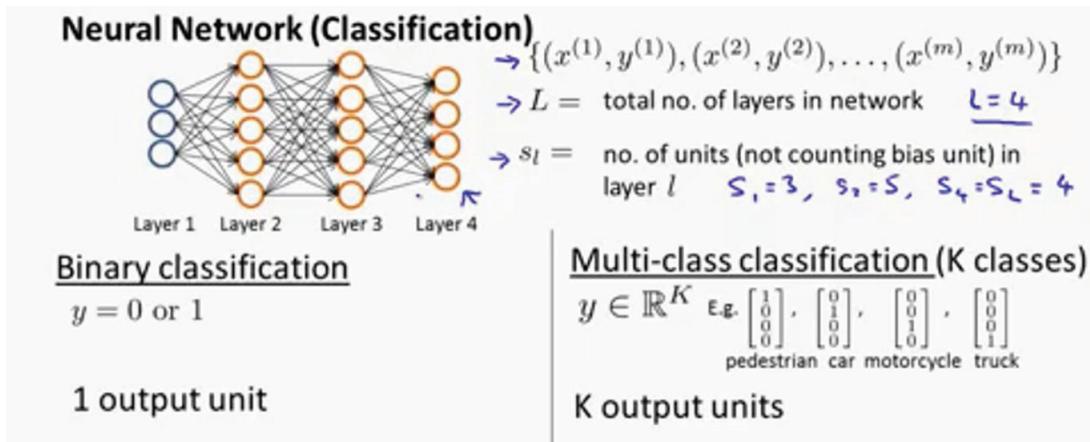


الأسبوع الخامس

Neural Networks: Learning

- التصنيف Classification
- تستخدم الـ NN اولا في التصنيف , و يكون التصنيف واحدا من اثنين
 - اما ان يكون binary يعني 1 او 0 , وهو يكون المخرج يا اه يا لا , مريض او غير مريض , العميل هيشترى او لا , العربية هتكلم ولا هتبظ , زي الرسم الایسر
 - يكون التقسيم لاكثر من حاجة , ياترى اللي في الصورة عربية ولا عجلة ولا كلب ولا سلحفة , يا ترى العميل هيشترى ولا ياجر ولا يسال ولا يشتكى , زي الرسم الایمن



- وللحصول على معادلة التكلفة cost function هنفكّر معادلة الـ Regression القديمة اللي كانت كدة

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

- بس بعد ما هنطورها ه تكون كدة

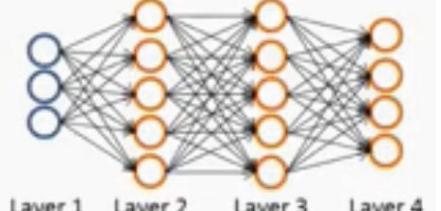
$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K \left[y_k^{(i)} \log((h_\Theta(x^{(i)}))_k) + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k) \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{j,i}^{(l)})^2$$

- هناقي في الجزء الاول من القانون سالب 1 على m اللي هي عدد الصفوف , بعدها اتنين سميشن , الاول لـ m اللي هي عدد الصفوف , والثاني لـ k اللي هو عدد وحدات المخرج (او بیناري بیقی 2 لو اکتر بیقی العدد ده) , بعدها قوانین ليوغاریتمات

- متساش ان الرقم اللي فوق في اي عنصر يدل على رقم الطبقة , واللي تحت على ترتيب انهي عنصر في الطبقة ,

معني كدة ان $y_k^{(i)}$ معناها الطبقة i و ترتيبها k

- الجزء الثاني الخاص بالـ regulation اللي هو بيعمل تعليم و تظبيط للرسم و القيم ، وهو لمدا على ضعف الـ m مضروبة في 3 سميشن
- الاول من 1 لـ L ناقص واحد ، الـ L هو عدد الطبقات ، يعني السميشن هتبدا من 1 لـ 3 لو عندنا 4 طبقات ،
 - وهتلاقى ان الثيتا $\Theta_{j,i}^{(l)}$ مكتوبة كدة ، و بالتالي رقم L هيكون فوق وهو فعلا اللي بيحدد رقم الطبقة
 - الثاني من 1 لـ SL وهو خاص بعدد الوحدات في الطبقة الاخيرة (4 في الرسمة فوق) ، وده بتتوافق مع اللي قلناه اننا بنعتمد علي عدد المخرجات المتاحة عندنا
 - الثالث من 1 لقيمة SL مجموع عليها 1 ، يعني لو الرسمة اللي فوق بيقي من 1 لـ 5
 - متتساش ان الثيتات متربعة

- طيب لو عايزيين نقل قيمة الكوست J لأقل قدر ممكن ، فنهنضطر نعمل تقاضل للمعادلة الكبيرة اللي فوق ، وه تكون تقاضلات جزئية partial بالنسبة للثيتات
 - ولو افترضنا اننا دخلنا معلومة واحدة x و هنخرج معلومة واحدة y فهتطون المعادلة كالتالي :
$$\begin{aligned} a^{(1)} &= x \\ z^{(2)} &= \Theta^{(1)} a^{(1)} \\ a^{(2)} &= g(z^{(2)}) \quad (\text{add } a_0^{(2)}) \\ z^{(3)} &= \Theta^{(2)} a^{(2)} \\ a^{(3)} &= g(z^{(3)}) \quad (\text{add } a_0^{(3)}) \\ z^{(4)} &= \Theta^{(3)} a^{(3)} \\ a^{(4)} &= h_\Theta(x) = g(z^{(4)}) \end{aligned}$$

- هنشوف ان اولا $a1$ هي قيمة الاكس الداخلة فه تكون قيمة $z2$ هي ثيتا 1 في $a1$ (متتساش ان الـ z كنا برمز لها بقيمة الـ a في الثيتا)
- قيمة $a2$ هي لما ادخل قيمة الـ z في الفنكشن g (اللي هي دالة السيجمويد)
- الناتج $a2$ لما اصربه في ثيتا 2 هيدينني $z3$ واللي هتوصلني لـ $a3$ لما ادخلها في السيجمويد تاني
- اخيرا قيمة $a3$ اصربه في ثيتا 3 ، و ادخلها السيجمويد هتيدينني $a4$ وهو اصل المخرج
- طبعا المسار ده لاربع طبقات بس ، لو اكتر هتريد
- والمسار ده لو مدخل واحد ، لو اكتر الموضوع هيكون اعقد
- لكن لمزيد من الدقة هنسخدم تكنيك يسمى Backpropagation Algorithm
 - التكينيك ده ، بيستخدم عشان اعمل تعديل في قيم ثيتات ، بناء علي الـ feedback الراجع من

