بسم الله الرحمن الرحيم

انهارده هنتكلم علي الباك بروب .. احنا فجأه في مكان هنحسب ال gradients بالنسبه لأي differentiable function .. ولما هنعمل كدا .. ال paradigm بتاع ال learning as function approximation هيبان هو عباره عن ايه .... لإن دلوقت انت تقدر تكتب ال objective و تكتب complicated parameterized function وتستخدم optimization عشان تتعلم parameters كويسه لل objective function ... تعال نشوف لما كنا بنتكلم علي اختيار ال architecture بتاع النويرال نتورك كان قدامنا ايه ..

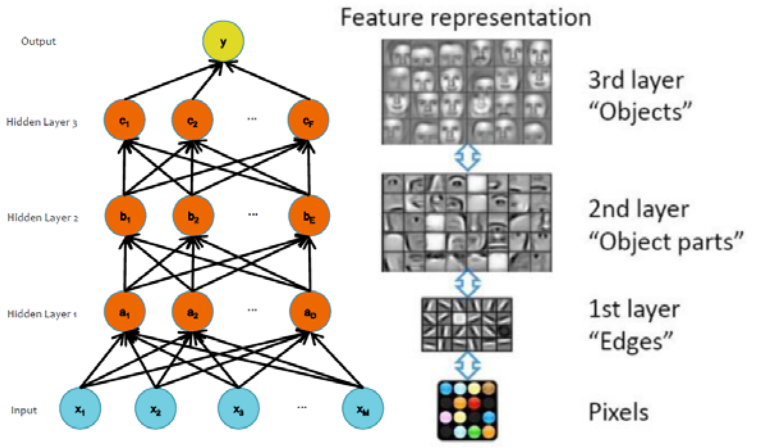
Neural Network Architectures

Even for a basic Neural Network, there are many design decisions to make:

1. # of hidden layers (depth)
2. # of units per hidden layer (width)
3. Type of activation function (nonlinearity)
4. Form of objective function

الدكتور بيقول ان في سبب ممكن يخليك انك تفكر في layers اكتر .. وهو انك فعلا عارف ان النتورك بتاعتك هتعمل حاجه احسن من اللي انت بتقدر تعملو عشان تقدر تجيب level of abstraction معين .. فافتكر في ال feature engineering ... قصة ان الcomputer vision بيقدر يطلعلك features of image .. بس ايه اللي هيحصل لو هو اللي عملو ده كان غلط ومش شغال ... وكان الطريقه الصح هو انك يبقا عندك initial function تقدر تطلع ال edges وبعدها تاخد ال edges وتحطهم مع بعض ... فالدكتور رايح ناحية اننا نستخدم ال NN في الحوار ده

Different Levels of Abstraction



Example from Honglak Lee (NIPS 2010)

20

دلوقت الدكتور بيتكلم علي اختيار الأكتفيشن فانكشنز .. راح يتكلم علي السجمويد فبيقول ان السلوب بتاع السجمويد تقريباً صفر طول منت بتبعد عن الصفر.. وده مشكله عشان انت هتاخد ال gradients .. فلو ال gradients كانت صغيره .. فانت مش هيبقا عندك progress كبير .. الدكتور بيتكلم علي السلايد الجايه و بيوضح يعني ان ال tanh فعلا بتبقا احسن من ال sigmoid .. الدكتور بيتكلم علي الكيرف الأسود اللي فوق خالص .. بيقولك ان ديه مشكلة ال vanishing gradients .... اعتقد هو مسميها sigmoid level 5 ... يمكن المقصود هنا هو sigmoid(sigmoid(sigmoid(sigmoid(sigmoid())))) ...

Understanding the difficulty of training deep feedforward neural networks

AI Stats 2010

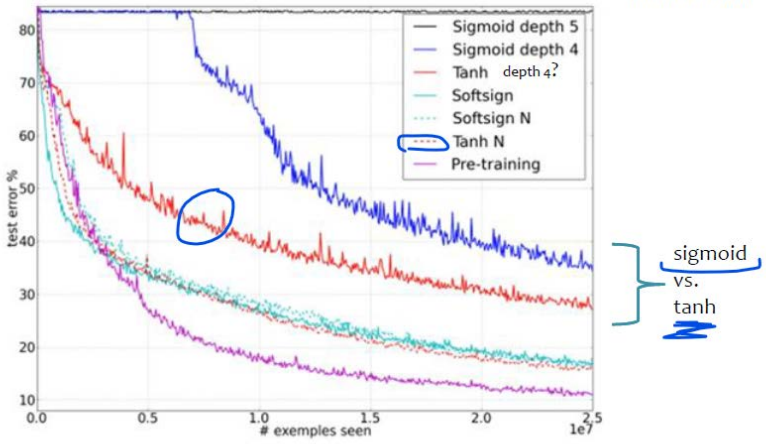
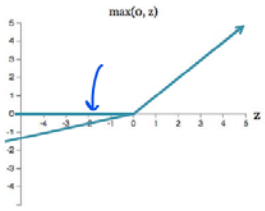


