التعلم العميق Deep Learning

- الدرس الأول : التعلم العميق و الشبكات العصبية
  - الأسبوع الأول : مقدمة للتعلم العميق
  - الأسبوع الثاني : أساسيات الشبكات العصبية
  - الأسبوع الثالث : الشبكات العصبية المجوفة
  - الاسبوع الرابع : الشبكات العصبية العميقة
- الدرس الثاني : تطوير الشبكات العميقة : المعاملات العليا
  - الأسبوع الأول : السمات العملية للتعلم العميق
  - الأسبوع الثاني : الحصول علي القيم المثالية
  - الأسبوع الثالث : ضبط قيم الشبكات العميقة
- الدرس الثالث : هيكلية مشاريع الـ ML
  - الأسبوع الأول : استراتيجيات الـ ML - 1
  - الأسبوع الثاني : استراتيجيات الـ ML - 2
- الدرس الرابع : الشبكات العصبية الملتفة CNN
  - الأسبوع الأول : أساسيات الشبكات العصبية الملتفة
  - الأسبوع الثاني : حالات عملية من الشبكات العصبية الملتفة
  - الأسبوع الثالث : التعرف علي الأشياء
  - الاسبوع الرابع : التعرف علي الوجه
- الدرس الخامس : الشبكات العصبية المتكررة RNN
  - الأسبوع الأول : مفهوم الشبكات العصبية المتكررة
  - الأسبوع الثاني : المعالجة اللغوية الطبيعية NLP
  - الأسبوع الثالث : نماذج التتابع

.

درس 1: التعليم العميق و الشبكات العصبية

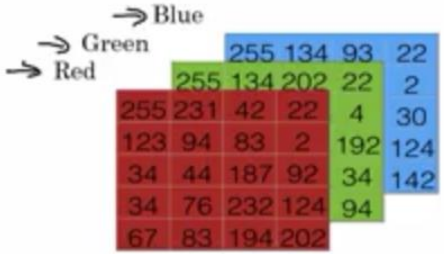
الأسبوع الثاني : أساسيات الشبكات العصبية

==============================

للتعرف علي أساسيات الـ NNعلينا التحدث بشكل مستفيض عن الكلاسيفيكاشن (التصنيف)

التصنيف هي عملية فرز المدخلات الي احد صنفين , غالبا ما تكون 1 و 0 , فمثلا اذا كنا نريد تصنيف الصور , هل بها قطة ام لا , فوجود قطة بها تعني 1 و عدم وجودها تعني 0

و علينا ان نتكلم كيف يتم تخزين الصور , فصورة مثل هذه , تخزن بهذه الطريقة في الحاسب



فالجهاز يقوم بعمل 3 مصفوفات لقيمة كل بيكسيل فيها , كم فيها من الالوان الثلاثة الرئيسية RGB
فلو كانت الصورة 100 بيكسيل في 100 بيسكيل , هيكون عندنا 3 مصفوفات ابعاد كل واحدة 100 في 100

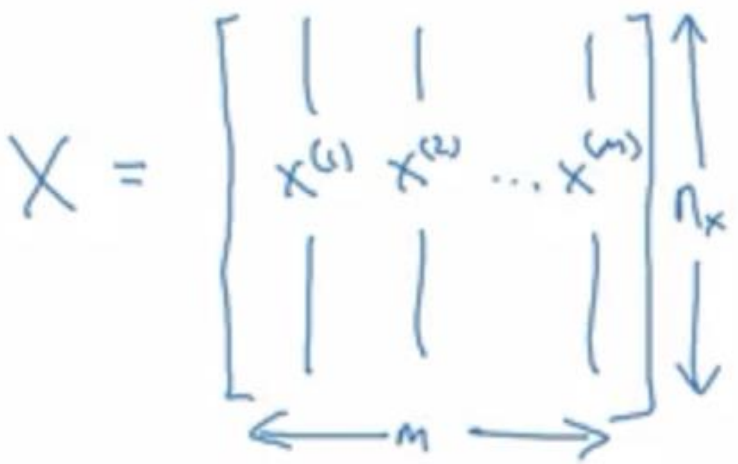
بعدها لازم يتم تحويل كل البيانات ديه لفيكتور (مصفوفة من بعد واحد) هيكون باسم X , فهيتم نقل ارقام مصفوفة الاحمر بالترتيب (255 و 231 و 42 و 22 و 123 و . . .) بحيث تكون 10 الاف رقم (100 في 100) بعدها يتم نقل ارقام الاخضر , بعدها الازرق , يعني هيكون الفيكتور اكس عبارة عن 30 الف صف في عمود واحد

$$x = \begin{bmatrix}255\\231\\42\\\vdots\\255\\134\\202\\\vdots\\255\\134\\93\\\vdots\end{bmatrix}\begin{array}{l}\left.\begin{array}{c}\\\\\\\end{array}\right\}\text{red}\\\left.\begin{array}{c}\\\\\\\end{array}\right\}\text{green}\\\left.\begin{array}{c}\\\\\\\end{array}\right\}\text{blue}\end{array}$$

وبالتالي هتكون الاكسات مصفوفة 30 الف رقم , والواي هيكون 1 او 0 حسب فيها قطة او لا

وقبل ما نكمل لازم نحدد بعض المصطلحات الهامة :

- مجموع عدد الارقام داخل مصفوفة X هيكون اسمه n , واللي هو حاصل ضرب الطول في العرض في 3 (حاليا 30 الف)
- عدد الصور المستخدم للتدريب (training sample) هيكون اسمه m (سمول)
- وبالتالي انا عندي x1 , x2 , x3 دلالة عن مصفوفات كل صورة فيهم لغاية xm
- فيه كمان , y1 , y2 لغاية ym للدلالة علي الناتج لكل صورة هل 1 يعني فيه قطة ولا 0 يعني مفيش
- عشان اجمع كل البيانات مع بعض هيكون فيه مارتكس اسمها X بالكابيتال , وهي تجميع كل الاكسات مع بعض , بحيث ارقام اول صورة تكون اول عمود , وتاني صورة تاني عمود و هكذا , فهيكون عدد الصفوف هو عدد الارقام لكل صورة (n) وعدد العواميد هو عدد الصور للتدريب (m)



اما واي فببساطة هنرص الارقام جنب بعض , بحيث المصفوفة المجمعة لواي Yكابيتال , هتكون صف واحد في m عمود يعني مثلا [1 0 1 1 0 0 0 1 1]

$$Y = [y^{(1)} \ y^{(2)} \ \cdots , y^{(m)}]$$

-

ماذا عن التصنيف ؟

عايزين نعمل لوجيستيك ريجريشين (تصنيف) لمجموعة من الصور (الانبت اكس) عشان يكون الناتج عندنا هل فيها قطة (واي تساوي 1) او مفيهاش قطة (واي تساوي 0)

