بسم الله الرحمن الرحيم

في المحاضره ديه هنتكلم علي ازاي نقدر نعمل generalization لل perceptron لأكتر من layer وهنقول كمان الباك بروب بيشتغل ازاي و ايه هي ال important optimizers اللي موجوده في التنسور فلوو مثلاً ... هنراجع شويه علي اللي احنا قلناه علي ال Linear models و ال non-linear models .. أول حاجه هنتكلم عن اللينير ريجريشن .. واستهخدمنا الموديل اللي هو y = w.T * X .. لو جينا لل linear classification كان في 3 موديليز .. اول واحد هو ال Mixture of Gaussian و ده كنا بنعرف فيه ال prior اللي هو ال pi_c = p(class) .. و ده كان multinomial .. وكمان بنعرف ال likelihood of each class ده بيبقا Gaussian distribution و بعدين بنحسب ال posterior لكل كلاس .. وده اللي هو p(c|x) = k * p(c) * P(x|c)

مشكلة ال mixture of Gaussian ان عندك افتراض ان ال CCD هي gaussians .. فعملنا relaxation للافتراض ده و رحنا ناحية ال logistic regression .. فبالنسبه للوجستيك أول حاجه شفناها هي ال binary case .. و ديه ليها الاحتمالية p(c|x) = sigmoid(w.T x) و لو عندنا multiclass case .. هتبقا p(c_k|x) = e^(w_k x) / (summation over j for all w_j.T x)

بعد كدا دخلنا في البرسيبترون ولو استخدمنا ال threshold activation function في الحاله ديه ال y = sign(w.T x) .. لو استخدمنا السجمويد اكتفيشن فانكشن ... p(y) = sigmoid(w.T x) ...

دلوقت احنا كنا اتكلمنا علي طرق نقدر ن extend بيها ال linear models اللي عندنا .. عشان نقدر يبقا عندنا functions that are not stragight lines بتبقا كرفات و بالتالي عندك non-linear models ...

احنا عملنا ده للاتنين classification and regression .. فهنكتب ملخص سريع كدا للي احنا عملناه و الافكار اللي احنا هنشوفها و نتوسع فيها علي مدار بقيت الكورس ... "انا تعبت من كتر الأفكار" .. "انشف كدا دحنا لسه بنلعب بال linear regression هه عمو bayes بيشاورلك من بعيد" ..

أول حاجه هي ال non-linear regression .. y = w.T phi(x) .. و هنا الفكره انك بتبدل الإكس ب فاي أوف إكس .. و هنا اانت بتعمل mapping للفكره بتاعتك لل new space .. طالما ال mapping non linear حتي لو انت بتعمل linear classification في ال new space انت فعلياً non-linear في ال original space ... فديه كانت الفكره اللي عملناها لما احتجنا نعمل non-linear regression ...

اللي هنحتاج نعملو برضو هو ال Multi-layer neural networks ... مش هنكتب معادلات دلوقت .. بدل ما افترض a certain basis function هيبقا عندنا adaptive basis functions ...

هنتكلم دلوقت علي ال non-linear classification .. اول حاجه هي كانت ال generalized logsitc regression ... انت عندك binary class .. فهتقول p(c|x) = sigmoid(w.T * phi(x)) .. لو multi-class .. p(c_k | x) = e^(w_k phi(x)) / (summation over j for all w_j.T phi(x))

ال generalized perceptron هيبقا y = sign(w.T phi(x)) .. و p(y) = sigmoid(w.T phi(x)) ... فبالنسبه لل non-linear models كدا انت دايماً بتشيل ال x و تحط مكانها ال basis functions اللي non-linear .. وكلو يقدر يروح لل approach ده .... بس هنا في مشكله ازاي نختار ال phi ..

