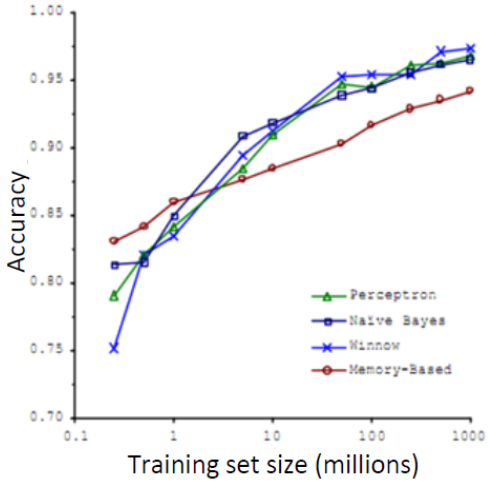
الأسبوع العاشر

التعامل مع البيانات الضخمة

- البيانات الضخمة هي وقود الـ ML و بدونها تكون ليس لها فائدة
- العدد الضخم من الصفوف m و الأعمدة n هي التي تجعل الخوارزم يحدد بدقة و كفاءة التنبؤ او التقسيم
- كفاءة الـ ML الان افضل من السابق ليس لتطوير الماكينات او الخوارزميات , ولكن لكثرة البيانات
- و نري هنا العلاقة بين كمية البيانات التي يتم التعامل بها و كفاءة العملية



- حتي ان بعض العلماء قالو :

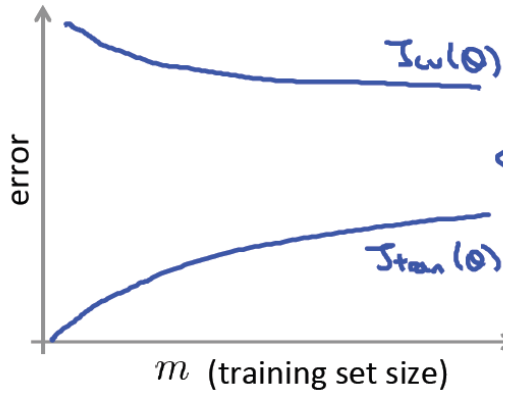
"It's not who has the best algorithm that wins.
It's who has the most data."

- و لكن مع المميزات الكبيرة ، تظل فيه عيوب موجودة :
  - أكبر و هام عقبة هي التكلفة الضخمة , الموجودة في التعامل مع عشرات الملايين من البيانات مع بعض
  - كمان الوقت الرهيب المطلوب
- فمثلا
  - لو انا عندي 100 مليون صف , هعمل gradient descent بالمعادلة ديه :

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

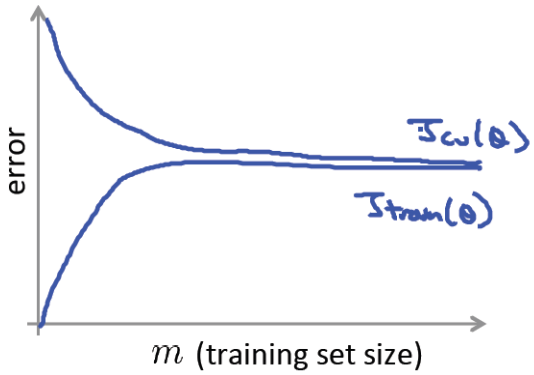
  - فحساب سميشن لـ 100 مليون صف ده رقم مذهل , و هيكون غالي جدا و بطئ
  - فمن الافكار اللي ممكن تتعمل , اني اختار بشكل عشوائي مثلا 10 الالف صف بس , و ممكن ساعتها الدقة تكون هي هي

- لو فاكر , فيه نوعين من التعامل مع الزيادة في البيانات :
  - النوع الاول , ان اضافة بيانات , تصاحبها زيادة في الكفاءة



- هتلاقي هنا ان مع اضافة اعداد اضافية من الصفوف , بيصاحبها تقليل قيمة J_{cv} يعني زيادة كفاءة العملية
- بس واضح هنا ان فيه مسافة بين J_{cv} و J_{train} وده معناه ان مع اضافة المزيد من الصفوف هتقل الـ J_{cv} اكتر , يعني هتزيد الكفاءة , فهنا لازم ازود العدد

  - بينما النوع التاني , ان اضافة المزيد من البيانات , ملهاش لازمة



- فهنا واضح ان كلا من J_{cv} و J_{train} شبه ثابتين , وزيادة صفوف مبتعملش اي حاجة , يبقي الافضل اني اقف عند الرقم ده , ومضيعش وقتي و فلوسي
- وقد يكون حل للحالة ديه ,اني اضيف features جديدة , او طبقة مخفية جديدة في الـ NN واللي هتعمل الشكل زي الشكل الاول

- و هنتعرف دلوقتي علي نوعين اساسيين , للتعامل مع الكميات الكبيرة من البيانات :
  - التدرج العشوائي Stochastic Gradient Descent
  - خريطة التقليل Map Reduce

- **التدرج العشوائي Stochastic Gradient Descent**
  - احنا بنستخدم تكنيك الـ Gradient Descent في مختلف انواع الـ ML عشان نقلل قيمة الـ J و نجيل افضل قيم للثيتا اللي تخلينا نقدر نتوقع اي ارقام جاية
  - لكن المشكلة مع الاعداد الضخمة من البيانات , بيكون صعب و مكلف جدا اننا نعمل ده , فبيكون الحل اننا نستخدم تكنيك (التدرج العشوائي Stochastic Gradient Descent) عشان يساعدنا في عمل التدرج بشكل سليم