يعني الفورمة هتكون كدة :

$$Given\, x\,,\;\; \hat{y} = P(y = 1|x), \text{where } 0 \;\le \hat{y} \le 1$$

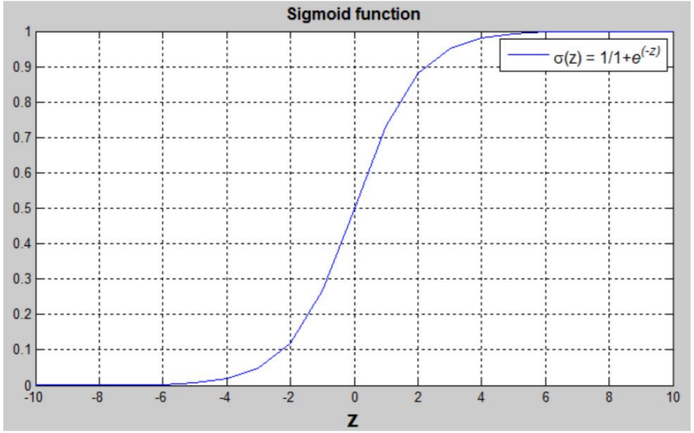
واي هات اللي هي قيمة واي المتوقعة , يعني قيمة الواي اللي الخوارزم هيتنبأ بيها , حيث واي هات هتكون بين الصفر و الواحد , صفر في حالة مفيش اي قطة , 1 في حالة كل الصور قطط

لاحظ اننا قبل كدة كنا بنقول علي القيمة المتوقعة رمز اتش بدل واي هات

هنقول علي معاملات الفيتشرز (العناصر) اللي هتكون الثيتات رمز w عشان ترمز للويتس (اللي هي قيمة ثيتات)
فهتكون واي هات (المتوقعة) تساوي :

$$\hat{y} = \sigma(w^T x + b)$$

رمز السيجما ده معناها ان دالة السيجمويد , اللي دايما بنستخدمها في الكلاسيفيكاشن , عشان بتجيب ناتج من صفر لواحد , واللي بيكون شكلها كدة



قيمة الزي هي الدالة (Wt x + b) , والسيجمويد هتكون 1 علي 1 زائد اكسبونينشيال ناقص الزي

$$s = \sigma(w^T x + b) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

وبالتالي لما تكون الزي قليلة سالب , هتكون الإي قيمة كبيرة , وواحد علي كبير يبقي صفر , و العكس , لما تكون إي قيمة كبيرة , هتكون الإي صغيرة جدا , فهتساوي واحد , ولما تكون زي بصفر , الإي هتكون بواحد يعني نص

تعالي نتكلم عن صيغة الدبليو و البي

الدبليو هي قيمة ثيتات و المفروض ان القيمة ديه كلها $w^T x + b$ تساوي , ثيتا زيرو في اكس زيرو + ثيتا 1 في اكس 1 + ثيتا 2 في اكس 2 و هكذا

ثيتا زيرو هي نفسها قيمة b و اكس زيرو بواحد , فثيتا زيرو في اكس زيرو تساوي البي , والبي نفسها b اختصار لكلمة bias و هو القيمة اللي كنا بنحطها في التنبؤ عشان الانحراف

بينما الدبليو هي مصفوفة عمود واحد و فيها صفوف بعدد الثيتات , فلما اعمل ترانسبوز ليها , هتكون صف واحد و فيها عوامد بعدد الثيتات , ولما اضربها في الاكس اللي هي عمود واحد و فيها صفوف بعدد الاكسات , تعطيني قيمة ثيتا 1 في اكس 1 + ثيتا 2 في اكس 2 و هكذا

$$X_0 = 1 , \quad x \in \mathbb{R}^{n_x+1}$$

$$\hat{y} = \sigma(\theta^T x)$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n_x} \end{bmatrix} \begin{matrix} \} b \leftarrow \\ \\ \} w \leftarrow \\ \\ \\ \end{matrix}$$

معادلة الخطأ للتصنيف Cost function for LR

الأول عايزين نوضح كام حاجة مهمة
دايما السوبر سكريبت (الرقم اللي بيكون فوق الرمز) , بيوضح رقم العينة اللي باتكلم عنها , بينما السب سكريبت (الرقم اللي تحت) بيوضح باتكلم عن انهي عنصر داخل نفس العينة

يعني لما يكون عندي إكس فوق فيه 3 , وتحت 5 , يبقي باتكلم علي العنصر الخامس في العينة التالتة , يعني إم تساوي 3 , وإن تساوي 5

إذن لو بصينا في المعادلة ديه

$$\hat{y}^{(i)} = \sigma(w^T x^{(i)} + b), \text{ where } \sigma(z^{(i)}) = \frac{1}{1 + e^{-z^{(i)}}}$$

معناها ان (واي هات آي) يعني قيمة واي المتوقعة للعينة آي (العينة رقم 5 مثلا) هي سيجمويد دبليو ترانسبوز (دبليو ملهاش علاقة بالآي لانها ثابتة) , مضروبة في إكس آي (يعني كل اكسات العينة الخامسة) , + بي اللي هي ثابتة

و بالتالي مصطلح زي آي , المقصود بيه قيمة دبليو ترانسبوز, مضروبة في إكس آي, + بي

كمان ممكن نقول إن فيه واي هات لكل واحد من العينة , واللي احنا نفسنا تكون بتساوي واي الحقيقية لنفس العينة , والمعطيات هي اكسات ووايات لكل واحد من العينا , كل أكس فيها هي في الحقيقة مصفوفة اكسات العناصر

$$Given\left\{\left(x^{(1)},y^{(1)}\right),\cdots,\left(x^{(m)},y^{(m)}\right)\right\},\ we\ want\ \hat{y}^{(i)}\approx y^{(i)}$$

وبالتالي معادلة الخطأ هتكون كالتالي , رمز إل اللي يشير لـ lose هيكون نص فرق قيمة واي هات وواي الحقيقية مربع
فلو كانت واي المتوقعة زي واي الحقيقية بالظبط (التوقع سليم) , هتكون إل تساوي صفر
وكل ما تبعد واي المتوقعة عن واي الحقيقية , تزداد قيمة الخطأ

لاحظ إن واي الحقيقية دايما برقم صحيح صفر او واحد (ممكن يكون فيه 2 و 3 لو كان تصنيف لاكثر من شئ) , بينما واي المتوقعة ممكن تكون رقم عشري

$$L\left(\hat{y}^{(i)},y^{(i)}\right)=\frac{1}{2}\left(\hat{y}^{(i)}-y^{(i)}\right)^2$$

بس المعادلة ديه محدش بيستخدمها حاليا لانها بتعمل صعوبة كبيرة في الوصول للقيمة الاوبتيمم للثيتات , فهنستخدم المعادلة اللوغاريتمية ديه :

$$L\left(\hat{y}^{(i)},y^{(i)}\right)=-\left(y^{(i)}\log\left(\hat{y}^{(i)}\right)+\left(1-y^{(i)}\right)\log\left(1-\hat{y}^{(i)}\right)\right)$$