- و ده معناه ببساطة ، اننا مش مجرد هنحسب قيمة الـ a_4 اللي هو المخرج النهائي و خلاص ، لكن هنقارنها بقيمة y اللي هي الناتج الحقيقي اللي جايبينه من الـ training data
- يعني لو انا عندي training data عن بيوت ، المدخل فيها (مساحة ، عدد الغرف ، الموقع ، الدهان) و المخرج هو (السعر) ، فهدخل المدخلات x_1, x_2, x_3, x_4 و هشوف المخرج اللي جايلي من الـ NN وقارنه بالسعر الحقيقي الموجود في الـ training data
- و هنا نستخدم العنصر ديلتا ، اللي هو الفرق بين القيمة الحقيقية والمتنوعة

$$\delta^{(L)} = a^{(L)} - y^{(t)}$$

- والكلام يطبق على اي طبقة (رمز L) فالديلتا 4 (الطبقة الاخيره) هيساوي a_4 (الرقم الاخير الناتج من الـ NN ناقص y (القيمة الحقيقة لسعر البيت)
- خد بالك ان قيم ديلتا و a و y مش ارقام و خلاص ، لكن فيكتور (مصفوفة عمود واحد في صفوف كتير) و عدد الصفوف هو نفسه عدد الوحدات في الطبقة الاخيره
- فالاول هنجيب ديلتا 4 ، من بيانات الطبقة الرابعة والواي ، ومنها هجيب ديلتا 3 بالقانون ده

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} * g'(z^{(3)})$$

- هتلافي ان ديلتا 3 ، معتمدة علي قيمة ديلتا 4 ، وثيتا 3 و z_3 ، وبالمثل ديلتا 2 هتكون كده

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

- اول حاجة $(\Theta^{(3)})^T$ هو فيكتور ثيتا 3 معموله ترانزبوس
- ديلتا 4 هي القيمة السابق ايجادها من المعادلة اللي قبلها
- الضرب هنا $(* .)$ اللي هو ضرب كل عنصر في اخوه ، مش ضرب مصفوفات
- الـ g هي تفاضل الـ g يعني السigmoid ، ولما هفاضل سigmoid الـ z_3 هتبقي كده $(1-a^{(3)}) * a^{(3)}$ وخد بالك كل ديه فيكتورات ، والـ 1 ده فيكتور وحابد
- خد بالك ان مفيش حاجة اسمها ديلتا 1 ، لأن اي ديلتا بتجيip قيمة الفرق عشان نرجعها اللي قبلها ، ولو جبنا ديلتا 1 هنرجعها لمين
- فكدة تبقي الصيغة العامة للديلتا هي :

$$\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) * a^{(l)} * (1 - a^{(l)})$$

- يعني ديلتا اي طبقة ، هي ثيتا لنفس الطبقة ترانزبوس ، مضروبة في ديلتا الطبقة اللي بعدها ، بعدها اضرب قيم المصفوفة ديه ، في a نفس الطبقة ، في وحابد ناقص a نفس الطبقة
- طبعا مع مراعاه ، ان القانون ده يتطبق بداية من الطبقة قبل الاخيره للطبقة الثانية $(\delta^{(2)}, \delta^{(L-2)}, \dots, \delta^{(L-1)})$ ، لأن لازم ديلتا الاخيره تكون موجودة في القانون ، ديلتا الاخيره مش هعملها بده طبعا ، لكن بالقانون بتاعها

$$\delta^{(L)} = a^{(L)} - y^{(t)}$$

- دلوقتي بقى هنحسب مجموع قيم ديلتا التراكمية :
 - هنعمل رمز جديد اسمه ديلتا مثلث ، وهي مجموع قيم ديلتا على بعض , و هيكون بالقانون :
- $$\Delta_{i,j}^{(l)} := \Delta_{i,j}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$$
- وده لو كان قيم مفردة ، لكن لو كان فيكتور هيكون :
- $$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$$
- بحيث ان الديلتا المثلث ، هتترايد فيها قيم كل دلتا صغيرة في a السابق لها
- وأخيرا هنحسب قيمة D من قيمة دلتا المثلث اللي اتحسبت حالا ، واللي تعتبر قيمة regulation اللي هي لتنظيم قيمة المخرجات ، وتكون بالمعادلة ديه

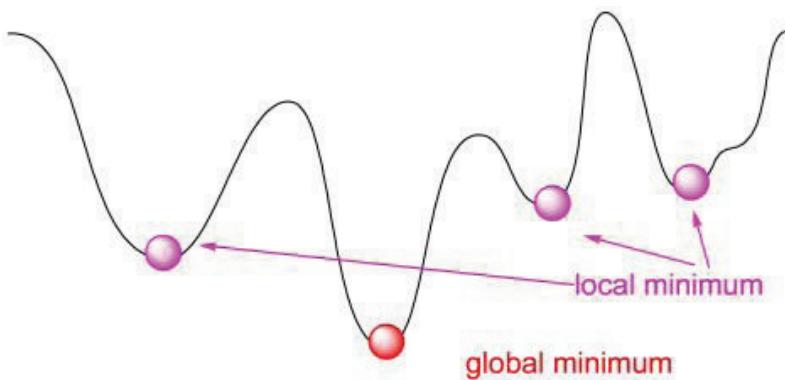
$$D_{i,j}^{(l)} := \frac{1}{m} \left(\Delta_{i,j}^{(l)} + \lambda \Theta_{i,j}^{(l)} \right), \text{ if } j \neq 0.$$

$$D_{i,j}^{(l)} := \frac{1}{m} \Delta_{i,j}^{(l)} \text{ if } j = 0$$

- فقيمة D معتمدة علي ديلتا مثلث و ثيتا ولما ، وفي حالة ان J تساوي صفر فهنجلي التيرم اللي فيه لمدا
- وبالحساب التفصيلي هناقلي ان قيمة D هي نفسها قيمة تفاضل J

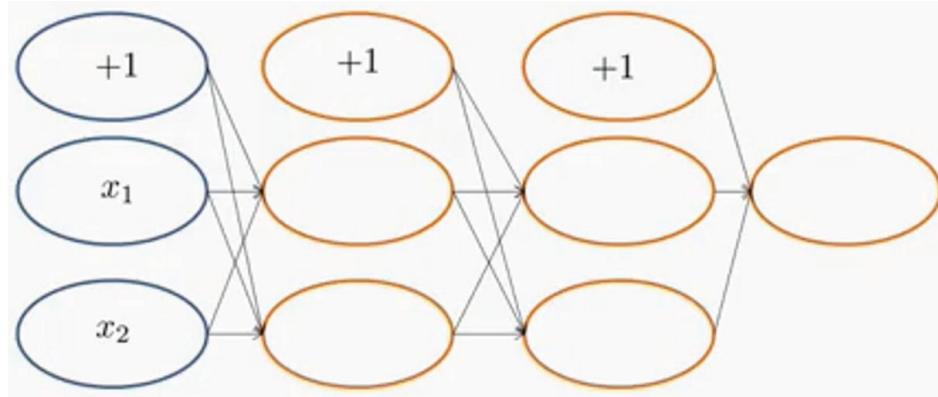
$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