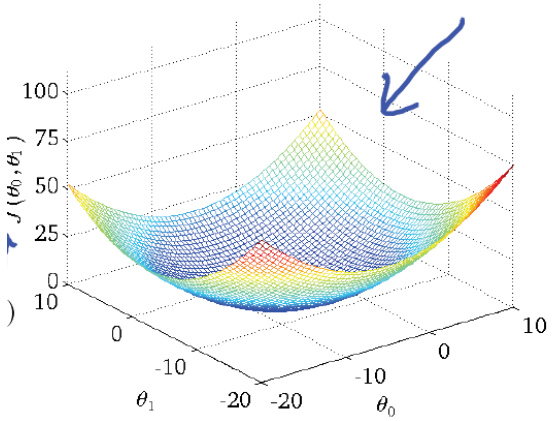
- تعالي نفتكر الـ Gradient Descent
  - دالة الإتش المتوقعة بتكون كدة :

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$$

  - بينما قيمة الـ J هتكون

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

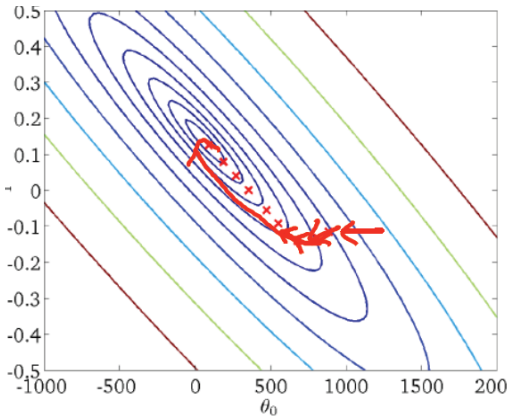
  - ساعتها الرسم هيكون كدة :



  - و الـ gradient هيكون انا نعيد الدالة ديه للوصول لاقل قيمة للـ J :

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

  - ولو فاكرين الرسم الخاص بيها , هنلاقي ان لو تم اختيار اي قيم لثيتا , فلما باعمل gradient هتتحرك قيم ثيتا تدريجيا , لغاية لما توصل للنص بالظبط , اللي فيها اقل قيمة للـ J يعني الـ global minimum

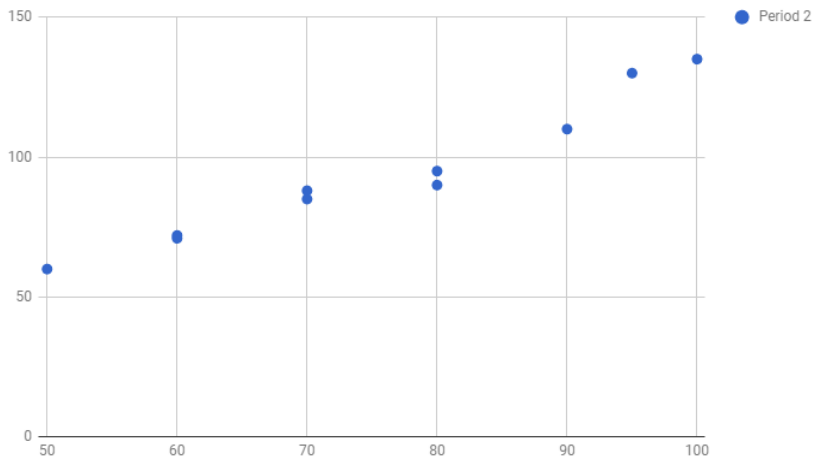


- فين بقي المشكلة :
  - المشكلة ان التعامل مع المعادلة ديه , بما فيها السمشن $\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$ (وهي اسمها Batch Gradient Descent) ده معناه وقت رهيب جدا في قراءة عشرات الملايين من الصفوف , واعمل لها سميشن , واشوف الفروق و اطرح , بعدها هعيد نفس الخطوة كذا مرة , وده معناه وقت رهيب , وتكلبة كبيرة
  - عشان كدة هنعمل فكرة , عشان نتجنب ضياع الوقت

  - قبل ما نفهم الفكرة , عايزين نعرف اصلا هو الـBatch Gradient Descent (الطريقة العادية) بيشتغل ازاي
    - بفرض ان عندي 10 بيانات زي كدة :

مساحة البيت (X_1 (م^2)	50	60	60	70	70	80	80	90	95	100
عدد الغرف X_2	1	1	1	2	2	2	3	3	3	4
السعر (الف $) Y	60	72	72	85	88	90	95	110	130	135

  - ساعتها هيكون الرسم كدة



- لاحظ ان المساحة اكس 1 , وعدد الغرف اكس 2 , بينما السعر هو واي
- عشان اعمل best fit line هنفرض الثيتات قيم معينة , وليكن ثيتا 1 = 1 و ثيتا 2 = 3
- هنيجي نحسب قيمة الإتش لكل القيم , يعني هتكون قيمة اتش 1 تساوي (1*50) + (1*3) = 53 , وهكذا

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$$

- اكرر الخطوة مع باقي البيانات , بحيث يكون عندي عشر اتشات , واحدة لكل قيمة
- ابدأ اطبق الـ BGD , واللي هيحسب فرق القيمة المتوقعة بين كل اتش(القيمة المتوقعة), و الـ Y (القيمة الحقيقية) , واعمل لها مربع و اقسمها علي ضعف الـ m