وقبل ما نفسرها , هنتكلم عن خطوة سابقة عشان نفهمها :

في حالة كانت واي (القيمة الحقيقية) تساوي 1 , فالاحتمالية هتساوي واي هات (القيمة المتوقعة) , عشان لو كانت واي هات تساوي كمان , فالاحتمالية تساوي 1 يعني بتساويها (ترو) , ولو كانت هات تساوي صفر فالاحتمالية صفر كمان لانها مش بتساويها (فولس)

وفي حالة واي تساوي صفر , فالاحتمالية تساوي 1 ناقص واي هات , عشان لو كانت هات المتوقعة تساوي 1 فالاحتمالية صفر (فولس) لانها مش بتساوي واي , ولو كانت هات بصفر ,فالاحتمالية بواحد (ترو) لانها بتساويها

$$\text{If} \quad y = 1: \qquad p(y|x) = \hat{y}$$

$$\text{If} \quad y = 0: \qquad p(y|x) = 1 - \hat{y}$$

و ديه ممكن نصيغها بالمعادلة ديه :

$$p(y|x) = \hat{y}^{y} \, (1-\hat{y})^{(1-y)}$$

واي هات اس واي , مضروبة في 1 ناقص واي هات اس 1 ناقص واي

في حالة واي تساوي 1 , هتكون واي هات اس 1 , و1ناقص واي هات هتختفي

وفي حالة واي تساوي صفر هيختفي الجزء الاولي , وهيتبقي 1 – واي هات

ولو عملت لوج للطريفين , هتلاقي ان الاس هيبي جنب اللوج بحيث تعمل لنا المعادلة اللي بنتعامل معاها:

$$L(\hat{y}^{(i)}, y^{(i)}) = -(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)})\log(1 - \hat{y}^{(i)}))$$

وديه بتغطي الحالتين للواي

في حالة واي تساوي 1 , هيختفي التيرم التاني , وهتكون معادلة الخطأ هي سالب لوج الواي المتوقعة , و ده هيجعل قيمة الخطأ صفر لو كانت واي المتوقعة بواحد , وهيجعل الخطا كبير لو كانت واي المتوقعة ابعد عن 1

$$\text{If } y^{(i)} = 1: L\left(\hat{y}^{(i)}, y^{(i)}\right) = -\log\left(\hat{y}^{(i)}\right)$$

وفي حالة واي تساوي صفر , هيختفي الجزء الاول , وهتكون سالب لوج 1 – القيمة المتوقعة , ولو كانت واي المتوقعة تساوي صفر , هتكون لوج 1 , يعني الخطا صفر , ولو كانت ابعد عن الصفر قيمة الخطا بتزداد

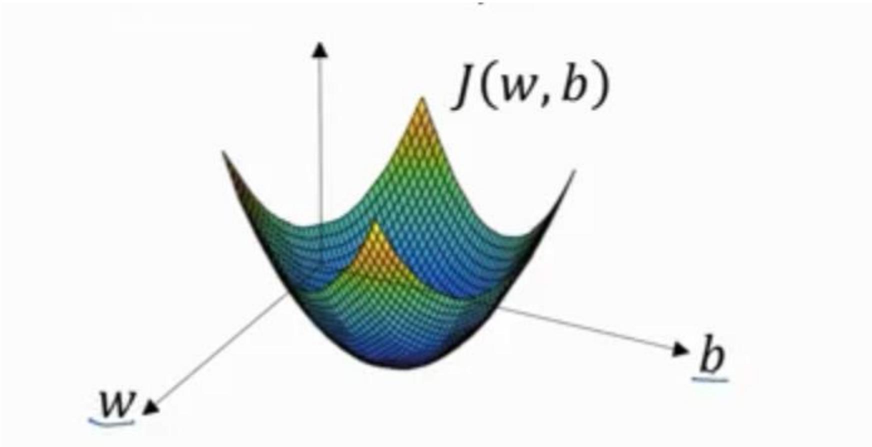
$$\text{If } y^{(i)} = 0: L\left(\hat{y}^{(i)}, y^{(i)}\right) = -\log\left(1 - \hat{y}^{(i)}\right)$$

كل اللي فات ده كانت قيمة الخطأ لعينة واحدة من العينات , فماذا عن الخطأ الكلي , ده اللي هيكون اسمه معادلة الكوست , اللي بنرمز ليه برمز جي J , واللي هيكون مجموع قيم اخطاء كل عينة , مقسوم علي عددها إم
يعني هيكون كدة

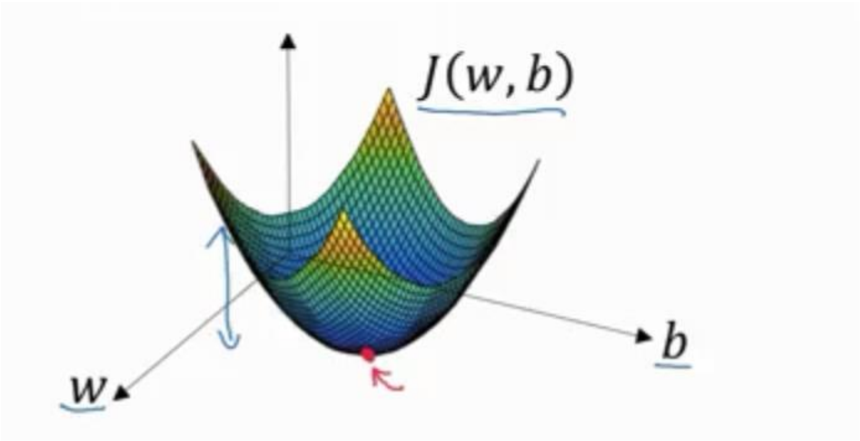
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L\left(\hat{y}^{(i)}, y^{(i)}\right) = -\frac{1}{m}\sum_{i=1}^{m}\left[\left(y^{(i)}\log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right)\log\left(1 - \hat{y}^{(i)}\right)\right]$$

الانحدار التدريجي Gradient Descent

فاكرينالانحدار التدريجي من محاضرات التنبؤ , واللي بيكون شكله كدة



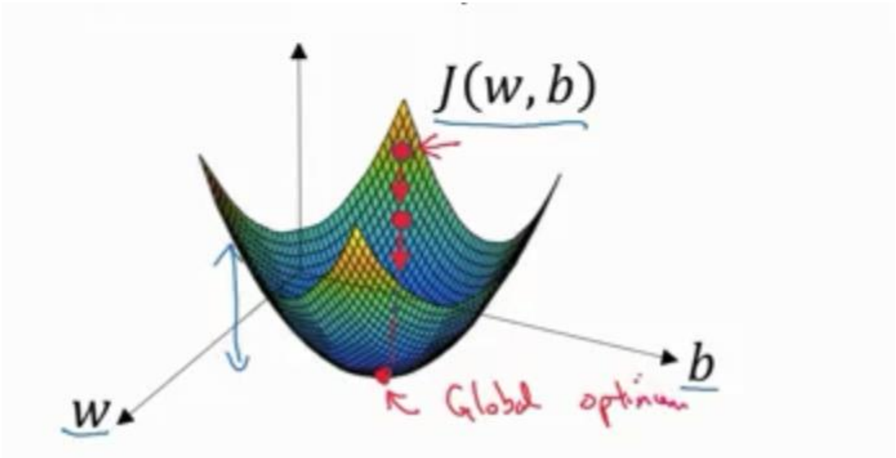
قلنا من شوية ان الـ w هي الاوزان يعني مصفوفة الثيتات , والـ b هي قيمة الثيتا صفر , وقيمة الـ J هي قيمة الخطأ , أي إن قيمة كلا من الدبليو و البي بتحدد قيمة الـ J , وبالتالي احنا بنبحث عن قيم دبليو و بي اللي بتعمل قيمة جي اقل ما يكون



