 ลงไป

```
% Randomly initialize the weights to small values
epsilon_init = 0.12;
W = rand(L_out, 1 + L_in) * 2 * epsilon_init - epsilon_init;
```

ภาพที่ 3 โค้ดเพื่อสุ่มค่าเริ่มต้น

2.3 Backpropagation

ในขั้นตอนนี้คุณต้องเขียนโค้ดเพื่อทำตามอัลกอริทึม backpropagation โดยเขียนเพิ่มลงใน nnCostFunction.m ดูภาพขั้นตอนจากขวามาซ้ายได้ในภาพที่ 4



ภาพที่ 4 การอัพเดทค่าจาก backpropagation

backpropagation ตามหลักการคือ คำนวณความผิดพลาด (error) ที่เกิดขึ้นในแต่ละโหนด โดยเริ่มจากชั้นเอาท์พุต ค่าผิดพลาดเขียนแทนด้วย $\delta_j^{(l)}$ อัลกอริทึมนี้จะรันลูปไปทุกชุดข้อมูล $(x^{(i)},y^{(i)})$ ซึ่งประกอบด้วย 4 ขั้นตอนดังนี้

1) กำหนดค่าอินพุต a(1) = x และรัน feedforward ไปทุกชั้น อย่าลืมเติม bias unit a_0 ให้กับทุก a ด้วย

2) สำหรับแต่ละเอาท์พุตในชั้นเอาท์พุต ให้คำนวณ error จากสมการ

$$\delta_k^{(3)} = (a_k^{(3)} - y_k),$$

3) สำหรับ hidden layer l=2 หาค่า error ได้จาก

$$\delta^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)} .* g'(z^{(2)})$$

4) รวมค่า gradient ของแต่ละชั้นได้จาก

$$\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$$

อย่าลืม เอา $\delta_0^{(2)}$ ออก

หลังรวมทุกชุดข้อมูลหาค่าผลรวม unregularized gradient ได้จาก

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)}$$

2.4 Gradient checking

ในส่วนนี้ เราเขียนฟังก์ชันมาให้คุณแล้วในไฟล์ computeNumericalGradient.m คุณไม่ต้องทำอะไร โค้ดส่วนนี้มีไว้เพื่อตรวจสอบผลลัพธ์ค่า gradient จากโค้ดของคุณกับค่าที่ถูกต้อง ควรมีความแตกต่างน้อยกว่า 1e-9

2.5 Regularized Neural Networks

เมื่อคุณคำนวณค่า gradient ตามอัลกอริทึม backpropagation เรียบร้อย ให้คำนวณ regularized เพิ่มลงไปด้วย โดยสามารถคำนวณแยกเทอมเฉพาะ regularization และนำค่าไปบวก ค่า regularization คำนวณได้ตามสมการ

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} \qquad \text{for } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)} + \frac{\lambda}{m} \Theta_{ij}^{(l)} \qquad \text{for } j \geq 1$$

อย่าลืมว่า การทำ regularization นี้เราจะทำเฉพาะ j ≥ 1 ซึ่งก็คือ คอลัมน์แรกของ theta (l) ซึ่งก็คือค่า bias
ถ้าโค้ดที่คุณเขียนถูกต้อง จะพบว่า relative difference มีค่าน้อยกว่า 1e-9

2.6 Learning parameters โดยใช้ fmincg

ในการบ้านนี้เราใช้ fmincg() ที่มีให้เพื่อทำ optimization การเรียกใช้งาน fmincg เราได้เขียนมาให้แล้ว ถ้าทุกขั้นตอนถูกต้อง คุณควรเห็นรายงานความถูกต้องของการทำนายบนชุดข้อมูลได้ 95.3%
คุณอาจลองเปลี่ยนจำนวนรอบดู เพื่อดูว่าค่าความถูกต้องเพิ่มขึ้นหรือไม่ (ตัวแปร MaxIter) แต่การตรวจการบ้านไม่ได้ขึ้นกับผลความถูกต้องของการทดลองเพิ่มรอบการรันนี้

ตารางคะแนน

ที่	งานที่ต้องทำ	ไฟล์ที่ต้องส่ง	คะแนน
1 2	Feedforward and Cost Function Regularized Cost Function	nnCostFunction.m nnCostFunction.m	30 15
3	Sigmoid Gradient	sigmoidGradient.m	5
4	Random Initialization	randInitializeWeights.m	5
5 6	Neural networks Gradient Regularized Gradient	nnCostFunction.m nnCostFunction.m	30 15
	คะแนนรวม		100