Figure from Glorot & Bentio (2010)

Activation Functions

- A new change: modifying the nonlinearity
  - reLU often used in vision tasks

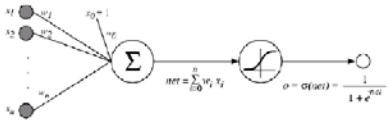


$$\max(0, w \cdot x + b).$$

Alternate 2: rectified linear unit

Linear with a cutoff at zero

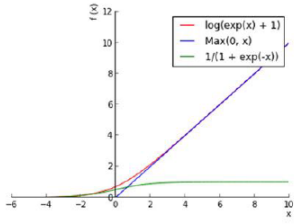
(Implementation: clip the gradient when you pass zero)



Slide from William Cohen

Activation Functions

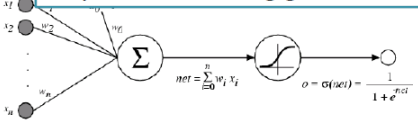
- A new change: modifying the nonlinearity
  - reLU often used in vision tasks



Alternate 2: rectified linear unit

Soft version: log(exp(x)+1)

Doesn't saturate (at one end)
Sparsifies outputs
Helps with vanishing gradient



Slide from William Cohen

Objective Functions for NNs

1. Quadratic Loss:
   - the same objective as Linear Regression
   - i.e. mean squared error
2. Cross-Entropy:
   - the same objective as Logistic Regression
   - i.e. negative log likelihood
   - This requires probabilities, so we add an additional "softmax" layer at the end of our network

	Forward	Backward
Quadratic	$J = \frac{1}{2}(y - y^*)^2$	$\frac{dJ}{dy} = y - y^*$
Cross Entropy	$J = y^* \log(y) + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = y^* \frac{1}{y} + (1 - y^*) \frac{1}{y - 1}$

الأوبجكتيف فانكشنز بتاعت ال NNs .. ليهم اسامي مختلفه شوية هنا في ال NN ... الهدف من السلايد ديه هو ال terminology يعني .. حاجه زي Quadratic loss .. اللي هي بتبقا (true – predicted) * 0.5 .. ال y star في السلايد هي ال predicted .... ديه هي ال Quadratic loss الي استخدمناها في ال mean squared error في ال linear regression .. عندك cross entropy هو نفس ال objective اللي بنستخدمو عشان ال binary logistic regression و ال multinomial .. هو بس بيبقا ال NLL بتاع الداتا .. حاجه من اللي تلاحظها في ال literature هو ان الناس بتستخدم ال NLL اللي هو ال cross entropy بشكل كتير في ال probabilistic models .. بس انت مش لازم تعمل كدا يعني عادي .. بس في اكتر الأوقات بتلاقي ال cross-entropy بيبقا احسن ...