ولازم تكون المعادلة الخاصة بالجي تكون مقعرة , عشان يكون ليها حد ادني واضح اعرف اوصله , وده السبب اللي خلانا ناخد المعادلة اللوغاريتمية لان نهايتها واضحة وبتكون مقعرة كدة

بينما فيه معادلات تانية مش بتكون مقعرة , فمش هينفع اجيب منها الحد الادني زي كدة

عشان كدة انا ممكن ابدا بارقام مبدئية للدبليو و البي , واعمل تكرار Iteration اكتر من مرة وكل مرة هجيب قيمة للجي اقل , لغاية لما اجيب الـ global minimum



طيب ماذا عن المعادلة الرياضية , ازاي باقدر اوصل للقيمة الدنيا في الـ J ؟

ببساطة بابدأ افرض قيمة دبليو باي قيمة , واستخدم المعادلة ديه عشان اوصل للقيمة الدنيا :

Repeat {

$$w := w - \overset{\text{learning rate}}{\alpha} \underbrace{\frac{dJ(w)}{dw}}_{\text{"}dw\text{"}}$$

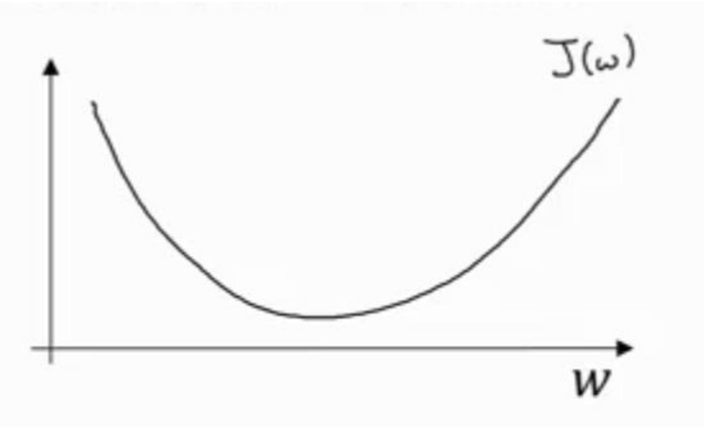
}

$$w := w - \alpha\, dw$$

وهي معناه ببساطة ان قيمة دبليو الجديدة , هي القديمة , ناقص حاصل ضرب الفا في تفاضل جي بالنسبة لدبليو

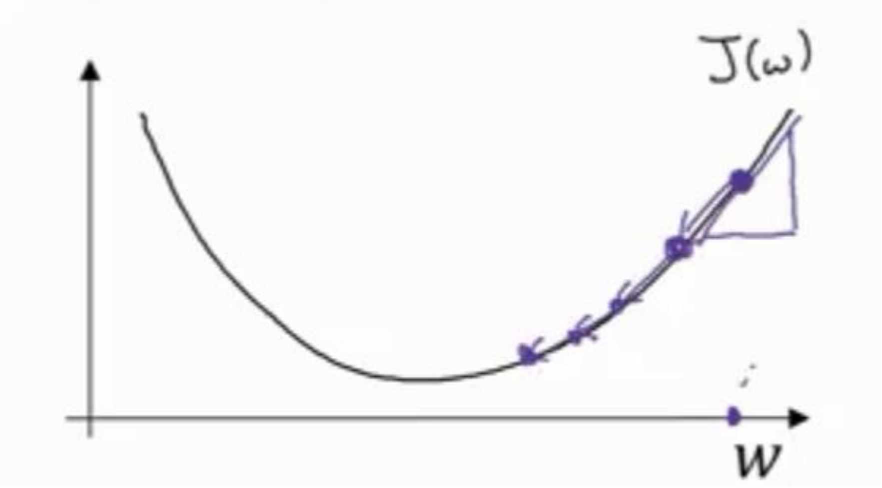
اولا الفا هي الـ learning rate واللي بتحددلنا مدي سرعة عمل الـ learning , وكالمعتاد , لو كبيرة هتكون سريعة و مش دقيقة , لو قليلة هتكون دقيقة و بطيئة

بالنسبة للتفاضل فلازم نشوف الرسم عشان نفهم



لو هنتجاهل بي مؤقتا , ونقول ان الدبليو هي المؤثر الوحيد (متنساش ان دبليو هي مصفوفة الثيتات والبي هي ثيتا 0) , فلو قلنا ان المعادلة اللوغاريتمية هتكون مقعرة كدة , ففيه قيمة للدبليو بحيث تعمل القيمة الدنيا للـ J

فلو فرضنا اني لما اخترت قيمة دبليو كانت علي اليمين , يعني اكثر من قيمة دبليو المثالية زي كدة :

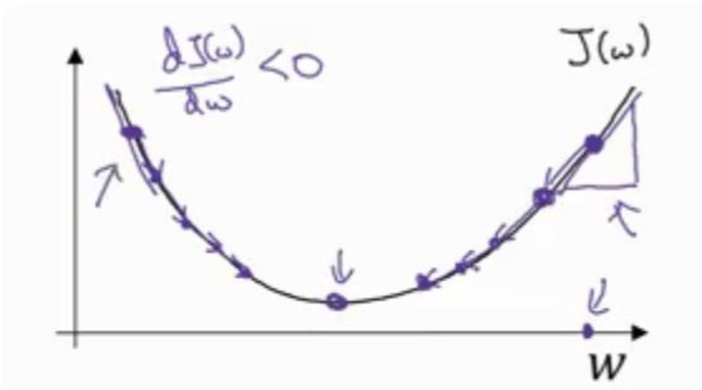


ففي الجانب الايمن , لو عملت اشتقاق للجي , هلاقي ان فيمته هتكون موجبة , وكبيرة شوية , فلما ابص علي المعادلة , هنلاقي ان الدبليو الجديدة هتكون القيدمة ناقص قيمة التفاضل الكبير شوية مضروب في الفا , يعني قيمة دبليو هتقل شوية , وهتنقل للنقطة اللي علي شمالها