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

- يعني اول حاجة هتكون $(53\text{-}60)^2$, والتانية $(63\text{-}72)^2$, وهكذا
- اقسم مجموعهم علي 20 , هيطلعلي قيمة J مثلا بـ 20
- و عشان اظبط قيم الثيتا , هاضطر اروح للقانون ده :

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

- اني اقول ثيتا 1 (اللي هي 3) , هتساوي نفس الـ 3 , ناقص الاتي
  - الفا (نفرض انها ب1) مقسومة علي 10 , تبقي 0.1
  - مضروبة في محصلة $\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$
    - يعني (50)* (53-60) يعني سالب 350 , مجموعة علي اللي بعدها اللي هو :
    - 60 في(63-72) يعني سالب 540 , وهكذا
  - لما اجمع كل ده و اضربه في 0.1 , هيطلع مثلا رقم سالب 0.4
  - تبقي قيمة ثيتا 1 بقت 2.6
  - هعيد نفس الموضوع لثيتا 2
- كدة انا وصلت لقيمتين لثيتا 1 و 2, بدل ما كانو (3 و 1) , بقو (2.6 و 1.1)
- ساعتها ابدا العملية من الاول تاني , و اكرر نفس الخطوات , وطبعا الارقام هتتغير لان الثيتات اتغيرت
- اشوف تاني قيمة J هلاقيها مثلا 17.6 بدل 20 , معني كدة اني شغال سليم (الخطا J بيقل)
- اكرر كمان مرة بالتوالي (قيم ثيتا جديدة , اشوف قيمة J) لغاية لما الـ J تقرب جدا للصفر , ساعتها اعرف اني وصلت للثيتا السليمة

- كل اللي فات ده , كان الطريقة العادية , و مشكلته الاساسية مع العدد الكبير من البيانات (عشرات الملايين) اني هاضطر اني امسك البيانات و اعوض فيها عشان اجيب الاتش , والفرق , اكتر من مرة , وكل مرة هامشي خطوة واحدة بس في تحسين قيمة الثيتات
- فلو ان كل مرة هتاخد مني ساعة عشان يقرا 300 مليون صف , فمعني كدة اني محتاج مثلا 20 ساعة , عشان اعمل 20 خطوة لتحسين الثيتا
- فهنعمل طريقة الـ stochastic واللي هتخلينا نقرا البيانات مرة واحدة , وكل خطوة بقراها , هتعمل تعديل في الثيتا اوتوماتيك ,فبمجرد ما اخلص كل الصفوف , هلاقي ان الثيتا بقت سليمة و جاهزة

- طريقة الـ Stochastic Gradient Descent
  - اولا هنعمل دالة اسمها الكوست , وهيكون مدخلاتها 3 حاجات :
    - الثيتات (1 و 2)
    - الإكسات (1 و 2)
    - الواي

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

  - و هنقول ان قيمة الكوست هي نص مربع الفرق بينهم
  - لاحظ ان مفيش سمشين ولا m , معني كدة ان فيه كذا كوست , و كل صف ليه الكوست بتاعه
  - يعني في المثال بتاعنا , هيكون كوست 1 = $(53-60)^2$ يعني 49
  - عشان احسب قيمة J هيكون القانون

$$J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} cost(\theta, (x^{(i)}, y^{(i)}))$$

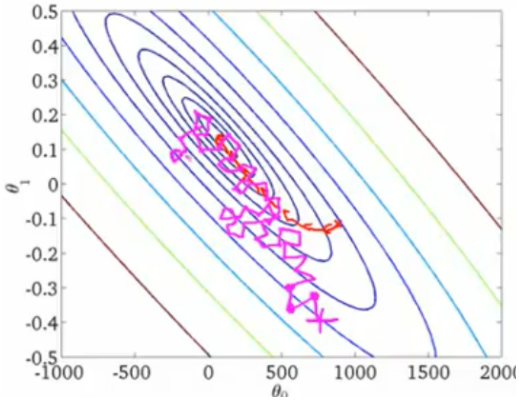
  - يعني هجمع كل الكوستات (نقول بقت 400) واقسمها علي m=10 يعني 40

- ديه مش خطوات عمل الـ Stochastic Gradient Descent , ديه بس حساب الـ J بيها , لكن خطواتها بتكون كالتالي :
  - اولا نعمل shuffle للبيانات عندنا , يعني لغبطة ليهم , عشان لو كان فيه اي نوع من الترتيب ليهم يتلغي
  - نعين قيمة الفا قول مثلا 0.01
  - القانون هو :

$$\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

  - اطبق في اول نقطة (ثيتا 1 = 3 , اتش 1 = 53 , واي 1 تساوي 60 , اكس 1 تساوي 50)