Objective Functions for NNs

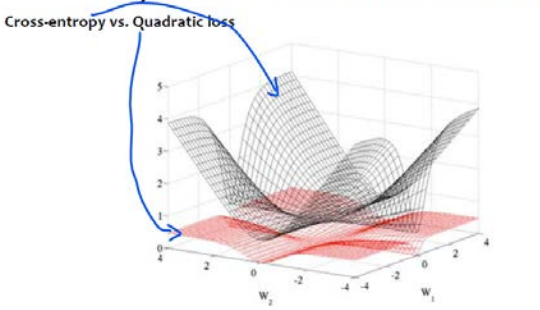
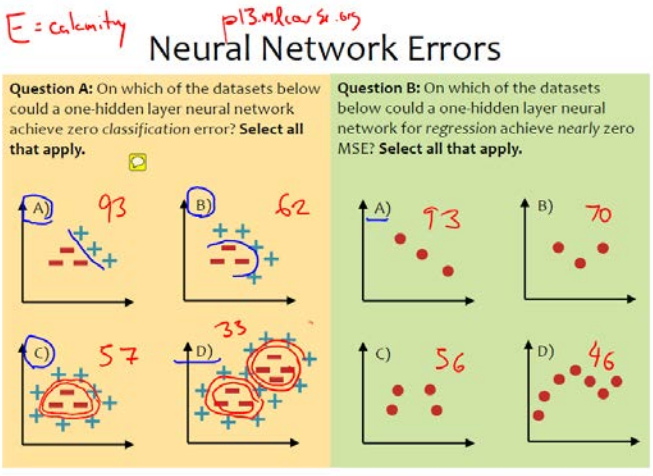
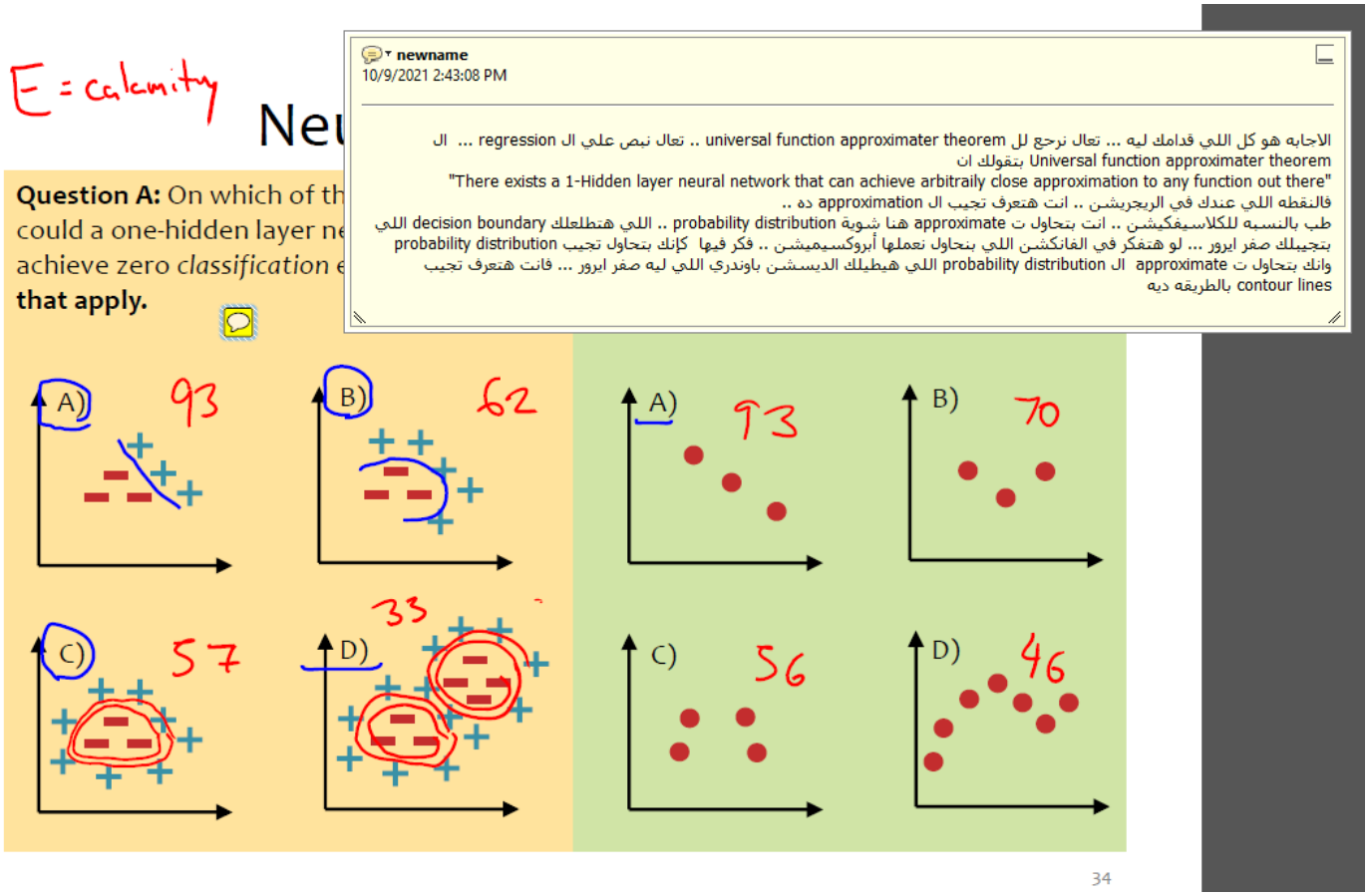