لما اكرر تاني هلاقي ان التفاضل اللي هيتطرح هيكون اقل شوية , لان الميل قل شوية , فالقيمة المطروحة هتكون اقل شوية , وبرضه هدخل شمال

هكرر الموضوع لغاية لما اوصل للحد المثالي للدبليو او قريب منه , هلاقي ان الاشتقاق قريب من الصفر , عشان الخط مستقيم , يعني الدبليو مش هتتغير

ولو كنت اخترت الدبليو علي الشمال زي كدة :



علي الشمال الاشتقاق بالسالب , فلما اضربه في الفا و اطرحه , هلاقي ان دبليو بتزيد معايا مش بتقل , يعني بتدخل يمين , وكل ما ادخل يمين قيمة الاشتقاق بتقل , لغاية برضه لما اوصل للحد الادني

وهيتم في البي نفس اللي حصل في دبليو , مع فارق ان الاشتقاق هيكون بالنسبة لها

$$w := w - \alpha \frac{d J(w,b)}{dw}$$

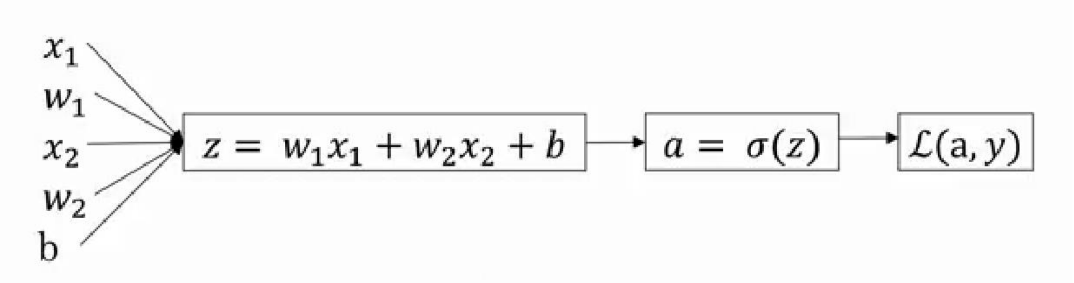
$$b := b - \alpha \frac{d J(w,b)}{db}$$

طب لو فيه اكتر من إكس في المعادلة الاساسية
مش عايزين ننسي ان المعادلات العادية في حالة وجود اكس هي :

$$z = w^T x + b$$
$$\hat{y} = a = \sigma(z)$$
$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$

ولو فيه اكتر من اكس , هتكون البي هي هي (لانها قيمة مضافة ثابتة هيتجمعو مع بعض) لكن هيكون فيه اتنين دبليو لاتنين اكس , لان كل اكس هيكون ليها الثيتات الخاصة بيها والي هي الدبليو

فهتكون كالتالي :



لو هبدا اعمل باك بروباجيشن , يعني ارجاع عكسي , عشان اشوف التغيير في النهايات بيعمل تغيير قد ايه في المداخل

فهبدا في التفاضل الاخير . اللي هيكون

dl/da

يعني تفاضل المعادلة النهائية بالنسبة لقيمة ايه , واللي هي قيمة سيجمويد الزي ,, زي ما هي فوق

لو جيت تعمل اشتقاق للمعادلة ديه بالنسبة لايه :

$$\mathcal{L}(a, y) = -(y\log(a) + (1-y)\log(1-a))$$

هتلاقي ان الواي ثابت بالنسبة لها , و اشتقاق اي لوج هو اشتقاقه فوق علي نفسه, و بالتالي الاشتقاق هيكون :

$$-\frac{y}{a} + \frac{1-y}{1-a}$$

ولو رجعت خطوة ورا , عشان تجيب اشتقاق ايه بالنسبة لزي يعني

da/dz

هيكون قيمته

a(1-a)

وبالتالي عشان اجيب قيمة اشتقاق المعادلة كلها إل بالنسبة لزي , هاستخدم قاعدة السلسلة chain rule

$$\frac{dL}{dz} = \frac{dL}{da} \cdot \frac{da}{dz}$$

يعني هضرب ديه $-\frac{y}{a} + \frac{1-y}{1-a}$ في ديه $a(1-a)$

هتعطيني نتيجة مباشرة و هي :

$$a - y$$

و بالتالي النتائج الفرعية لاشتقاق إل بالنسبة لدبليو 1 او 2 او بي , هتكون كدة

$$\frac{\partial L}{\partial w_1} = \text{"}dw_1\text{"} = x_1 \cdot dz \qquad dw_2 = x_2 \cdot dz \qquad db = dz$$

كل اللي فات ده عن عينة واحدة , يعني صف واحد , طيب ماذا عن التطبيق لأكثر من صف , يعني عدد الإم كبير ؟ ؟

وقتها المعادلات المعروفة اللي هي كانت كدة

$$z = w^T x + b$$
$$\hat{y} = a = \sigma(z)$$
$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$

هتبقي كدة :

For $i = 1$ to m

$$z^{(i)} = w^T x^{(i)} + b$$
$$a^{(i)} = \sigma(z^{(i)})$$
$$J{+}{=} -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})]$$

يعني كل معادلة هتكون مرتبطة بقيمة بتفاصيل الصف اللي هي معاه
ولما احي اعمل الاشتقاقات , قيمة الـ dz اللي هي اصلا dj/dz اللي هي كانت :

$$a-y$$

هتكون :

$$\underline{dz^{(i)}} = a^{(i)} - y^{(i)}$$

وقيم dw1 اللي هي اصلا dl/dw1 واخواتها واللي كانت اصلا :

$$\frac{\partial L}{\partial w_1} = \text{"}dw_1\text{"} = x_1 \cdot dz. \qquad dw_2 = x_2 \cdot dz. \quad db = dz.$$

هتكون كدة

$$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$$
$$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$$
$$db \mathrel{+}= dz^{(i)}$$

(متنساش ان الإل هي نفسها الجي , بتتكتب كدة او كدة)

بعدها متنساش انك تقسم كل قيم جي و دي دبليو1 و 2 و بي علي الإم (عدد العناصر) عشان تجيب المتوسط

$$J \mathrel{/}= m$$

$$dw_1 \mathrel{/}= m \;;\quad dw_2 \mathrel{/}= m \;;\quad db \mathrel{/}= m.$$

واخيرا اعمل ابديت لقيم دبليو 1 و 2 و بي , مع طرحها من قيمة الاشتقاق مضروب في الفا (معامل التدريب)

$$\begin{aligned} w_1 &:= w_1 - \alpha \underline{dw_1} \\ w_2 &:= w_2 - \alpha \underline{dw_2} \\ b &:= b - \alpha \underline{db} \end{aligned}$$

وديه الصورة العامة :