Non-linear Models

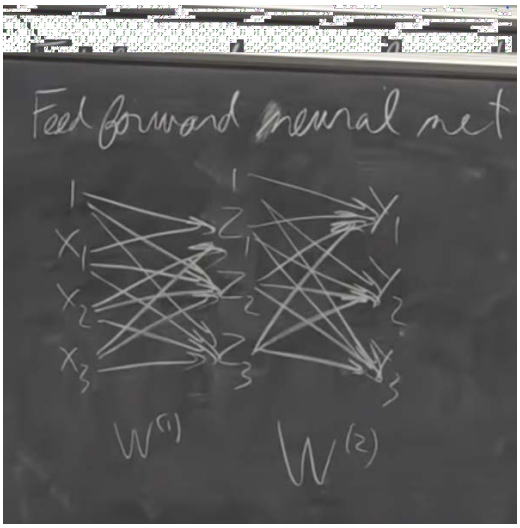
- **Convenient modeling assumption:** linearity
- **Extension:** non-linearity can be obtained by mapping $\boldsymbol{x}$ to a non-linear feature space $\phi(\boldsymbol{x})$
- **Limit:** the basis functions $\phi_i(\boldsymbol{x})$ are chosen a priori and are fixed

- **Question:** can we work with unrestricted non-linear models?

هل نقدر نشتغل ب unrestricted non-linear models؟ .. هل لازم اكون معرّف ال basis functions a prior .. الاجابه تقدر انك تشتغل اه .. عندك فكرتين .. اول واحده بدل ما يكون عندنا fixed basis .. اختارها علي حسب الداتا .. وحط قدامك basis functions كتيره .. و ده هنلاقيه في ال SVM ..

الحل التاني انك تستخدم multiple layers زي ال deep learning ... استخدم multi-layers .. ال basis مش ثابته .. انت هت adabpt them .. و هيبقا عندك فانكشنز عددهم ثابت انما هم بي adapt للترينج داتا ...

تعال ناخد ال feed forward neural net ..



Two-Layer Architecture

- Feed-forward neural network

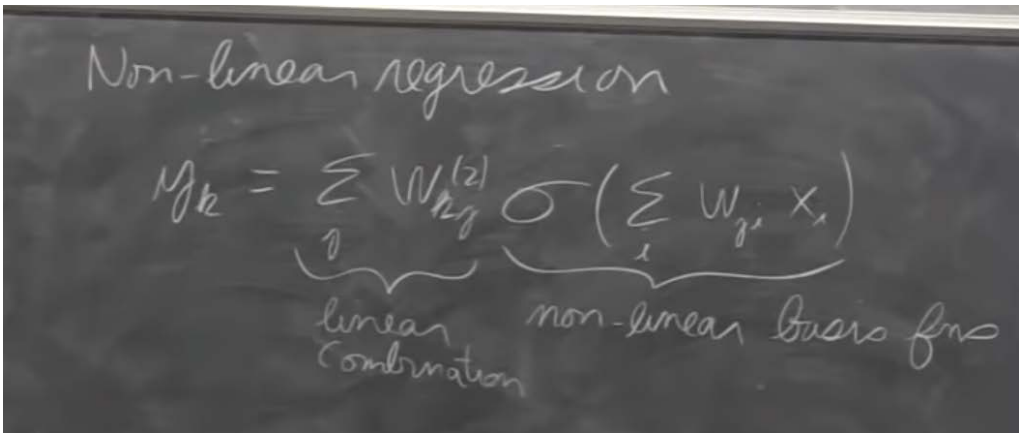
- Hidden units: $z_j = h_1(\boldsymbol{w}_j^{(1)}\bar{\boldsymbol{x}})$ — w_j → jth row; column vector
- Output units: $y_k = h_2(\boldsymbol{w}_k^{(2)}\bar{\boldsymbol{z}})$ — kth row
- Overall: $y_k = h_2\left(\sum_j w_{kj}^{(2)} h_1\left(\sum_i w_{ji}^{(1)} x_i\right)\right)$