- اذن ثيتا 1 (قيمتها 3) هتساوي 3 ناقص الفا مضروبة في (53-60) مضروبة في 50 , يعني سالب 350
- امسك السالب 350 , واضربها في الفا , هتكون سالب 0.35
- لما اطرح 3 منها , هتكون 3.35
- ادخل في النقطة اللي بعدها بس بقيمة ثيتا تساوي 3.35 , يعني :
- قيم النقطة التانية (ثيتا 1 = 3.35 , اتش 1 = 63 , واي 1 تساوي 72 , اكس 1 تساوي 60)
- هاعمل تعديل لثيتا 1 , واللي مثلا هتكون 3.30
- اكرر نفس العملية للصف التالت و الرابع و هكذا
- هلاقي ان كل صف بادخله , قيمة ثيتا بتتعدل شوية شوية , ولما بادخل في الصف اللي بعده بقيمة ثيتا المتعدلة , بيكون القيمة اللي بتتضاف او بتقل اقل
- لغاية لما اوصل لنهاية الصفوف , هلاقي ان قيمة ثيتا اللي ظهرت هي افضل قيمة ليها
- نفس الكلام هاعمله لثيتا 2 (الحقيقة اني همشيهم بالتوازي مع بعض , مفيش تعارض)
- و اخيرنا , فيه 3 ملحوظات مهمة لازم يتقالو :
  - أولا اننا مش لازم نعمل الـ 300 مليون صف كلهم , عشان اساسا لما بامشي مثلا مليون صف , بلاقي ان قيم ثيتا بقت قريبة جدا من القيم السليمة , وبالتالي كل ما بازود صفوف , بلاقي ان مفيش اي تغيير في الثيتات , فبضيع وقتي علي الفاضي
  - ثانيا العكس
    - اني ممكن بعد ما اخلص كل بياناتي , الاقي اني محتاج ابدا الموضوع من الاول , يعني بدات اللوب و ثيتا 1 ب 3 , وثيتا 2 ب 5 , وخلصتها و ثيتا 1 ب 4.5 , وثيتا 2 بسالب 1
    - ساعتها ممكن ابدا من الصف الاول كمان مرة , بالثيتات الجديدة , و كل صف تتغير القيم شوية بشوية لغاية لما اوصل لقيم افضل
    - و ممكن حتي اعيد لغاية 10 مرات
  - الحاجة التالتة ,و هي متعلقة بدقة الخطوات
    - في الطريقة العادية Batch الخطوات بتبقي اكثر دقة , وبتجري مباشرة (المسار الاحمر), بينما هنا في الـ SGD الخطوات علي الجراف بتكون عشوائية شوية , وممكن اروح يمين و شمال وابعد و اقرب (الخط البمبي) , بس بشكل عام انا متجه للهدف بتاعي



- و ده ناتج الي ان , الطريقة الاولي بتجيب محصلة الثيتات , التانية بتمسك قيمة قيمة , صحيح ان الطريقة التانية شكلها ابطء , والـ steps علي الجراف اكتر , بس طبعا اسرع كتير , لان وقت عمل كل العملية في الطريقة التانية , يساوي وقت عمل خطوة واحدة من الطريقة الاولي
- و من الرسم , واضح تفسير (اولا و ثانيا) , اني من ناحية ممكن مكملش كل البيانات طالما وصلت لاقل J عندي , ومن ناحية , ممكن بعد ما اخلص كل البيانات , الاقي اني محتاج اشتغل تاني عشان لسة موصلتش

- تحديد قيم نظام الـ Stochastic
  - و يقصد بها ضبط قيمة الفا , وهي التي تقوم بتحديد خطواتها
  - كمان عايزين نعرف هل الـ stochastic بتعمل convergence (اقتراب) ولا قيمة الـ J بتزيد

  - عايزين نفتكر حاجة , ان اصلا حساب قيمة التكلفة J هي مش من ضمن خطوات الحصول علي افضل ثيتا , وقانون الحصول علي الـ J موجود بس عشان نتأكد من قيمتها
  - لما كانت القيم صغيرة , كنا بنحسب الـ J كل مرة , عشان نتاكد انها بتقل , لكن لما يكون عدد الصفوف عشرات الملايين , هتكون مشكلة كبيرة , اننا نحسب الـ J كل مرة , وده هيضيع وقت رهيب مننا

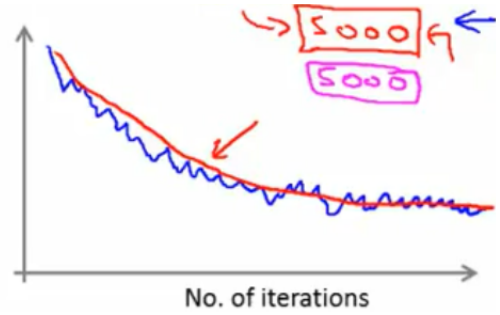
  - نفتكر ان صيغة معادلة الـ J في الـ Stochastic بتكون كدة

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

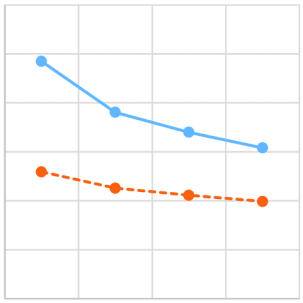
- فالخطة بتكون كالتالي :
  - في نظام الـ Stochastic انا بامشي صف صف , فلما امشي لمدة الف صف , هقف ساعتها , واحسب قيمة الـ J
  - حساب الـ J بيكون بنظام , اني احسب قيمة الالف J الاولي , واجيب المتوسط بتاعهم
  - اكرر نفس الخطوة بعد الف صف كمان
  - ابدأ أقارن بين متوسطات الـ J اللي جايبها من كل الف صف , وهيبان معايا هل الـ J بتعمل كونفيرج ولا لا
  - استخدام المتوسط عشان معتمدش علي قيم فردية ممكن تلغبط حساباتي