- وخد بالك ان قيم J في الـ NN تكون non-convex يعني ممكن يكون فيه لوكال و جلوبال مينيم ، وده يخلي فيه مخاطرة ان اللوب تقع و تتوقف وتـ stuck في لوكال مش جلوبال ، لكن ده نادر جدا

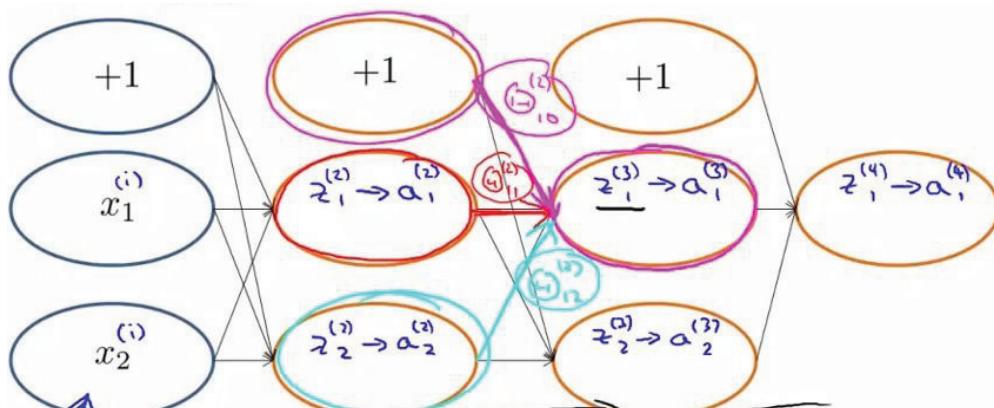


- وعشان نفهم الـ Backpropagation كويس ، تعالي نفهم ايه اللي بيحصل في الـ forward propagation الخلية البسيطة ديه

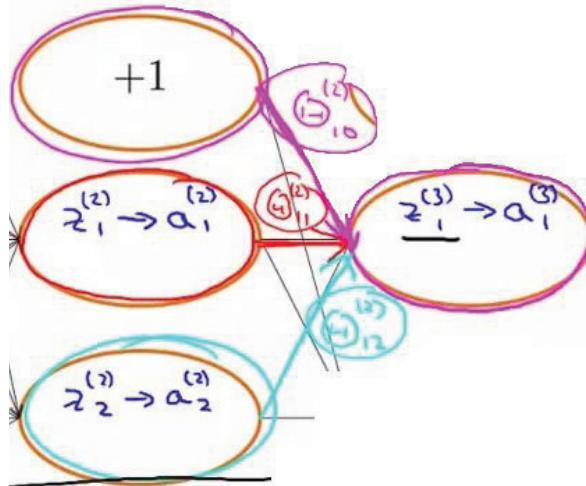
- الخلية ديه تحتوي علي اتنين مدخل (غير بتاع الـ bias) و طبقتين مخفيتين ، و مخرج واحد



- بما ان الاكسات هي المدخلات ، فهنقول ان المدخلين هما $x_1^{(i)}$ و $x_2^{(i)}$
- في الطبقة الثانية ، حاصل ضرب الثيتات في الاكسات هو الـ $Z^{(2)}$ فهناقي ان في الطبقة الثانية $Z^{(2)}$ هتكون مجموع ضرب الثيتات في الاكسات من الطبقة الاولى ، وهكذا
- قيمة $a_1^{(2)}$ ه تكون حالة السigmoid لما ادخل فيها $Z^{(2)}$
- بنفس الطريقة ز ي $a_3^{(2)}$ ه تبقى سigmoid قيمة $Z_3^{(2)}$ و اخيرا $a_4^{(2)}$ ه تبقى سigmoid قيمة $Z_4^{(2)}$
- وديه الرسمة المجمعة



- وعشان نفهمها كويس ، بص معايا على القيمة $a_1^{(3)}$ ، هناقي ان مدخلاتها هي :



- قيمة $a_1^{(3)}$ هي تكون حاصل ضرب ثيتا اتنين (10) (يعني رايح من العنصر 0 ومتوجه الي العنصر 1) ، مضروبة في 1 اللي معمولة باللون بنفسجي، زائد ثيتا اتنين (11) (يعني رايح من العنصر 1 ومتوجه الي العنصر 1) مضروب في قيمة $a_1^{(2)}$ (يعني ايه في الطبقة الثانية العنصر الاول) اللي باللون الاحمر زائد ثيتا اتنين (12) (يعني رايح من العنصر 2 ومتوجه الي العنصر 1) مضروب في قيمة $a_2^{(2)}$ (يعني ايه في الطبقة الثانية العنصر الثاني) اللي باللون اللبناني

- وفي النهاية تكون المعادلة كدة

$$z_1^{(3)} = \Theta_{10}^{(2)} \cdot 1 + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} \cdot a_2^{(2)}$$

• طيب ماذا عن الـ Backpropagation

- عشان نفهمها كويس ، تعالي نبص علي معادلة التعامل مع الـ Backpropagation اللي بتكون كدة

$$J(\Theta) = -\frac{1}{m} \sum_{t=1}^m \sum_{k=1}^K \left[y_k^{(t)} \log(h_\Theta(x^{(t)}))_k + (1 - y_k^{(t)}) \log(1 - h_\Theta(x^{(t)}))_k \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{j,i}^{(l)})^2$$

- هنفترض ان فيه مدخل واحد فمش هنحتاج سمشن ، و هنتجاهل مؤقتا اللمنا ، فهناكى ان المعادلة تم اختصارها لده :

$$cost(t) = y^{(t)} \log(h_\Theta(x^{(t)})) + (1 - y^{(t)}) \log(1 - h_\Theta(x^{(t)}))$$

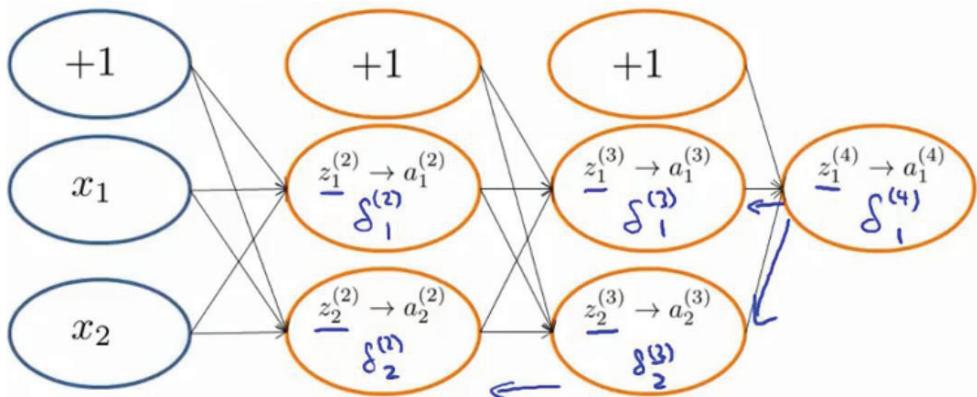
- وحتى المعادلة دي ممكن نختصرها و تكون كدة