المهم ان الأكتفيشنز اللي common أوي هم اللي عندك في الصوره تحت

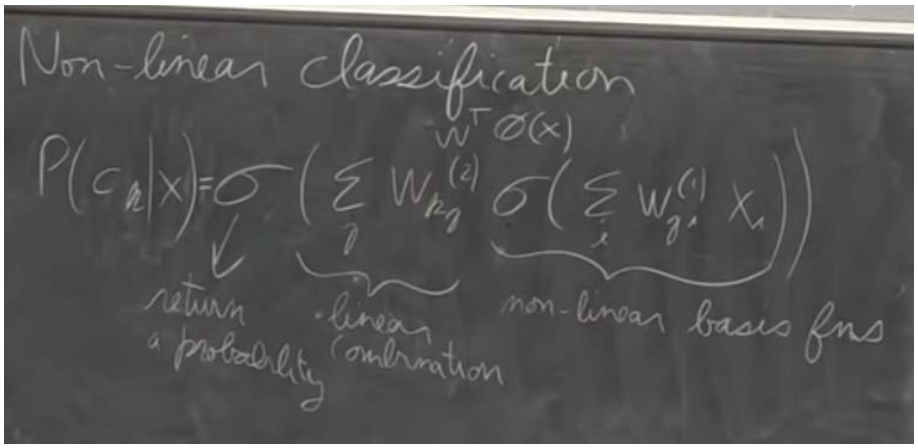
Common activation functions h

- Threshold: $h(a) = \begin{cases} 1 & a \geq 0 \\ -1 & a < 0 \end{cases}$
- Sigmoid: $h(a) = \sigma(a) = \frac{1}{1+e^{-a}}$
- Gaussian: $h(a) = e^{-\frac{1}{2}\left(\frac{a-\mu}{\sigma}\right)^2}$
- Tanh: $h(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$
- Identity: $h(a) = a$

الدكتور بيرسم ال tanh علي الصبوره .. شبه ال sigmoid بس بتروح من 1 ل -1 .. فدلوقت لما هنعمل non-linear regression مع neural network ليها 2 layers اللي تقدر تعملو انك تستخدم non-linear function لل h1 .. و بعدين خلاص كدا استخدم ال identity .. فلل non-linear regression .. هتقول ان ال y_k = Summation over j for (w_k_j * sigmoid(Summation over I for Wji xi)



كدا انت عندك ال sigmoid function in weights في الحاله ديه انت ال sigmoid مش ثابته انما ال sigmoid هنا بت adapt للترينج داتا .. نفس الكلام ممكن نعملو علي ال Non-Linear classification ... فانت تقدر تحسب p(c_k|x)



طيب دلوقت احنا فهمنا ازاي نستخدم 2 لايرز .. عشان نعمل النان لينارتي .. هنا السؤال ازاي نعمل optimization ... في الجورذمز كتيره ... اللي المعظم بيستخدموه هو

Weight training

- Parameters: $< \boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \ldots >$
- Objectives:
  - **Error minimization**
    - **Backpropagation (aka "backprop")**
  - Maximum likelihood
  - Maximum a posteriori
  - Bayesian learning

ده بيخليك تاخد ال errors اللي بتحسبها و ت back prop وتحسب ال gradients لكل الويتس ... برضو تقدر تستخدم بقيت الطرق بس دول مش مشهورين ومش بيسكيلو جامد ..

Least squared error

objective → minimize squared loss

- Error function

Neural Network

$$E(\boldsymbol{W}) = \frac{1}{2}\sum_n E_n(\boldsymbol{W})^2 = \frac{1}{2}\sum_n ||f(\boldsymbol{x_n}, \boldsymbol{W}) - y_n||_2^2$$

- When $f(\boldsymbol{x}, \boldsymbol{W}) = \underbrace{\sum_j w_{kj}^{(2)}}_{\text{Linear combo}} \underbrace{\sigma\left(\sum_i w_{ji}^{(1)} x_i\right)}_{\text{Non-linear basis functions}}$

then we are optimizing a linear combination of non-linear basis functions

تعال بقا نحسب الجريدنت وناخد خطوه هناك .. ده مش بيبقا سهل انك تعزل ال weights لما تحط الجريدينت بصفر .. عشان تحسب الجريدينت هل في طريقه سهله ؟ .... الدكتور بيقول الاجابه فعلاً هي الباك بروب ... انما احنا مش بنعمل كدا دلوقت احنا دلوقت بنعمل automatic differentiation

Sequential Gradient Descent

- For each example $(\boldsymbol{x}_n, y_n)$ adjust the weights as follows:

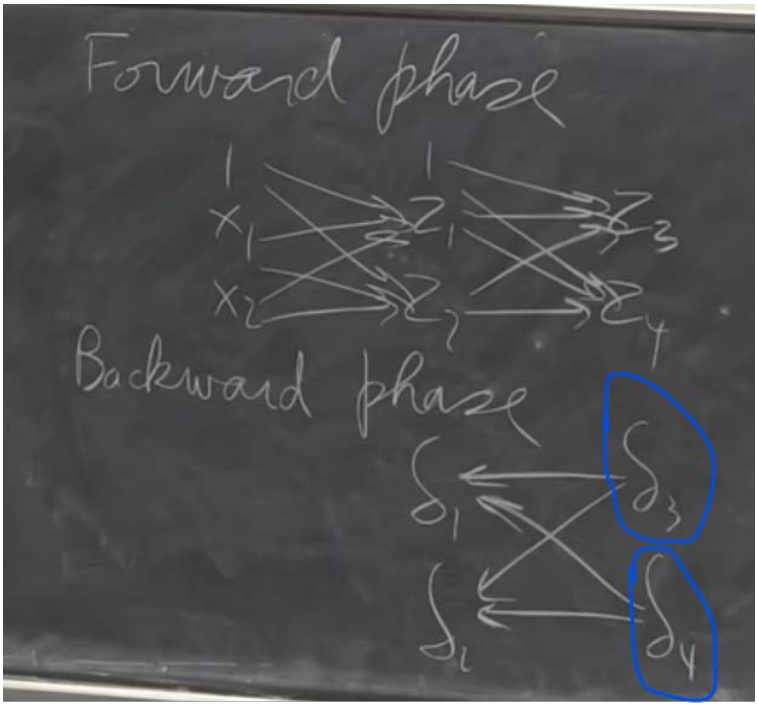
$$w_{ji} \leftarrow w_{ji} - \eta \frac{\partial E_n}{\partial w_{ji}}$$

- How can we compute the gradient efficiently given an arbitrary network structure?
- Answer: **backpropagation algorithm**
- Today: **automatic differentiation**

اللي هو مش بنشتغل بإيدينا يعني .. في بقيت السلايدز هنشوف ازاي بنحسبها بإيدينا .. عشان نفهم بس ايه اللي بيحصل تحت ال API .. بس فعلياً عندك تنسور فلو و باي تورش انك تستخدم ال automatic differentiation ... هنشوف ازاي نحسب الحوار ده بإيدينا لمثال واحد ... الباك بروب نفسو بيشتغل علي مرحلتين .. اول مرحله وانت رايح عشان تحسب ال estimations و بعدها تقارن بالتارجت .. تحسب الإيرور .. و تاخد الإيرور ده وترجع بيه .. الايرور اسمو دلتا يعني .. ده اللي هيرجع معاك و ده هيخليك تحسب gradient لكل ويت .. تعال نرسم صوره عشان نتنيل نفهم :")

" انا مكتئب لو انت بتقرا المحاضره ديه خش ابعتلي ميل بحاجه تفرحني "muhammedmahmoud063@gmail.com

المهم .. يلا نشوف الفورود فيز .. هبدأ من الإنبوت و اروح للأوتبوت .. زي ما الدكتور بيرسم كدا دلوقت ... وانت في Backward phase .. هتبدأ من الآخر .. وهوب ترجع للأول ..



Forward phase

- Propagate inputs forward to compute the output of each unit
- Output z_j at unit j:

$$z_j = h(a_j) \quad \text{where} \quad a_j = \sum_i w_{ji} z_i$$

المشكله مش وانت رايح .. المشكله وانت راجع ..

Backward phase

- Use chain rule to recursively compute gradient
  - For each weight w_{ji}: $\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$
  - Let $\delta_j \equiv \frac{\partial E_n}{\partial a_j}$ then

$$\delta_j = \begin{cases} h'(a_j)(z_j - y_j) & \text{base case: } j \text{ is an output unit} \\ h'(a_j)\sum_k w_{kj}\delta_k & \text{recursion: } j \text{ is a hidden unit} \end{cases}$$

  - Since $a_j = \sum_i w_{ji} z_i$ then $\frac{\partial a_j}{\partial w_{ji}} = z_i$

linear combination of the deltas

تعال نشوف مثال

Simple Example

- Consider a network with two layers:
  - Hidden nodes: $h(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$
    - Tip: $tanh'(a) = 1 - (tanh(a))^2$
  - Output node: $h(a) = a$

- Objective: squared error

Simple Example

- Forward propagation:
  - Hidden units: $a_j = \sum_i w_{ji} x_i$ $\quad z_j = \tanh(a_j)$
  - Output units: $a_k = \sum_i w_{ki} x_i$ $\quad z_k = \text{---}(a_k)$
- Backward propagation:
  - Output units: $\delta_k = z_k - y_k$
  - Hidden units: $\delta_j = (1 - z_j)^2 \left(\sum_{k=1}^{K} w_{kj} \delta_k\right)$
- Gradients:
  - Hidden layers: $\frac{\partial E_n}{\partial w_{ji}} =$
  - Output layer: $\frac{\partial E_n}{\partial w_{kj}} =$

انت عندك الدلتات .. فاضلك الجردينت بس ...