- طيب عايزين نتعرف علي القيم المختلفة , وتاثيرها في الـ Stochastic

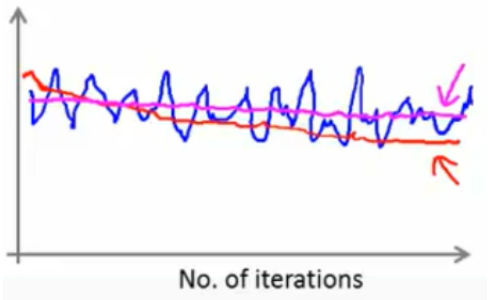
  - **أولا عدد المرات اللي بحسب بعدها قيمة J**
    - عرفنا من شوية ان ممكن كل الف صف , اعمل check علي الكوست J , لكن ماذا لو غيرنا الرقم ده , زيادة او نقصان



- متنساش اولا اني محتاج الجراف بشدة , عشان اعرف امتي اقف , لان اول ما الجراف يبدا يعمل flatter يعني استواء , فالافضل اني اقف عشان مضيعش وقت ومجهود , في حين ان الـ J مش هتقل
- كل ما كان العدد اصغر (الخط الازرق) كل ما كان الخط خشن اكتر , ولكن ادق عشان هعرف بالظبط امتي هقف
- كمان لما العدد يصغر , هعرف النتيجة بشكل اسرع يعني مثلا كل دقيقة هترسم نقطة جديدة , لكن المجموع الكلي هياخد وقت اكتر
- بينما لما الرقم يكبر (الخط الاحمر) الخط بيكون ناعم اكتر , لكن اقل دقة , كمان هحتاج وقت اكتر علي ما تطلع النتيجة واحدة واحدة
- خد بالك لو العدد كبر جدا , هيتحول الجراف لخطوط مستقيمة ورا بعض



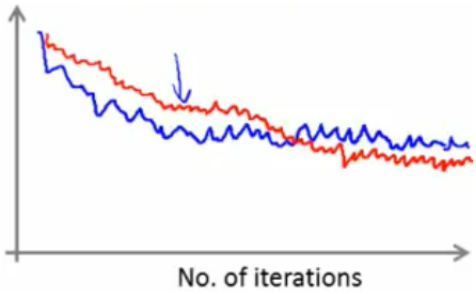
- و خد بالك انت ممكن تشوف حاجة زي كدة



- و الخط الازرق بيوضح ان الـ J بتزيد و تقل بشكل عشوائي , وديه مشكلة معناها ان الخوارزم مش قادر يتعلم حاجة
- ساعتها اقلل شوية عدد المجموعات من الف لـ 5000 مثلا
- فيا اما اقدر اوصل للخط الاحمر اللي بيقل شوية بشوية و اكمل معاه للاخر

- يا اما الاقيه زي الخط البمبي , يبقي برضه مش قادر يتعلم , ساعتها , اغير حاجة في قلب البيانات نفسها , ازود features او اغير الداتا

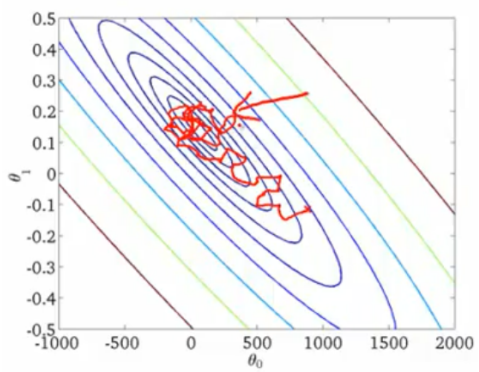
- **ثانيا قيمة ألفا** α
  - الألفا , بتكون في القانون ده $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$ يعني هي بتضرب في فرق الاتش عن الواس , مضروبة في اكس
  - لما بتزيد قيمتها بتكون قيمة تعديل الثيتا اكبر شوية , و ده سلاح ذو حدين , لانه ممكن يقربنا اكتر , او يبعدنا اكتر , بس الأكيد انه بيخلي الرسم خشن شوية
  - بينما لو قلت قيمة الالفا , بتكون الحساب ابطئ , لأن الخطوات اصغر , وبالتالي اكتر , لكن اكثر دقة
  - وده بيخلي الرسم يكون ناعم شوية , وغالبا اكثر دقة
  - هنشوف الرسم الازرق الالفا كبيرة شوية , بينما الاحمر الالفا اصغر , وغالبا بتكون قيمتها قدام شوية افضل من الالفا الاكبر



  - أما لو لقيت ان الرسم بقي كدة

  - فمعني كدة ان الألفا كبيرة , فلازم تقللها فورا

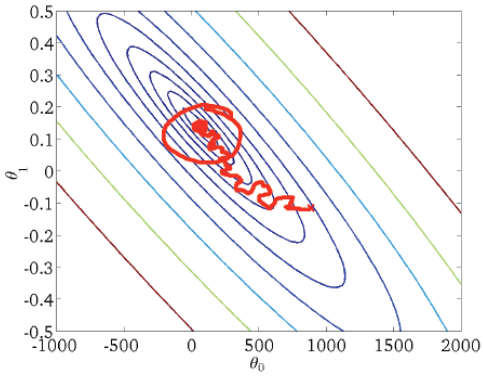
- وقيمة الفا فيها ملحوظة ذكية :
  - لو تم استخدام قيمة الفا ثابتة في كل الـ Stochastic يعني في كل الصفوف من 1 لـ 200,000,000 فده مش غلط , لكن هيخلي الحل بطئ شوية ,لان الطريق للقيمة الـ minimum هيبقي ملتوي زي كدة