$$cost(i) \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$$

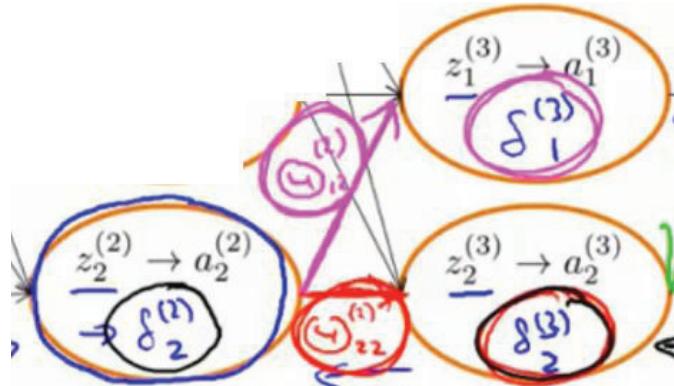
- فمعنى كدة ان التكلفة هي تقريبا مربع الفرق بين الـ H (القيمة المتوقعة) والـ Y (القيمة الحقيقية)

- تعالى بقى نتتبع الـ Backpropagation عشان نعرف الديلتا بيتجي ازاى

○



- معروف ان اول ديلتا علي اليمين (دليتا 1-4) (اللي هي في الطبقة الرابعة يعني المخرج) هتساوي قيمة الـ y ناقص قيمة a_4 اللي هي المخرج
- من (دليتا 1-4) هنقدر نجيب قيم (دليتا 3-1)(طبقة الثالثة العنصر الاول) و (دليتا 3-2) (طبقة الثالثة العنصر الثاني)
- كمان (دليتا 1-2) (طبقة الثانية العنصر الاول) هنجيب قيمها من كل من (دليتا 1-3) و (دليتا 2-3)
- وبرضه (دليتا 2-2) (طبقة الثانية العنصر الثاني) هنجيب قيمها من كل من (دليتا 1-3) و (دليتا 2-3)
- تعالى نبص على الدايرة اللي فيها (دليتا 2-2)



- هتلافق ان قيمة (دليتا 2-2) هي اصلا بتتجي من قيمتين الديلتا اللي بعدها , اللي هي باللون البني و الاحمر , وه تكون مضروبة في ثيانتها

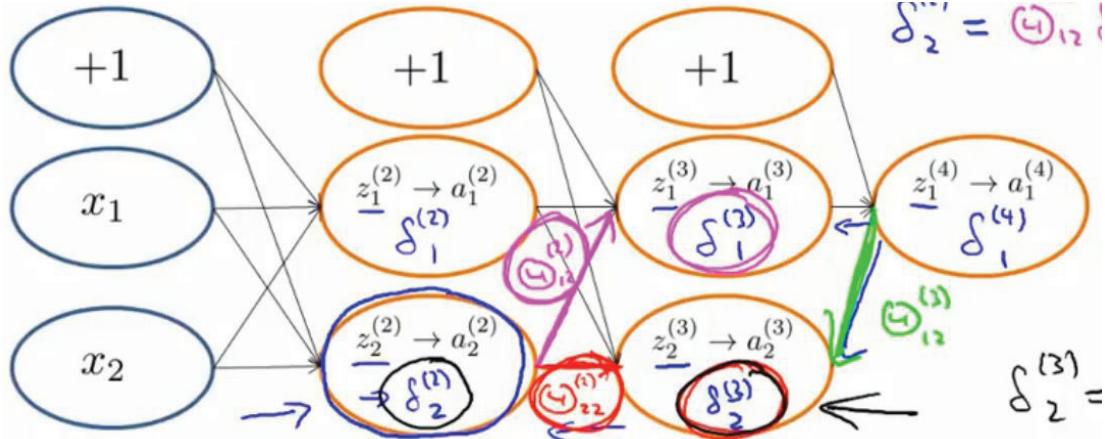
$$\delta_2^{(2)} = \textcolor{brown}{\delta_{12}^{(1)}} \delta_1^{(3)} + \textcolor{red}{\delta_{22}^{(1)}} \delta_2^{(3)}$$

- كمان هتللحظ ان قيمة (دليتا 3-2) هي حاصل ضرب ديلتا الي بعدها (دليتا 1-4) في ثيانتها

$$\delta^{(3)}_2 = \Theta_{12}^{(3)} \cdot \delta^{(4)}_1.$$

و يكون الشكل المجمع كدة

$$\delta^{(3)}_2 = \Theta_{12}^{(3)} \cdot \delta^{(4)}_1$$



و ده اللي بيشرح باختصار تكنيك الـ Backpropagation

وان كان الكلام ده يشرح فهمه او تخيله فتطبيقه سهل الي حد ما

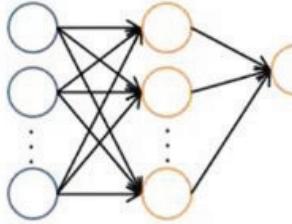
• طيب لو افتكروا الكود اللي كنا بنجيب فيه معادلة الكوست ، كان كالتالي :

```
function [jVal, gradient] = costFunction(theta)
...
optTheta = fminunc(@costFunction, initialTheta, options)

Neural Network (L=4):
    Theta(1), Theta(2), Theta(3) - matrices (Theta1, Theta2, Theta3)
    D(1), D(2), D(3) - matrices (D1, D2, D3)
```

لكن كنا قبل كدة بنتعامل مع كل الحاجات ديه علي اساس انها فيكتور . حاليا احنا مضطرين اننا نحولها من مصفوفات لفيكتور ، باعتبار ان الـ NN قائمة علي المصفوفات مش الفيكتور ، بينما المعادلة ديه بصيغة الفيكتور

هناقي ان لو عندنا خلية NN فيها 10 مدخلات ، وطبقة خفية 10 عناص ، وخرج واحد زي كدة



وقتها عدد العناصر بالترتيب ، هيكون $10 \times 10 \times 10 = 1000$

$$s_1 = 10, s_2 = 10, s_3 = 1$$

- ساعتها ثيتا 1 (العوامل اللي بتتضرب في قيم اكس 1) ه تكون مصفوفة 10x11 ، لأن عندي 11 عنصر في الطبقة الاولى (الـ 10 زائد الواحد بتات bias) وكل عنصر فيهم عامل اسمهم للعشر عناصر في الطبقة الثانية ، فه تكون 10 في 11 ، يعني 110 قيمة
- كذلك الامر للثيتا 2 من الطبقة الثانية للتالتة، ه تكون 10 في 11
- اما الثيتا الاخيرة ، فه تكون بس 11 قيمة ، لأن عندي 11 $(10 + 1)$ عنصر في الطبقة قبل الاخيرة ، واللي تحتاج 11 ثيتا

$$\Theta^{(1)} \in \mathbb{R}^{10 \times 11}, \Theta^{(2)} \in \mathbb{R}^{10 \times 11}, \Theta^{(3)} \in \mathbb{R}^{1 \times 11}$$