Logistic regression on m examples

$J=0;\ dw_1=0;\ dw_2=0;\ db=0$

For $i=1$ to m

$z^{(i)} = w^T x^{(i)} + b$

$a^{(i)} = \sigma(z^{(i)})$

$J += -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$

$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$ $\quad n=2$

$db \mathrel{+}= dz^{(i)}$

$J /= m$

$dw_1 /= m;\ dw_2 /= m;\ db /= m.$

$dw_1 = \frac{\partial J}{\partial w_1}$

$w_1 := w_1 - \alpha\, dw_1$

$w_2 := w_2 - \alpha\, dw_2$

$b := b - \alpha\, db$

Andrew Ng

_

التوجيه Vectorization

المقصود بها هو التسريع في العمليات الرياضية للاكواد المستخدم عبر استخدام تكنيكات محددة , تتعلق بالمصفوفات , اكثر من الدورات (loops) وتسمي فيكتوريزاشن لانها تشير الي المصفوفات

والعملية مهمة لانها بتختصر كميات كبيرة من الوقت , خاصة لما يكون الكود اصلا طويل و محتاج وقت كبير

فمثلا لو انا عايز احسب قيمة زي اللي هي :

$$z = w^T x + b$$

متنساش ان الدبليو و الاكس مصفوفتين كلا منهما عمود واحد , وعدد الصفوف يساوي عدد العناصر إن

$$w = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix} \quad x = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$$

فلو هعملها بالطريقة العادية (غير الموجهة) هتكون

Non-vectorized:

```
z=0
for i in range(n-x):
    z+= w[i]*x[i]
z+=b
```

بينما لو اتعملت بالطريقة الموجهة هتكون سطر واحد :

Vectorized

$$z = \underbrace{np.dot(w,x)}_{w^T x} + b$$

فالتوجيه مش بس اسهل في كتابة الكود , ده كمان اسرع في التنفيذ بكتير

و هنا فيه مثال عملي في بايثون

```
import numpy as np
import time

a = np.random.rand(1000000)
b = np.random.rand(1000000)

tic = time.time()
c = np.dot(a, b)
```

```
toc = time.time()
print(c)
print("vectorized version: " + str(1000 * (toc-tic)) + "ms")

c = 0
tic = time.time()
for i in range(1000000):
    c += a[i]*b[i]

toc = time.time()
print(c)
print("non-vectorized version: " + str(1000 * (toc-tic)) + "ms")


250347.13984556697
vectorized version: 3.0002593994140625ms
250347.1398455659
non-vectorized version: 1409.0805053710938ms
```

تم عمل عملية رياضية وهي ضرب مليون رقم عشوائي في مليون رقم عشوائي , لما عملناها بالطريقة العادية اخدت 1409 ملي ثانية , بينما بالطريقة العادية اخدت 3 ملي ثانية , والنتيجة هي هي , فطبعا في العمليات المعقدة هيكون الفرق باين جدا

وهتلاحظ ان الفرق تقريبا من 300 ل 400 ضعف , و ده معناه ان العملية اللي هتاخد دقيقة واحد في العملية الموجهة , هتحتاج 5 ساعات في الغير موجهة

ملحوظة :

CPU : Central Process Unit (normal machine)
GPU : Graphical Process Unit (plugged device)
SIMD : Single Instruction Multiple Data

فالأساس هنا انك تحاول تتجنب استخدام (فور لووب) علي قد ما تقدر , طالما فيه بديل ليها

فمصلا لو عايز ا ضرب مصفوفة بصفوف و عواميد , في فيكتور , اللي هيكون عمود واحد و فيه صفوف زي كدة

$$u = Av$$

فممكن اضربها بالطريقة العادية الطويلة للكود كدة :

```
u = np.zeros((n,1))
for i ...
    for j ...
        u[i] += A[i][j] * v[j]
```

او الاسرع اني استخدم الطريق الموجهة , بكود مباشر كدة :

A = np.dot(A, v)

u = np.dot(A,v

نفس الفكرة , لو عايز مثلا اعمل اكسبونينشيال لعدد من العناصر زي كدة

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \rightarrow u = \begin{bmatrix} e^{v_1} \\ e^{v_2} \\ \vdots \\ e^{v_n} \end{bmatrix}$$

فبالطريقة العادية :

```
u = np.zeros((n,1))
for i in range(n):
    u[i]=math.exp(v[i])
```

```
u = np.zeros((n, 1))
for i in range(n):
        u[i] = math.exp(v[i])
```

او بالطريقة الموجهة :

```
import numpy as np
u = np.exp(v)
```

```
u = np.exp(v)
```

ونفس الفكرة لباقي الدوال المشابهة :

np.log(v)
np.abs(v)
np.maximum(v,0)
v**2   1/v

ولو هنوسع المثال شوية   فمثلا في الكود الكبير الخاص التصنيف اللي هو :

Logistic regression derivatives

J = 0, dw1 = 0, dw2 = 0, db = 0
for i = 1 to m:

$$z^{(i)} = w^T x^{(i)} + b$$
$$a^{(i)} = \sigma(z^{(i)})$$
$$J \mathrel{+}= -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$$
$$dz^{(i)} = a^{(i)}(1 - a^{(i)})$$
$$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$$
$$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$$
$$db \mathrel{+}= dz^{(i)}$$

$$J = J/m,\ dw_1 = dw_1/m,\ dw_2 = dw_2/m,\ db = db/m$$

هتلاحظ ان فيه اتنين فور لوب , الاولي فوق المكتوبة , والتانية اللي عند dw اللي هي

$$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$$
$$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$$
$$db \mathrel{+}= dz^{(i)}$$

وده في حالة ان فيه اكتر من دبليو

فممكن استغني عن الفور هنا عن طريق اني اعمل فيكتور باسم دبليو فوق كدة :

$$dw = np.zeros((n_x, 1))$$

بدل كدة :

$$dw1 = 0, \quad dw2 = 0,$$

ووقتها هيكون الضرب كدة

$$dw \mathrel{+}= x^{(i)} dz^{(i)}$$

بدل كدة :

$$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$$
$$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$$
$$db \mathrel{+}= dz^{(i)}$$

و في القسمة علي إم هتكون قسمة dw مباشرة , بدل ما اقسم عنصر عنصر

$$dw_1 = dw_1/m, \quad dw_2 = dw_2/m$$

كمثال تاني اعمق
لو قلنا في التصنيف اني عايز اجيب زد 1 , زد 2 , وهكذا , هتكون معادلاتهم كدة :

$$z^{(1)} = w^T x^{(1)} + b \qquad z^{(2)} = w^T x^{(2)} + b \qquad z^{(3)} = w^T x^{(3)} + b$$

$$a^{(1)} = \sigma(z^{(1)}) \qquad a^{(2)} = \sigma(z^{(2)}) \qquad a^{(3)} = \sigma(z^{(3)})$$

متنساش ان الدبليو هي عبارة عن مصفوفة ثيتات , بالشكل ده

$$\Theta = \underbrace{\begin{bmatrix} \Theta_0 \\ \Theta_1 \\ \Theta_2 \\ \vdots \\ \Theta_{n_x} \end{bmatrix}}_{} \begin{matrix} \}\, b \leftarrow \\ \\ \left.\begin{matrix} \\ \\ \\ \end{matrix}\right\} w \leftarrow \\ \\ \end{matrix}$$