- بينما لو عرفنا نعمل آلية , ان قيمة الفا تقل تدريجيا , لأن كل ما تقل , كل ما الدقة تزيد تدريجيا ,فيتم الوصول اسرع
- لكن هيتم تقليلها ازاي ؟
- فيه صيغة بتتعمل , بحيث قيمة الفا تقل تدريجيا مع زيادة عدد المحاولات , وهي :

$$\alpha = \frac{\texttt{const1}}{\texttt{iterationNumber} + \texttt{const2}}$$

- بحيث يكون فيه ثابت فوق وثابت غيره تحت , والقسمة علي عدد المحاولات هيخلي الفا تقل تدريجيا , وساعتها هيكون شكله كدة



- بعض الناس يرفضو يعملو ده , هربا من تحديد قيم الثوابت الجديدة , بس بشكل عام وجودها مفيد

- طريقة الـ Mini-Batch Gradient Descent
  - اتعرفنا علي النوع اللي فات الـ SGD و عرفنا انه افضل كتير مع القيم الكتيرة
  - دلوقتي هنعرف نوع تاني اسمه الـ Mini-Batch Gradient Descent و اللي قد يكون افضل منه في بعض الحالات

- باين من اسمها اني هاطبق نموذج مصغر من الطريقة الباتش (الطريقة العادية المعتمدة علي ايجاد مجموع مربع الفروق لكل البيانات يعني هامشي لكل البيانات كلها , وهكرر الموضوع تاني خطوة خطوة)
- بس التطبيق هيكون بشكل مختلف شوية عشان نفهمه تعالي نفتكر طريقتي الـ Batch و الـ Stochastic

- طريقة الـ Batch
  - تعتمد علي ايجاد قيمة مربعات الفروق بين الاتش و الواي , لكل الداتا (يعني هامشي علي كل الصفوف) . وفي الاخر اظبط بيها قيمة الثيتا , وارجع تاني اعيد نفس الخطوة من الاول كذا مرة

- طريقة الـ Stochastic
  - تعتمد علي اني استخدم كل صف اني اعمل تعديل لقيمة ثيتا , بحيث , علي ما اوصل لاخر صف , تكون ثيتا بالفعل وصلت للقيمة المثلي

- طريقة الـ Mini-Batch هي وسط بين الاتنين
  - بدل ما امشي زي طريقة الـ Stochastic خطوة خطوة بكل صف علي حدة , هاحدد رقم الباتش (b) واللي بيكون غالبا من 2 لـ 100 , هنقول انه هو 10
  - وبعدها هامسك اول عشر صفوف (نفس عدد الـ b) و اجيب مربعات الفروق زي طريقة الباتش
  - و من القيمة الصغيرة ديه , هاعدل قيمة ثيتا
  - بعدها ادخل علي العشر صفوف اللي بعدهم , واجيب قيمة مربعات الفروق تاني , وبيها اعدل قيمة ثيتا
  - اكرر نفس الموضوع لغاية لما البيانات عندي تخلص
  - وكاني بامسك البيانات باقسمها علي اساس عندي قيمة الـ b بكام , و كل مجموعة فيهم اعتبرها باتش صغيرة , زي كدة

- وكأن لو عندي الف صف , وهحدد قيمة الـ b بـ 10 , هيكون الكود كدة :

Say $b = 10, m = 1000$.

Repeat {

for $i = 1, 11, 21, 31, \ldots, 991$ {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

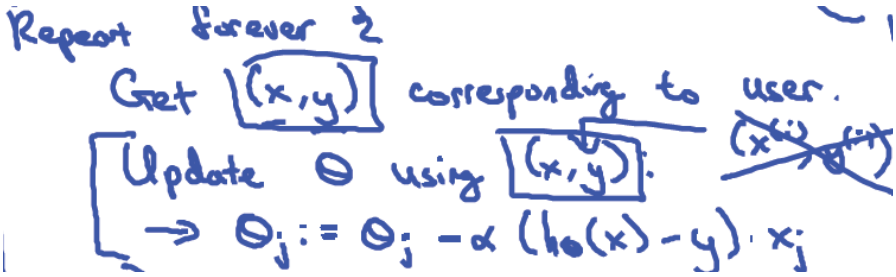
(for every $j = 0, \ldots, n$)

- اني هاعمل خطوات , كل خطوة بـ 10 , وهيكون اخر رقم عندي 991 , عشان يقفل عند الالف , و كل مرة اعمل باتش صغير , والثيتا بتتعدل هنا او هنا
- وكأن الفرق الاساسي بين الـ mini batch و بين الـ stochastic اني في الـ stochastic باعمل تعديل للثيتا صف صف , بينما في الـ mb باعمل تعديل ليه كل b صف, وكان الـ stochastic هي تطبيق للـ mb لما b=1

- التعليم الأونلاين Online Learning
  - و المقصود بيه , انك تقدر تستفيد من البيانات اللي بتجيلك بشكل مستمر و متزايد , في المواقع و غيرها, عشان تقدر تعدل من تعاملك , و تتخذ قرارات افضل
  - فالفرق الأساسي , اني مش بتعامل مع بيانات ثابتة , لكن متغيرة و متزايدة