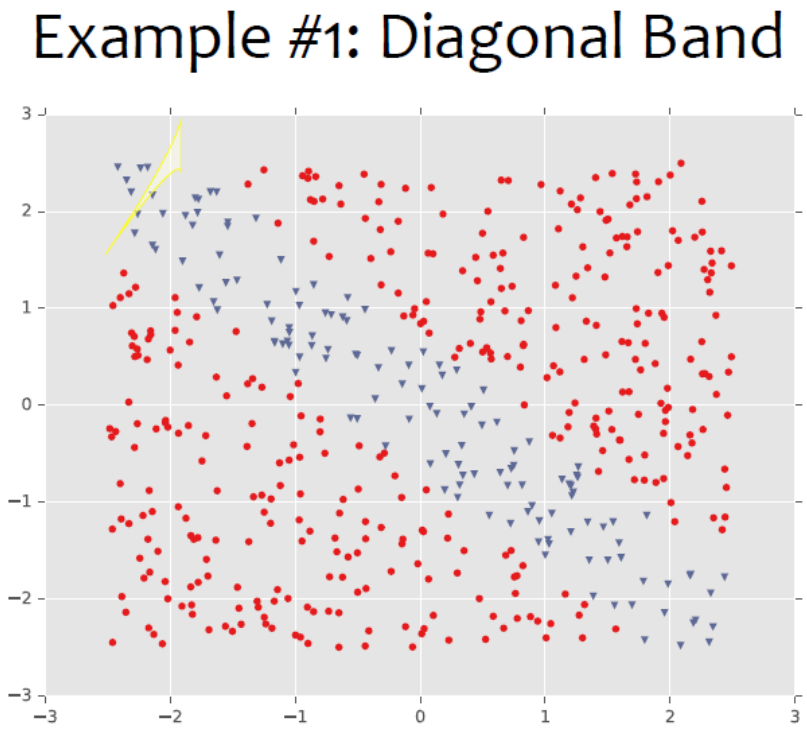
Figure 5: *Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers, W_1 respectively on the first layer and W_2 on the second, output layer.*

Figure from Glorot & Bentio (2010)



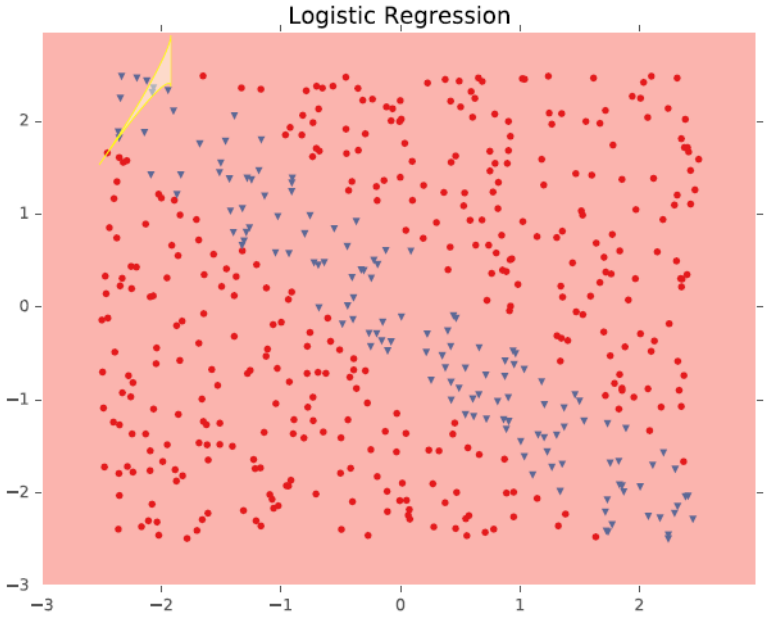


طيب هنتكلم علي ايه اللي بيحصل علي ارض الواقع باستخدام ال neural network .. الدكتور خد اول مثال فبيقول ام عملناه باللوجستيك ريجريشن ..

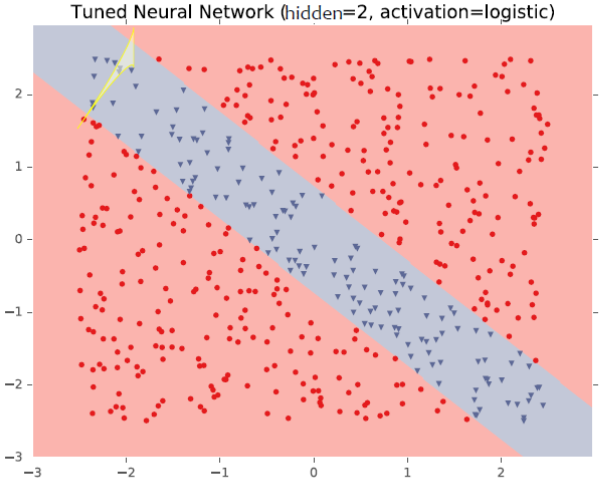


طلع كلو احمر .. زي الصوره اللي تحت ديه ..