وبالنسبة لقيم الـ D ه تكون بنفس المقاسات بالضبط لأنها معتمدة على الثيتا

$$D^{(1)} \in \mathbb{R}^{10 \times 11}, D^{(2)} \in \mathbb{R}^{10 \times 11}, D^{(3)} \in \mathbb{R}^{1 \times 11}$$

متتساش ان معادلة الـ D اهي :

$$D_{i,j}^{(l)} := \frac{1}{m} \left(\Delta_{i,j}^{(l)} + \lambda \Theta_{i,j}^{(l)} \right), \text{ if } j \neq 0.$$

$$D_{i,j}^{(l)} := \frac{1}{m} \Delta_{i,j}^{(l)} \text{ if } j = 0$$

- طيب المطلوب حاليا ، تحويل كل المصفوفات ديه فيكتور ، عشان اعرف ادخلها في الدالة اللي فوق ، فهستخدم الكود ده لاوكтив :

```
thetaVector = [ Theta1(); Theta2(); Theta3(); ]
```

```
deltaVector = [ D1(); D2(); D3(); ]
```

- واللي هيدخل كل قيم المصفوفات بالترتيب في فيكتورات
- وبعد ما اخلص الحساب ، لو عايز ارجع اجيب القيم من الفيكتورات و احوله لمصفوفات بنفس المقاس ، استخدم الامور ده

```
Theta1 = reshape(thetaVector(1:110),10,11)
```

```
Theta2 = reshape(thetaVector(111:220),10,11)
```

```
Theta3 = reshape(thetaVector(221:231),1,11)
```

- واللي هيختار من الفيكتورات ، قيم بالترتيب لرصها في المصفوفات
 - وطبعا لاحظ ان الارقام 110 , 111 , 220 , وهكذا ، معتمدة على ابعاد المصفوفات الاصلية ، فهتتغير لو
الابعاد مختلفة
-

● تطبيق عملي :

- في اوكتيف ، اكتب معايا ده :

```
>> theta1 = ones(10,11)
```

- هي عملك مصفوفة وحيد 10 في 11

```
theta1 =
```

```
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1
```

- بعدها اكتب ده :

```
>> theta2 = 2* ones(10,11)
```

- هي عملك مصفوفة اتنينات 10 في 11

```
theta2 =
```

```
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
```

```
2 2 2 2 2 2 2 2 2 2  
2 2 2 2 2 2 2 2 2 2  
2 2 2 2 2 2 2 2 2 2  
2 2 2 2 2 2 2 2 2 2  
2 2 2 2 2 2 2 2 2 2
```

- بعدها اعمل ثيتا 3

```
>> theta3 = 3*ones(1,11)
```

- هي عملك مصفوفة 1 في 11

```
theta3 =
```

```
3 3 3 3 3 3 3 3 3 3
```

- دلوقتي قوله يجمعهم مع بعض

```
>> thetaVector = [ theta1(:); theta2(:); theta3(:); ]
```

- هي عمل فيكتور طويل جدا فيه كل الارقام

- ولو سالته المقياس كام هيقولك 231 صف في عمود واحد

```
>> size(thetaVector)
```

```
ans =
```

```
231 1
```

- دلوقتي لو جينا قناله اعمل مصفوفة وحط فيها ارقم ثيتا واحد هي عملها

```
>> NewTheta1 = reshape(thetaVector(1:110),10,11)
```

- كذلك الحال لاتنين و ثلاثة

```
>> NewTheta2 = reshape(thetaVector(111:220),10,11)
```

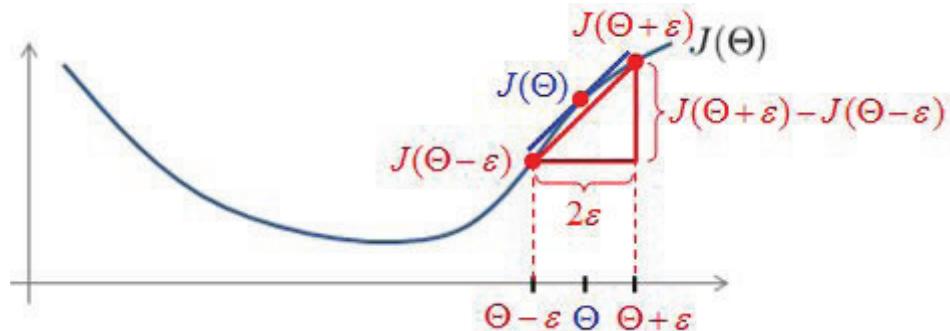
```
>> NewTheta3 = reshape(thetaVector(221:231),1,11)
```

- فالخطة اللي هامشي عليها كالتالي :

- أولاً هيكون معطى ليا قيم اولية للثيتا 1 و 2 و 3 ، همسكهم افكمه واحطهم في الفيكتور الطويل اللي اسمه initialtheta
 - بعدها هدخل initialtheta في دالة fminunc السابق شرحها ، واللى هتديني الثيتات النهائية بفيكتور بنفس طول initialtheta لكن هيكون اسمه thetavec
 - همسک thetavec و ارصله في مصفوفات ثيتا 1 و 2 و 3 تاني هاطبق FP , BP عشان اجيب L , D1 , D2 , D3
 - هفك قيم gradientvec عشان اجيب الفيكتور النهائي D1 , D2 , D3
-

• الفحص التدريجي Gradient Checking

- لما بدأوا يطبقو تكنيك NN سواء الـ BP او الـ FP وجدوا ان احياناً يتم انتاج انواع من المشاكل و الـ Bugs اللي بتلغبط الحساب و بتخلي القيم غير سليمة ، مع إن الـ L شكلها بيق و كانها سليمة ، لكن النتائج المتوقعة غير دقيقة في الآخر
- فبدوا في استخدام تكنيك Gradient Checking اللي بيقضى بالكامل على كل الـ Bugs ديه ، وده اللي هنشوفه حالاً عشان نفهمها ، تعالى نشوف الرسمة :



- بفرض ان الجراف ده يمثل محور اكس فيه الثيتا ، ومحور واي الـ L ، فعشان اجيب تفاضل الـ L عند الثيتا ، هشوف المعنى الهندسي ليها وهو ميل الخط عند النقطة ديه فهافرض علي محور اكس نقطة قبل و نقطة بعد الثيتا بمقدار ابسلون ϵ ، معني كدة ان قيمة واي عندهم ه تكون $J(\theta + \epsilon)$ و $J(\theta - \epsilon)$