- لو هناخد مثال
  - عندك شركة شحن , فالمستخدم بيدخل تفاصيل الشحنة , ومن فين لفين , فالسيستم بيحسب السعر اوتوماتيك
  - احيانا بيكون السعر غالبا فالمشتري اللي عمل بحث مش بيكمل العملية , واحيانا بيكمل
  - عايزين نستخدم البيانات ديه عشان نحدد السعر المناسب اللي السيستم هيظهره , من غير ما يتسبب في رفض المستخدم
  - و البيانات بتكون كالتالي :
    - وزن الشحنة :
    - المسافة بين المدينتين :
    - يوم الارسال :
    - معلومات ديموجرافيك عن المستخدم :
    - السعر :
    - هل المستخدم هيشترك ولا لا :
  - كل المعلومات الحمرا تعتبر اكسات (features), بينما الخضرا هي واي , (يشتري y=1 , ميشتريش y=0)

- عايزين نعمل خوارزم , بحيث يرسم علاقة بين اكس السعر , و واي هيشتري ولا لا , عشان من خلالها اقدر احدد السعر المناسب
- و الخوارزم هيكون كالتالي :



  - اني باخلي الثيتا دايما , يتم ظبطها عن طريق قيمة x , y اللي بتيجي من المستخدمين , وده ميكونش لعدد معين لكن repeat forever
  - طبعا مش مقصود بيها تكون infinite loop لكن ممكن كل كام ساعة اعمل update بالبيانات اللي عندي
  - لاحظ ان مش مكتوب X_i او Y_i لكن X , Y و ده عشان انا اصلا معنديش dataset باشتغل منها , وصفوف شغال واحد ورا التاني , لكن اي صف بيجيلي , اخده اعدل بيه القيمة و ارميه فورا
  - كمان فيه ميزة كبيرة للنظام ده , ان عنده القدرة انه يواؤم المتغيرات :
    - يعني لو ان الـ trend دلوقتي ان اغلب اللي عايزين يشحنو , مهتمين جدا بالوقت الخاص بالشحن , ومش هاممهم الفلوس , وقتها الخوارزم فورا هيحس بده , وهيعمل ثيتا كبيرة للإكس الخاصة بوقت الشحن , وثيتا صغير للسعر , بينما بعد كام سنة لو كان الاولي للناس هو السعر , وقتها هتزيد ثيتا السعر و تقل باقي الثيتات
    - كمان الموضوع ممكن يتقسم demographic يعني يبان ان الفئة A بتهتم اكتر بالسعر , فيتعمل لها ثيتات خاصة بيها , بينما الفئة B بتهتم بالـ safety فتكون الثيتات عندهم مختلفة

- مثال ثاني هو محركات البحث
  - وهو المقصود بيه الية اختيار اي عناصر بالتجديد من وسط الالاف , اللي هيناسب المستخدم اللي بيبحث عن حاجة معينة
  - الكلام ده سواء في موقع بحث عام , او في موقع ببيع فيه حاجات معينة , والمستخجم بيدور علي منتج و كتب اسمه و عندي منه بدائل كتير (كتب مثلا cell phone فهعرض له مين الاول)
  - و كان الغرض حاليا , اني اخلي محرك البحث يتعلم الالية اللي يقدر من خلالها يطور نفسه اول بال

- نبدا نفكر في نقاط الخوارزم (الاكسات و الواي)
  - هنقول ان الإكسات اكتر من حاجة متعلقة باللي بحث عنه , زي :
    - اسم تليفون معين لو كتبه

- فيه كلم كلمة كتبها متطابقة مع اللي عندي
- فيه كام صفة من اللي بيبحث عنها موجودة في التليفون
- صفات ديموجرافيك لو امكن

- بينما الـ y هي هل المستخدم هيدوس علي اللينك عشان يشوفه ولا لا
- و دلوقتي هنجيب احتمالية لكل منتج من اللي عندي P و اعرضهم من الكبير للصغير , يعني اعرض المنتجات اللي احتمالية انه يدوس عليها كبيرة
- العملية ديه بيسموها في الويب predictive click through rate او CTR يعني التنبؤ هل المستخدم هيدوس كليك ولا لا
- لاحظ ان كل ما مستخدم يدوس علي منتج سواء فوق او تحت , فده بيعملي تعديل طفيف في قيمة ثيتا , كمان لما مستخدم يتجاهل الحاجات اللي فوق , فده برضه بيعملي تعديل في الثيتا , بحيث تدريجيا اقدر احدد كلمة كذا تخليني اختار ايه
- يعني لما الاقي ان اغلب اللي كتبو cell phone with good gps اختارو تليفون سامسونج نوت 8 , ساعتها الثيتا في التليفون ده الخاصة بكلمة gps هتزيد , و هتظهر دايما , بينما لما اغلب اللي كتبو good camera مختاروش تليفون هاواوي , ولو اغلب اللي كتبو cheap phone اختارو هواوي , يبقي الثيتا الخاصة بالكاميرا في تليفون هواواي , هتقل , وبالتالي مش هيظهر للي بيبحثو عنه , و الثيتا الخاصة بالـ cheap هتزيد و هتظهر

- **خريطة التقليل وتوازي البيانات Map Reduce and Data Parallelism**
  - كل اللي فات كان النوع الأل من انواع التعامل مع البيانات الضخمة , وهي الطريقة العشوائية
  - هنتعرف حالا علي طريقة خريطة التقليل
  - و الفكرة الاساسية ليها , هو اني باستخدم اكتر من جهاز كومبيوتر لتطبيق الخوارزم , و ده طبعا بيسرع جدا من عملية ايجاد القيم المثلي
  - كون اننا هنشرحها قليل , فده مش معناه قلة اهمية لها , ولكن لانها بسيطة و مفيهاش تفاصيل كتير , لكن اهميتها بنفس اهمية باقي الابواب اللي درسناها , ان مكانش اكتر