Simple Example

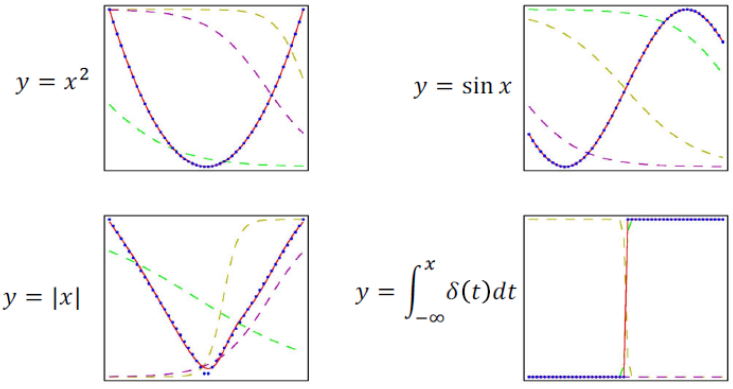
- Forward propagation:
  - Hidden units: $a_j = \sum_i w_{ji} x_i$ $\quad z_j = \tanh(a_j)$
  - Output units: $a_k = \sum_i w_{ki} x_i$ $\quad z_k = \text{---}(a_k)$
- Backward propagation:
  - Output units: $\delta_k = z_k - y_k$
  - Hidden units: $\delta_j = (1 - z_j^2)\left(\sum_{k=1}^{K} w_{kj} \delta_k\right)$
- Gradients:
  - Hidden layers: $\frac{\partial E_n}{\partial w_{ji}} = \delta_j x_i = \left[1 - (z_j)^2\right] \sum_{k=1}^{K} w_{kj} \delta_k x_i$
  - Output layer: $\frac{\partial E_n}{\partial w_{kj}} = \delta_k z_j = (z_k - y_k) z_j$

ده بيتم علي طول في الباكيدجز .. اللي هي ال automatic differentiation ... انت تقدر تغير النتورك و بعدين ت call API و خلاص علي طو ل... بم طخ .. السؤال دلوقت how well can we approximate different types of functions؟ ... علي الاقل لو عندنا 2 لايرز .. فتقدر ت approximate أي فانكشن .. فحتي لو معندناش عدد كبير اوي .. هنقدر برضو نشتغل عادي

بص ع الامثله ديه

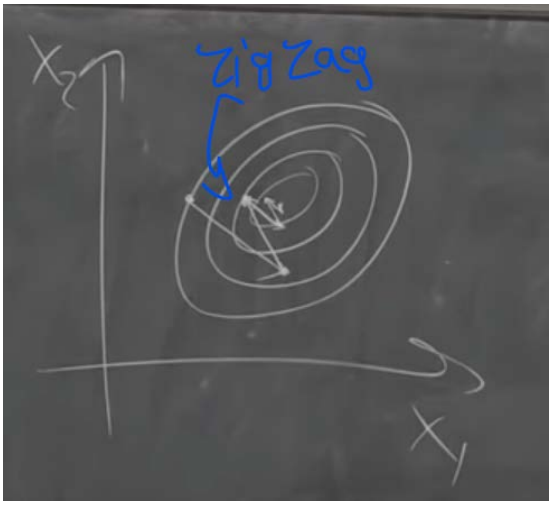
Non-linear regression examples

- Two layer network:
  - 3 tanh hidden units and 1 identity output unit



واحده من الفيتشرز المهمه لل NN هم مش one fixed model .. انما تقدر تغير كتير في الآركتكتشر .. انت عندك طريقه مبنيه علي حساب الجريدينت و يعتبر الطريقه الأحسن انك تعمل SGD .. دلوقت لو هتعمل SGD انت تقدر ت analyze الطريقه ديه وتيجي تبص علي طريقه حساب الجريدينت هتلاقيها سريعه .. بغض النظر عن ال architecture .. ال SGD ال ragte of convergence بيبقا بطيء ... لو عاوز حاجه زي Networn's method كدا سريعه فهي سريعه عشان عندك Quadratic convergence rate .. بس عيب الطريقه ديه بتحسب الجريدينت و تحسب ال hessian و في مشاكل في ال scalalbility عشان عندك بارمترز كتيره جدا .. فكل ما العدد بتاع البارمترز زاد كل ما حصل مشاكل في ال scalability .. في ال NN غالبا بيبقا عندنا non-convex optimization فبيبقا عندنا vally with so many deep points واحنا عاوزين نلاقي النقطه اللي ليها اقل ايرور .. بس الألجورذم ممكن يكونفرج لحاجه shallow و مي converges .. مشكله تانيه وهي ال overfitting عشان تمنع الأوفر فيتنج استخدم داتا كتيره .. او استخدم ال regularization وانت بتحط penalty term ..