- و هيكون الميل نفسه يساوي $\frac{\partial}{\partial \theta} J(\theta) \approx \frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{2\epsilon}$ ، بس بشرط فرض ان ابلسون قيمة صغيرة (عشان النهايات تكون مطبوعة) ، فهنفرضها بـ 0.0001 ، ويفضل متعملهاش اكتر من كدة عشان النهايات متبوطة ، ولا اقل من كدة عشان القسمة متكونش علي صفر

$$\frac{J(\theta + \epsilon) - J(\theta)}{2\epsilon}$$

- مع مراعاة ان ممكن اعمل تفاضل باضافة الاسلون بس زي كدة واللي بنسميهها تفاضل من اتجاه واحد , بينما اللي فوق من اتجاهين , غالبا بفضل الاتجاهين لانه اكتر دقة
- دلوقتي ممكن اكتب امر لاوكتيف بالطريقة ديه :

`gradApprox = (J(thetaPlus) - J(thetaMinus))/(2*epsilon)`

• طيب عظيم , مازالو عندي ثيتات كتيرة :

- ساعتها قيمة L ه تكون معتمدة على اضافة اسلون بس علي الثيتا اللي ه عملها تفاضل , واسيب الباقي زي ما هو فلو ه اعمل تفاضل L للثيتا 1 , ه تكون قيمة L ثيتا 1 + اسلون , و ثيتا 2 وهكذا , ناقص L ثيتا 1 اسلون , و ثيتا 2 وهكذا علي اتنين اسلون
- ولو ه اعمل تفاضل L للثيتا 2 , ه تكون نفس الموضوع بس مع اضافة اسلون علي ثيتا 2

$$\begin{aligned}\frac{\partial}{\partial \theta_1} J(\theta) &\approx \frac{J(\theta_1 + \epsilon, \theta_2, \theta_3, \dots, \theta_n) - J(\theta_1 - \epsilon, \theta_2, \theta_3, \dots, \theta_n)}{2\epsilon} \\ \frac{\partial}{\partial \theta_2} J(\theta) &\approx \frac{J(\theta_1, \theta_2 + \epsilon, \theta_3, \dots, \theta_n) - J(\theta_1, \theta_2 - \epsilon, \theta_3, \dots, \theta_n)}{2\epsilon} \\ &\vdots \\ \frac{\partial}{\partial \theta_n} J(\theta) &\approx \frac{J(\theta_1, \theta_2, \theta_3, \dots, \theta_n + \epsilon) - J(\theta_1, \theta_2, \theta_3, \dots, \theta_n - \epsilon)}{2\epsilon}\end{aligned}$$

○ وبالتالي الصيغة العامة المستخدمة ممكن تكون كدة :

$$\frac{\partial}{\partial \Theta_j} J(\Theta) \approx \frac{J(\Theta_1, \dots, \Theta_j + \epsilon, \dots, \Theta_n) - J(\Theta_1, \dots, \Theta_j - \epsilon, \dots, \Theta_n)}{2\epsilon}$$

○ وهيقي الكود كدة :

```
epsilon = 1e-4;
for i = 1:n,
    thetaPlus = theta;
    thetaPlus(i) += epsilon;
    thetaMinus = theta;
    thetaMinus(i) -= epsilon;
    gradApprox(i) = (J(thetaPlus) - J(thetaMinus))/(2*epsilon)
end;
```

- الاول بيحدد ثيما اسلون بـ 0.0001
- دلوقتي بيعمل لووب فور , من الـ n (اللي هو عدد الثيتات)
- فيكتور ثيتا بلس هيكون نفسه فيكتور ثيتا

- لكن عشان ازود قيمة ثيتا المطلوبة في الفيكتور ، هقول ان `thetaPlus(i) += epsilon` والي معناها ان قيمة ثيتا بلس الخاصة بـ `i` (يعني مثلاً ثيتا 5 دلوقتي في الفيكتور) هتساوي نفس قيمتها مضافة اليها ابسلون و وبالتالي قيمة الفيكتور هتبقي كدة :

$$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_i + 4 \\ \vdots \\ \theta_n \end{bmatrix} \rightarrow \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_i - 4 \\ \vdots \\ \theta_n \end{bmatrix}$$

- كل قيم ثيتا زي ما هي ، ما عدا ثيتا `i`
- هيتم تكرار نفس الحور مع السالب
- اخيراً احسب قيمة الجرادينت ، بالمعادلة
- ولاحظ ان المفروض قيمة الـ `gradApprox` اللي هطلع من هنا تكون قريبة جداً من قيمة `D` اللي هطلع من الـ `BP` ، وده اللي هيخليني متاكد ان شغلي صح و مفيش لغبطة في المعادلات

• أما خطوات التعامل مع هذا التكنيك تكون بالترتيب :

1. طبق الـ `BP` عشان اجيب قيمة الـ `D` عن طريق الـ `unroll` اللي قلناه من شوية
2. هات قيمة الـ `gradApprox` بطريقة التفاضل اللي قلناها حالاً
3. اتأكد ان القيمتين تقريباً زي بعض، ساعتها نعرف ان مفيش bugs
4. دلوقتي لازم بقى تلغى عملية التفاضل بتاعت الـ `gradApprox` لأن هي بتعطل الكود جداً ، عشان بتاخذ وقت رهيب ، وبالتالي هي بتسخدم مرة واحدة للتأكد من صحة الخطوة و تلغى بعدها فوراً العدم ضياع الوقت

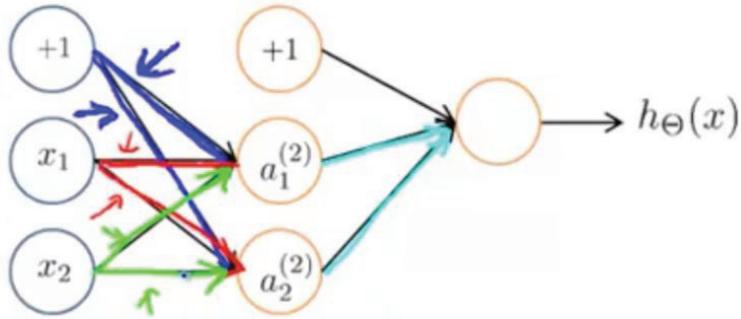
• طيب ماذَا عن قيم ثيتا اللي هبدأ بيها Gradient Descent

- لما كنا شغالين في الـ `logistic regression` كنا بنفرض قيم مبدئية لثيتا اصفار ، وده كان مفيهوش مشاكل ، لأن اللوب نفسها كانت بتضبط قيم ثيتات تدريجي لغاية لما تجيء قيم ثيتا اللي بتعمل اقل لـ
- بينما هنا مش هينفع نعمل قيم ثيتا بصفر ، ليه ؟
- لأن في الـ `NN` قيمة الطبقة الثانية بتعتمد على قيم ثيتا مضروبة في الاكسات ، زي ما شفنا في المعادلة القديمة

$$\begin{aligned} a_1^{(2)} &= g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3) \\ a_2^{(2)} &= g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3) \\ a_3^{(2)} &= g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3) \end{aligned}$$