- فكرته
  - غالبا بيتم تطبيق الفكرة ديه , علي اسلوب قابل لتجزيء المهام , واهمهم نوع الـ Batch Gradient Descent
  - قانون الـ BGD بيكون كدة

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

  - واللي فيه , احنا بنحسب قيمة اكس , في فرق الاتش من الاكس , و نعمل مجموع ده من صف 1 لاخر صف , وبعدها نضربها في الفا علي ام

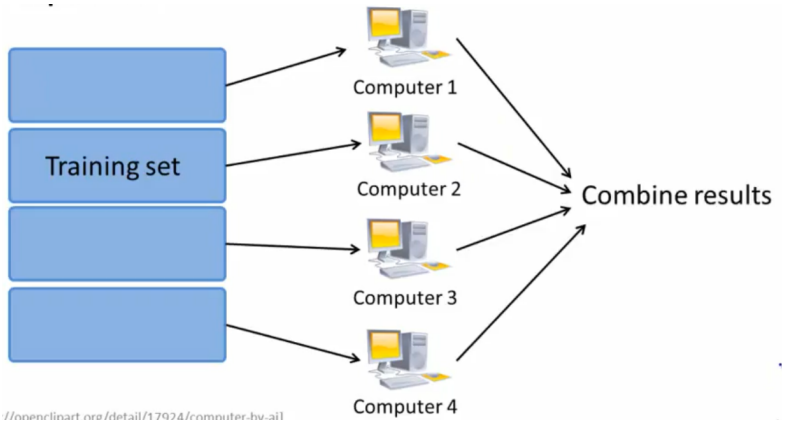
- و اكتر حاجة بتاخد وقت , هو حساب $(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$ خاصة لما عدد الصفوف يكون مثلا 200 مليون
- فاللي هنعمله كالتالي , لو عندي 10 اجهزة كومبيوتر , و عايز احسب القيمة ديه لـ 200 مليون صف , فهوزرعهم علي الاجهزة كلها
- بحيث الجهاز الاول يحسب قيمة $(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$ من 1 لـ 200,000 , والتاني من 200,001 لـ 400,000 وهكذا
- لو كان حساب ال200 مليون بياخد من جهاز واحد ربع ساعة , فحساب 200 الف هياخد دقيقة و نص , و بالتالي خلال الدقيقة ونص هتلاقي كل الاجهزة جابت قيمها
- اخيرا اجمع القيم اللي العشر اجهزة جابتهم , و اضربها في الفا و اقسمها علي 200 مليون (m) و اطرحها من ثيتا
- كدة انا عملت خطوة واحدة في تطوير الثيتا , اعيد نفس الموضوع , علي العشر اجهزة بنفس الطريقة
- فكدة انا اختصرت الوقت للعشر , عن طريق توزيع المهام علي عشر اجهزة
- لاحظ ان مش اي طريق ينفع اقسمها , يعني مثلا الـ stochastic انا بامشي صف صف , وباعدل الثيتا كل خطوة , فمش هاقدر اعمل ده علي 20 اجهزة
- وكأن لازم المهمة اساسا يبقي تنفع انها تتقسم لمهام متوازية , مش مهام متوالية , عشان اقدر اطبق عليها الفكرة ديه
- زي مثلا تطبيقات الـ logistic regression اللي فيها log
- اللي فيها الكوست كدة

$$J_{train}(\theta) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

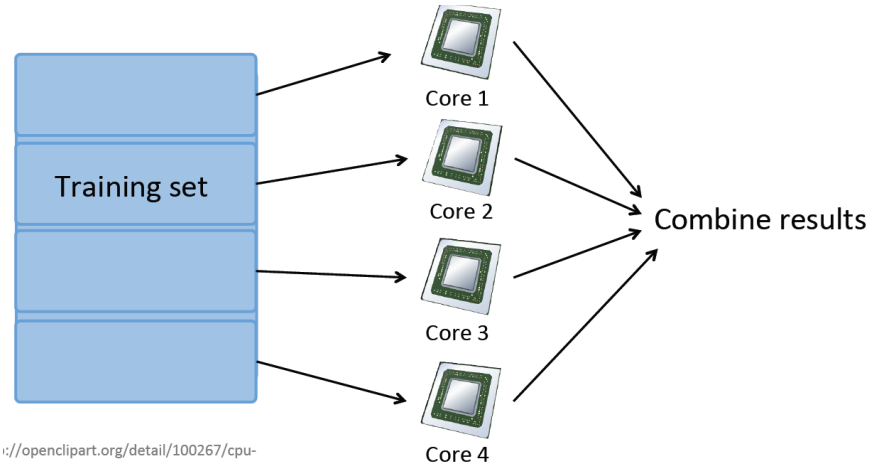
- و التفاضل الجزئي بتاعها كدة

$$\frac{\partial}{\partial \theta_j} J_{train}(\theta) = \frac{1}{m}\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

- و كان الفكرة اني باقسم البيانات عندي لاربع اقسام , وكل قسم يعمله جهاز , ويتجمع في الاخر



- ولاحظ ان تقسيم المهام علي اكتر من جهاز , بيتم عبر اني اعمل كذا جهاز حقيقي , وبينهم نيتورك , او نفس الجهاز بس يكون فيه اكتر من بروسيسور



- وطبعا النوع ده ميزته ان مفيهوش مشكلة بطئ النيتورك الموجود في النوع الاول