و بنسميها رمز دبليو كبير

اما كل اكس من الاكسات , فهنجمعها في مصفوفة X كبيرة اللي هتكون عواميدا بعدد الإكسات (عدد العناصر m) و صفوفها بعدد العناصر n اللي هي اصلا عدد صفوف اي اكس فيهم

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

وممكن اقول علي المطلوب (الزدات) ان فيه مصفوفة كبيرة اسمها Z من صف واحد و هي تجمع كل الزدات في عواميد كدة

$$Z = [z^{(1)} \; z^{(2)} \cdots z^{(m)}]$$

ساعتها ممكن نقول ان المعادلة النهائية هي :

$$Z = w^T X + b$$

فالدبليو ترانسبوز هتكون صف واحد و عواميد فيها الثيتات , ولما تضرب في X اللي هتكون كدة

$$[\text{———}]\, w^T \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

هتكون النتيجة عبارة عن قيم زي كدة

$$[w^T x^{(1)} \quad w^T x^{(2)} \quad \cdots \quad w^T x^{(m)}]$$

لما اجي اجمع عليها قيمة b فهضيفها لكل عنصر فيها زي كدة

$$\underbrace{[\boxed{w^T x^{(1)} + b} \quad \boxed{w^T x^{(2)} + b} \quad \cdots \quad w^T x^{(m)} + b]}_{1 \times m}$$

يعني هتعملي قيم زدات

لاحظ ان قيمة b هي قيمة مفردة يعني مصفوفة 1 في 1 , بس بايثون لما بيشوفك بتجمع رقم علي مصفوفة , فهيفرد الرقم عشان يكون مصفوفة بنفس حجم اللي بيتجمع عليها , والعملية ديه اسمها broadcasting

وبالتالي المعادلة النهائية هتكون :

$$Z = np.dot(w.T, X) + b$$

ونفس الفكرة في اني اجيب الـ a عن طريق عمل مصفوفة كبيرة اسمها A و يكون فيها قيم a كلها

$$A = [a^{(1)} \ a^{(2)} \ \dots \ a^{(m)}] = \sigma(Z)$$

و ده عن طريق استخدام دالة سيجمويد لكل قيم زي Z مع بعض , قيم زي كابيتال اللي هي مصفوفة كل قيم زي اللي جبتها

_

طيب ماذا عن ايجاد قيم الاشتقاقات :

عارفين من قبل كدة ان اشتقاق الزي هيكون بالمعادلة :

$$a - y$$

وبالتالي جميع اشتقاقات الزيهات هتكون :

$$dz^{(1)} = a^{(1)} - y^{(1)} \qquad dz^{(2)} = a^{(2)} - y^{(2)}$$

فهنعمل تجميع لاشتقاقات الزيهات في dZ كبيرة , واللي هتحتوي علي dz صغيرة

$$dZ = \underbrace{[dz^{(1)}\ dz^{(2)} \dots dz^{(m)}]}_{1\times m}$$

و ممكن نجمع الايهات الصغيرة في A كبيرة , وكذلك الوايات

$$A = [a^{(1)} \dots a^{(m)}] \qquad Y = [y^{(1)} \dots y^{(m)}]$$

فهتكون المعادلة المجمعة ليهم هي :

$$dZ = A - Y = [a^{(1)} - y^{(1)} \quad a^{(2)} - y^{(2)} \quad \dots]$$

طيب ماذا عن باقي الاشتقاقات , دبليو و بي ؟

الـ dw اساسا هي المفروض مجموع كل اكس في الـ dz الخاصة بيها, بعدها نقسمها علي m , يعني كدة

$$dw = 0$$
$$dw \mathrel{+}= x^{(1)} dz^{(1)}$$
$$dw \mathrel{+}= x^{(2)} dz^{(2)}$$
$$\vdots$$
$$dw \mathrel{/}= m$$

والـ db هتكون مجموع الـ dz مع بعضها و قسمتها علي m

$$db = 0$$
$$db \mathrel{+}= dz^{(1)}$$
$$db \mathrel{+}= dz^{(2)}$$
$$\vdots$$
$$db \mathrel{+}= dz^{(m)}$$
$$db \mathrel{/}= m.$$

فلو بدأنا بالـ db هنعملها ببساطة مجموع مصفوفة فيها كل الـ dz مع بعض

$$= \frac{1}{m} \, np.sum(dZ)$$

أما الـ dw فهتكون معادلتها

$$dw = \frac{1}{m} X \, dZ^T$$

متنساش ان الـ X الكبيرة هي مصفوفة صف واحد و عواميدها هي قيم إكسات لكل عينة من عينات الـ m
أما الـ dZ الكبيرة فهي كمان مصفوفة صف واحد و عواميدها قيم dz الصغيرة

فلما تعمل تدوير للـ dZ هتكون عمود واحد بصفوف كتير , اضرب اكس فيها الاول هتتحول للقيم :

$$\underline{x^{(1)}dz^{(1)}} + \dots + \underline{x^{(m)}dz^{(m)}}$$

ولما تقسمها علي m, هتجيبلك القيمة النهائية ليها , وبسطر واحد من الكود , ومن غير اي استخدام لفور لوب اللي بتضيع الوقت

وبالتالي العملية كلها اللي هي ديه

$J = 0,\ dw_1 = 0,\ dw_2 = 0,\ db = 0$

for i = 1 to m:

$$z^{(i)} = w^T x^{(i)} + b$$
$$a^{(i)} = \sigma(z^{(i)})$$
$$J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)})]$$
$$dz^{(i)} = a^{(i)} - y^{(i)}$$
$$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$$
$$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$$
$$db \mathrel{+}= dz^{(i)}$$

$J = J/m,\ dw_1 = dw_1/m,\ dw_2 = dw_2/m$

$db = db/m$

هتتحول للكود ده

$$Z = w^T X + b$$
$$= np.dot(w.T, X) + b$$
$$A = \sigma(Z)$$
$$dZ = A - Y$$
$$dw = \frac{1}{m} X dZ^T$$
$$db = \frac{1}{m} np.sum(dZ)$$

$$w := w - \alpha dw$$
$$b := b - \alpha db$$

_

عملية الـ broadcasting

و هي تستخدم في بايثون بالتحديد , عشان اقدر اتعامل مع مجموعة من العمليات مرة واحدة و بكود بسيط بدل ما اعمل عدد كبير من السطور , وده بيتم غالبا باستخدام مكتبات بايثون العديدة

فمثلا لو عندي مصفوفة فيها ارقام زي كدة

$$\begin{array}{r c} & \begin{array}{cccc} \text{Apples} & \text{Beef} & \text{Eggs} & \text{Potatoes} \end{array} \\ \begin{array}{r} \text{Carb} \\ \text{Protein} \\ \text{Fat} \end{array} & \begin{bmatrix} 56.0 & 0.0 & 4.4 & 68.0 \\ 1.2 & 104.0 & 52.0 & 8.0 \\ 1.8 & 135.0 & 99.0 & 0.9 \end{bmatrix} \end{array}$$