بالنسبه لمشكلة ال slow convergence .. تعال نشوف ايه اللي بيحصل ... تعال نتكلم علي مشكله في ال 2-D .. عندنا surface اللي مرسوم هو contour lines والمنيمم هو اللي في النص .. لو احنا بنعمل gradient decsent تعال نبدأ من النقطه اللي الدكتور حطها ... GD هتاخد خطوه في الاتجاه الاقل .. والاتجاه ده هتلاقيه عمودي علي ال controur .. خد بالك انو الحته ديه ال slope فيها steep ..



مشكلة الزجزاج .. ده مش behavior محبب .. وليه طرق اننا نبعد عن حاجه زي كدا .. والحاجه ديه هو الدايركشن بتاع الجريدينت .. ال GD بيبقا ليه مشاكل الزجزاج ... فبالتالي هل فيه حاجه نقدر نعدّلها .. ؟ فالمشكله هي .. منين ما بيبقا عندي large gradient انت ممكن يحصلك overshooting ... لو خدت خطوات صغيره ... هتلاقي انك مش عرضه لل overshooting .. فده بيقولك ان هيكون في surfaces وانت في مناطق ال gradient صغير ف ال surface بيبقا stable فاتحرك اسرع شويه وخد خطوه كبيره عادي ... لو عندك big gradient مينفعش تاخد خطوه كبيره .. فانت متقدرش تعتمد علي قيمة الجريدينت و هتحتاج تعدل الخطوات اللي بتتاخد .. المهم عشان تتعامل مع حاجه زي كدا .. في فكره اسمها AdaGrad .. ده ألجورذم والهدف انو يعدّل ال learning rate of each dimension علي حدي .. يعني ساعات بيبقا واحد و ساعات بيبقا كسور .. وده بيعتمد علي اني لو شفت بارشل ديرفتيف كتير في الماضي .. خلاص متاخدش خطوات كتيره .. فبالتالي انا هقسم علي جذرال rt ..

Adaptive Gradients

- Idea: adjust the learning rate of each dimension separately
- **AdaGrad:**

r_t always increasing

$$r_t \leftarrow r_{t-1} + \left(\frac{\partial E_n}{\partial w_{ji}}\right)^2 \quad \text{(sum of squares of partial derivative)}$$

+ve

$$w_{ji} \leftarrow w_{ji} - \frac{\eta}{\sqrt{r_t}} \frac{\partial E_n}{\partial w_{ji}} \quad \text{(update rule)}$$

- Problem: learning rate $\frac{\eta}{\sqrt{r_t}}$ decays too quickly

مشكله ال AdaGrad .. ان دايماً ال rt بتزيد فعشان كدا الليرننج ريت ال decays too quickly ... في طريقه تانيه اسمها .. RMSprop .. بتعمل weighted combination .. بس ديه الفكره .. مشكلتها انها معندهاش momenturm .. معناها انك لو في منطقه ال gradient ثابت .. زي مثلا انك ف عربيه و ماشي في طريق مستقيم .. هوب ما تيلا بينا ندوس ع البنزين بسرعه .. هنا بقا مفيش حاجه بتعمل كدا .. فبالتالي ديه مشكلة ال RMSprop .. مفيش حاجه بتزود ال step size لما ال direction يبقا ثابت .. ومن هنا دخل Adam algorithm . خد weighted moving average of the gradient itself .. وفي الأبديت .. خد خطوه في الدايركشن بتاع ال moving average .. المشكله احنا مش فاهمين امتي بيشتغل او مش بيشتغل . في مشاكل في ال proofs اللي طلعت .. فاحتاجو يعملو عليها تعديلات ... هي حاجه مهمه و مشهوره جداً الحقيقه ..