- فتصير الثيتات ، هيخلி قيمة $\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3$ (اللي هي الـ Z) للاولي مساوية للثانية للثالثة ، وكلهم هيساواو صفر ، لما نعملها في السيمجوميد ، هنلاقي كل قيم الـ a بواحد ، وه تكون متساوية
- حتى بعد اول دورة ، لو قيم الثيتات بقت برقم مش صفر ، فهي ه تكون بارقام متساوية ، كل الثيتات بنفس الرقم ، فبرضه قيمة الـ Z هتساوي بعض ، و هتجي من السيمجوميد نفس القيمة ، فبرضه كل الـ a ه تكون هي هي ، وديه مشكلة مش هتتحل

- ولو اكتر من طبقة ، هنلاقي كل الـ a في كل طبقة علي حدة بنفس القيمة و ديه مشكلة فزي ما باین في الصورة قيم الثيتا ه تكون ثابتة في الخطوط الزرقاء ، وثابتة للاحمر و ثابتة للاخضر ، فه تكون قيم ايها الطبقة الثانية هي هي



• طب الحل ايه ؟

- اننا نعمل قيم عشوائية لمصفوفات ثيتا 1 و 2 ، ونخى كل القيم اكبر من سالب ابسلون ، واقل من ابسلون ، وده يتم بامر rand في اوكتيف ، بالشكل ده

Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\epsilon, \epsilon]$
(i.e. $-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon$)

E.g.

```
Theta1 = rand(10,11)*(2*INIT_EPSILON)
- INIT_EPSILON;
```

```
Theta2 = rand(1,11)*(2*INIT_EPSILON)
- INIT_EPSILON;
```

- اننا نشكل مصفوفات ثيتا 1 و 2 بالابعاد المطلوبة ، وبامر rand و نضربها في ضعف مصفوفة ابسلون ناقص مصفوفة ابسلون ، وده هيخليها رقم عشوائي تماما
- كذلك ثيتا 2 ه تكون نفس النظام بس ابعادها مختلفة
- خ بالك ان ابسلون ديه غير ابسلون اللي فاتت اللي كانت 0.0001
- و ده هيكون الكود

If the dimensions of Theta1 is 10x11, Theta2 is 10x11 and Theta3 is 1x11.

```
Theta1 = rand(10,11) * (2 * INIT_EPSILON) - INIT_EPSILON;
```

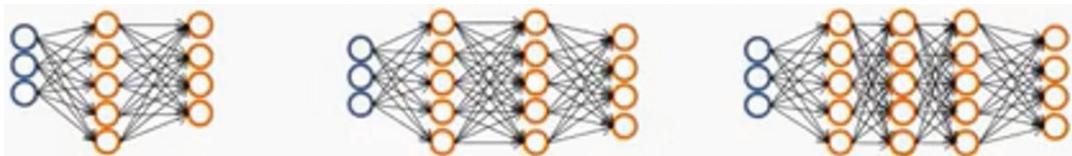
```
Theta2 = rand(10,11) * (2 * INIT_EPSILON) - INIT_EPSILON;
```

```
Theta3 = rand(1,11) * (2 * INIT_EPSILON) - INIT_EPSILON;
```

- بعد ما فهمنا خطوات الـ FP ، دلوقتي عايزيين نعمل خطوات واضحة و حدة لتنفيذها لتطبيق الكلام ده

• الخطوة الأولى :

- أولاً لازم نختار ما بين اي شكل او تصميم للـ NN ، فلو ان المسألة عندها فيها 3 مدخلات و اربع مخرجات ، فممكن نختار النموذج اليسير ، اللي فيه طبقة خفية 1 ، او النموذج الاوسط و اللي فيه طبقتين ، او اليمن اللي فيه 3 طبقات



- عدد المدخلات او المخرجات مش بيكون بمزاجنا ، لكن المدخلات هو عدد الـ features اللي هي x و المخرجات اللي هي classifications اللي هي الـ y يعني عندي كام تقسيمة مطلوبة

No. of input units: Dimension of features $x^{(i)}$

No. output units: Number of classes

- متنساش ان المخرجات بيتم تقسيمها كالتالي ، لو عندي المخرجات 10 مثلا ، ان الكاميرا هتصنف الصورة اما كلب او قط او جربوع او قنفد او او او ، فيتم عمل عشر مصفوفات فيكتور ، كل واحدة عبارة عن عمود واحد في 10 صفوف

- مصفوفة المخرج الاولى (كلب) هتكون كلها اصفار عدا اول رقم ، و مصفوفة المخرج الثاني ه تكون كلها اصفار عدا ثاني رقم وهكذا

$$\begin{matrix} \text{Dog} \\ \text{Cat} \\ \text{Bird} \\ \text{Cow} \\ \text{Horse} \\ \text{Sheep} \\ \text{Boat} \\ \text{Car} \\ \text{Train} \\ \text{Motorcycle} \end{matrix} \leftarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

- فيأغلب الاحيان ، بيكون عدد الطبقات الخفية 1 بس ، زي المثال اليسير ، بس كل ما يزيد عدد الطبقات الخفية كل ما تزيد الجودة ، ويزيد الوقت المطلوب لعملها و بالتالي التكلفة

- وطبعاً لو كان فيه اكتر من طبقة خفية ، فلازم عدد عناصر كل طبقة خفية تكون متساوية تماماً

- غالباً ما يكون عدد عناصر كل طبقة اكتر من عدد الـ features المدخلات ، مثلاً ضعفها

• الخطوة الثانية :

- اختيار قيم ثيتا بشكل عشوائي بقيم صغيرة تقترب من الصفر, زي ما شفنا من شوية

• الخطوة الثالثة :

- اعمل FP بحيث ادخل قيم اكسات مع ثيتات , واجيب قيم وايات

• الخطوة الرابعة :

- اجيب قيمة التكفة بتطبيق قانون الـ L

• الخطوة الخامسة:

$$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T \left| \frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta) \right| \text{ وقيمة المثلث}$$

- اعمل BP عشان اجيب قيم المشتقة

• الخطوة السادسة:

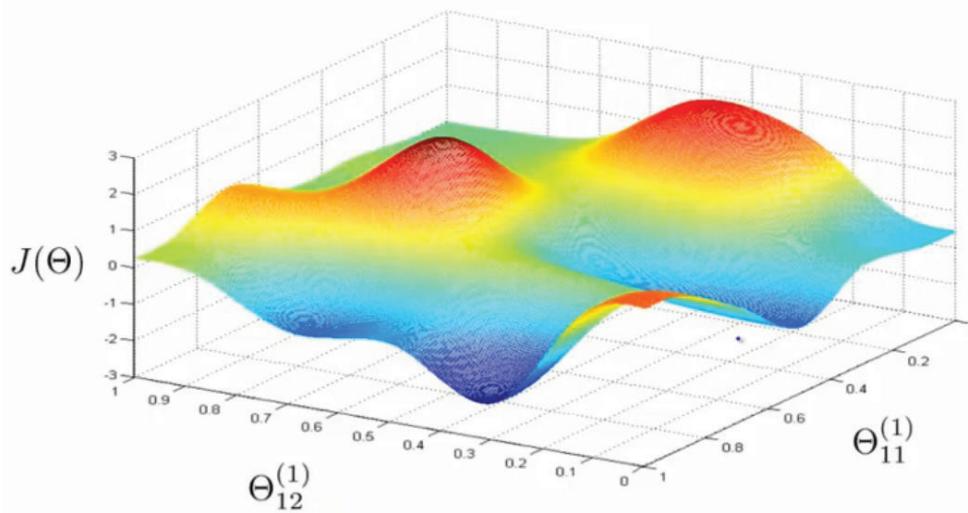
- قارن بين قيمة $\left| \frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta) \right|$ اللي جبتها من الـ BP وبين قيمة الجرادينت ل عشان تتأكد انهم بيساو بعض تقريبا
- متساش توقف الكود بتاع الفحص التدريجي اللي هو ده

$$\begin{aligned} \frac{\partial}{\partial \theta_1} J(\theta) &\approx \frac{J(\theta_1 + \epsilon, \theta_2, \theta_3, \dots, \theta_n) - J(\theta_1 - \epsilon, \theta_2, \theta_3, \dots, \theta_n)}{2\epsilon} \\ \frac{\partial}{\partial \theta_2} J(\theta) &\approx \frac{J(\theta_1, \theta_2 + \epsilon, \theta_3, \dots, \theta_n) - J(\theta_1, \theta_2 - \epsilon, \theta_3, \dots, \theta_n)}{2\epsilon} \\ &\vdots \\ \frac{\partial}{\partial \theta_n} J(\theta) &\approx \frac{J(\theta_1, \theta_2, \theta_3, \dots, \theta_n + \epsilon) - J(\theta_1, \theta_2, \theta_3, \dots, \theta_n - \epsilon)}{2\epsilon} \end{aligned}$$

• الخطوة السابعة:

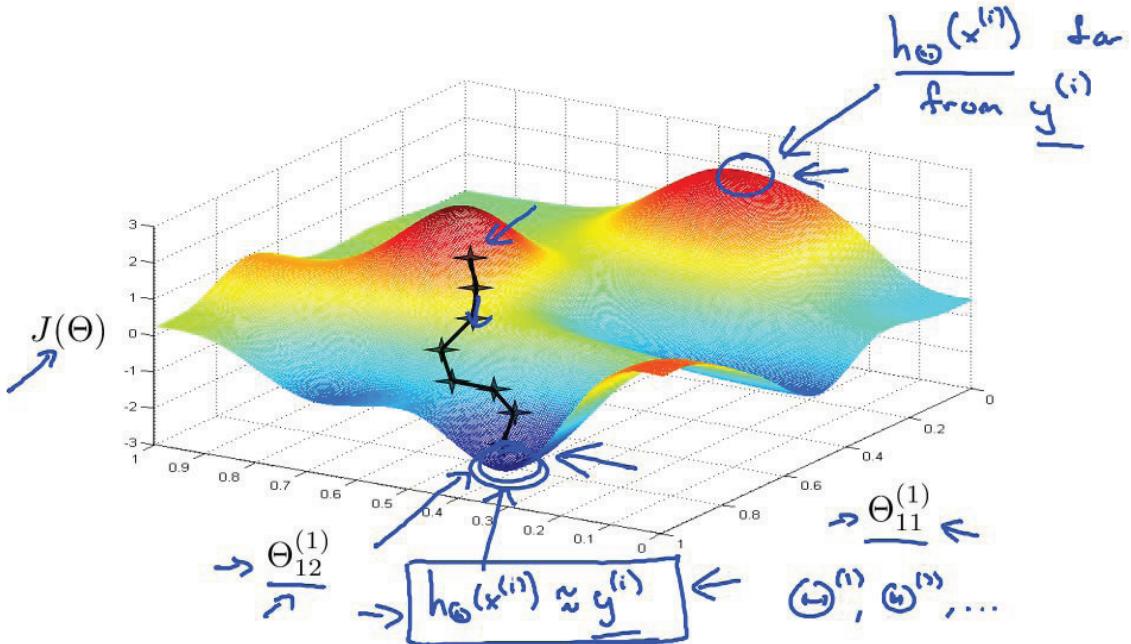
- أخيرا , هنستخدم القوانين اللي عندنا , عشان نجيب اقل قيمة للـ L

- تعالى نتعرف علي كيفية رسم الثيتات مع الـ L
- قيم الثيتا مش بتكون قليلة , ممكن تكون مصفوفات كاملة , لكن هنفترض حاليا انها اتنين بس مع الـ L



- من الواضح من الشكل ان النقط الحمرا هي القيم العالية للـ J والنقط الصفرا المتوسطة ، والزرقا لالقيم القليلة للـ J قيمة J تعبر عن الفرق بين قيم h المتوقعة ، وقيم y الحقيقية ، يعني مقدار الخطأ ، فمن الواضح ان كل ما تزيد قيمة J (النقط الحمرا) كل ما ده ووضح ان قيم الثيتا مش مطبوبة لأن الفرق كبير ، و ان الافضل اروح ورا النقط الزرقا

- لما باعمل قيم ثيتا اي قيم عشوائية ، فالكوند ببيدا من اي نقطة و خلاص ، و ببيدا يحسب قيمة الـ J ، بعدها بيجي الـ h عشان يحدد اتجاه النزول (عن طريق تغيير قيم الثيتات عن طريق الدليتا) ، لغاية لما نوصل لاقل قيمة ممكنة



• مثال عملي

- هنشوف هنا تطبيق في الـ NN على برامج قيادة السيارة بشكل آلي
- تحت علي الشمال ، نلاقي الطريق، الي لونه رصاصي شوية ، وهيميل يمين
- فوق علي الشمال هنلاقي عدادين ، اللي فوق هو يحدد فين مكان الدريكسيون ، بناء علي اختيار السوق البشري
- الخط الأبيض علي الشمال شوية يعني محد شمال حاجة بسيطة) ، بينما الخط اللي تحته بيوضح اختيار الخوارزم



● ازاي بيتم تدريب البرنامج :

- واحد يسوق العربية بسرعة هادية ، وتيجي الكاميرا تصور الطريق قدامها ، والكاميرا تلتقط كل ثانية صورة وتخزنها عندها ، وفي نفس الوقت ، الـ sensors المتركبة في الدريكسيون ، بتحسب عنها مقدار الدوران في الدريكسيون ، عشان يتم الربط بين ده و ده



- بعد فترة من التدريب ، يكون الخوارزم جاهز للتطبيق ، ويقوم السوق بالفعل بتشغيل القيادة الآلية ، والتي تصور الطريق و تقوم بالقيادة عليه بشكل مثالي
- رابط الفيديو

<https://www.youtube.com/watch?v=ilP4aPDTBPE>