واللي هي عبارة عن عدد الكالوري اللي في الكاربوهاديرات , والبروتين و الدهون , في اربع اكلات معينة

فلو انا محتاج اعمل نسبة مؤية لكل اكلة فيهم , يعني اشوف ال 56 كالوري في الكاربوهايدرات نسبتها قد ايه من كل الكالوريز (يعني علي مجموع الـ 56 مع 1.2 , مع 1.8 الخاصة بالتفاح) , ساعتها ممكن اعملها كدة :

```
import numpy as np

A = np.array([        [56.0, 0.0, 4.4, 68.0],
                      [1.2, 104.0, 52.0, 8.0],
                      [1.8, 135.0, 99.0, 0.9]        ]        )
cal = A.sum(axis=0)
print(cal)
```

```
# [59.  239.  155.4  76.9]

percentage = 100*A/cal.reshape(1,4)
print(percentage)
#[[ 94.91525424   0.           2.83140283  88.42652796]
# [  2.03389831  43.51464435  33.46203346  10.40312094]
# [  3.05084746  56.48535565  63.70656371   1.17035111]]
```

اللي حصل اني كتبت المصفوفة , و عملت مجموع لكل عمود , بعدها قسمت كل قيمة فيهم علي رقم كل عمود من المصفوفة التانية

و أمر sum مع كلمة axis=0 معناها اجمع راسيا , ولو قلتله تساوي 1 هيجمع افقيا , ولو من غير كلمة اكسي خالص هيجمع كل القيم لقيمة واحدة

بالنسبة لامر reshape هو مكتوب احتياطي مش اكتر , لان اصلا المصفوفة اللي طلعت حالا كانت هي 1 في 4 , فلو مكتبتهاش مفيش مشكلة

طيب ماذا عن عملية القسمة ,والمعروف ان القسمة في المصفوفة بتتم في حالتين , الأول لو هاقسم علي رقم واحد , فبيتم قسمة كل ارقام المصفوفة علي نفس الرقم ده

الحالة التانية لو هاقسم علي مصفوفة بنفس ابعادها بالظبط , ساعتها بيتقسم كل رقم من المصفوفة الاولي علي الرقم من المصفوفة التانية

فلو هاقسم المصفوفة ديه

15	17	4	18
25	3	22	14

علي ديه

3	1	2	6
5	1	11	7

هتكون النتيجة كدة

5	17	2	3
5	3	2	2

تعالي نشرح بالتفصيل

لو جيت اجمع مصفوفة علي رقم , فبايثون اوتوماتيك هيحول الرقم ده لمصفوفة

$$\begin{bmatrix}1\\2\\3\\4\end{bmatrix} + \begin{bmatrix}100\\100\\100\\100\end{bmatrix} \cancel{100} = \begin{bmatrix}101\\102\\103\\104\end{bmatrix}$$

فلو جمعنا المصفوفة ديه علي رقم 100 , فهو اوتوماتيك هيحولها لمصفوفة شبهها , وهتعمل الارقام ديه

كذلك الحال لو تم عمل جمع مصفوفة علي مصفوفة تانية مش مقاسها , فلو هينفع تكبر و تتنسخ عشان تكون قدها هيعمل كدة فمثلا

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}100 & 200 & 300\end{bmatrix}$$

هتكون كدة

$$\underset{(m,n)\ (2,3)}{\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}} + \underset{(1,n) \leadsto (m,n)\ (2,3)}{\begin{bmatrix}100 & 200 & 300\\100 & 200 & 300\end{bmatrix}} = \begin{bmatrix}101 & 202 & 303\\104 & 205 & 306\end{bmatrix}$$

كمان لما نجمع ديه علي ديه

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}100\\200\end{bmatrix}$$

هيعملها كدة

$$\underset{(m,n)}{\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}} \quad + \quad \underset{\substack{(m,1)\\ \downarrow \\ (m,n)}}{\begin{bmatrix}100\\200\end{bmatrix}} \begin{matrix}100 & 100\\200 & 200\end{matrix} \quad = \quad \begin{bmatrix}101 & 102 & 103\\204 & 205 & 206\end{bmatrix}$$

وبالتالي لما بيتم عملية رياضية بين مصفوفتين , والمصفوفة التانية مش متلائمة مع الاولي , بايثون بيحولها لو ينفع

عشان كدة لما قسمنا مصفوفة 3 في 4 , علي مصفوفة 1 في 4 , بايثون مد المصفوفة المقسوم عليها اللي هي 1 في 4 ,وخلاها 3 في 4 زي الاولي , بحيث يقسم كل رقم علي اخوه , وبالتالي تم قسمة كل رقم من ارقام الكالوري , علي المجموع عشان يجيب النسبة المئوية

_

و كون ان بايثون فيه خاصية الـ broadcasting (اللي هي تمديد المصفوفات حسب الشكل المطلوب) فديه ميزة و عيب

ميزة انها بتختصر وقت و جهد في الكتابة , وعيب انها بتخلي البرنامج يعمل حاجات ممكن انت مش مخططها , فهتعمل مشاكل

يعني مثلا لو عملنا الكود ده :

```
import numpy as np
a = np.random.randn(5)
print(a)
# [ 1.2, 2.3, 3.4, 4.5, 5.6 ]
print(a.shape)
# (5,)
print(a.T)
# [ 1.2, 2.3, 3.4, 4.5, 5.6 ]

Print(np.dot(a,a.T))
4.06
```

هنا نلاحظ ان في الاول عملنا مصفوفة 5 صفوف في عمود واحد , ولما ضربناها في التدوير بتاعها اللي هيكون 1 في 5 , المفروض يعملنا مصفوفة 5 في 5 , لكن بايثون عمل تحويل للنوع ده فورا فعمل الاولي 1 في 5 , والتانية 5 في 1 , فالناتج طلع 1 في 1
فلو انت عايز تجبر بايثون علي مسار معين , متستخدمش امر

```
a = np.random.randn(5)
```

لكن

a = np.random.randn(5,1)

يعني متخليش بايثون يفرضها اوتوماتيك لكن حددهاله واللي هيعملها دلوقتي 5 في 1 , ولما تضربها في تدويرها هيعملك مصفوفة 5 في 5 فالصيغة ديه مش كويسة

```
a = np.random.randn(5)
a.shape = (5, )
"rank 1 array"
```

الاحسن نستخدم ديه لفيكتور الصف او العمود

```
a = np.random.randn(5,1) -> a.shape = (5,1)   column vector
a = np.random.randn(1,5) -> a.shape = (1,5)   row vector
```

او ممكن استخدم امر ريشاب لو تم كتابة الفيكتور بشكل مش سليم

```
a = a.reshape((5,1))
```

-