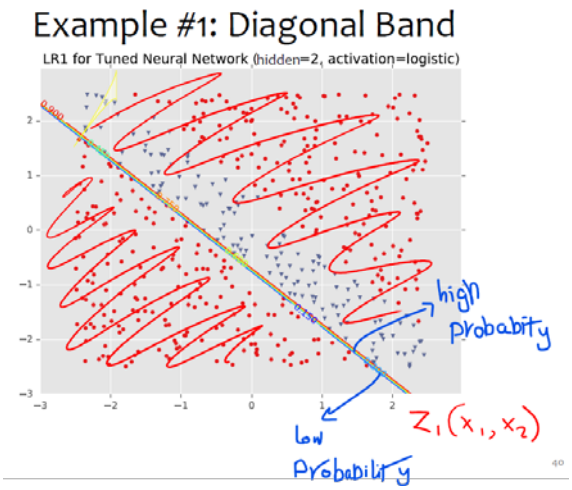
Example #1: Diagonal Band

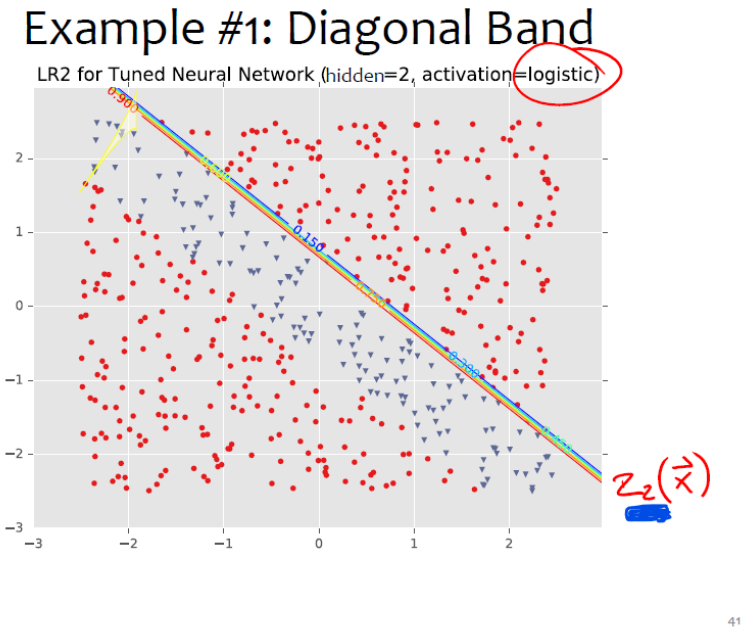


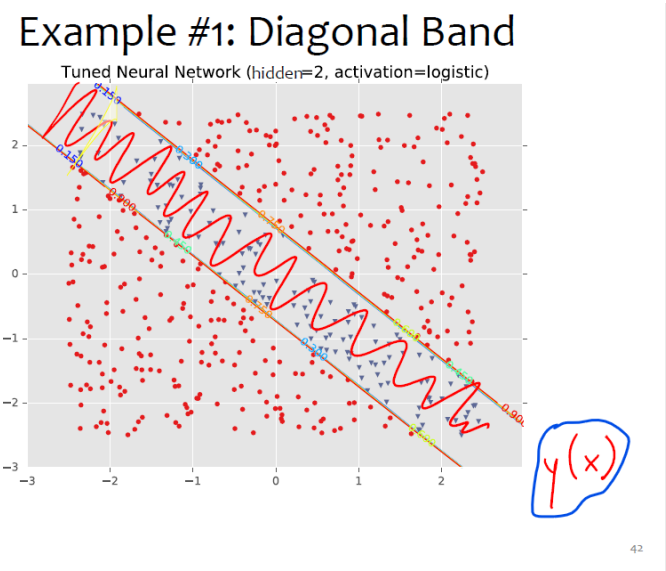
Example #1: Diagonal Band



لما جينا عملناها بالنويرال نتورك .. الدنيا اشتغلت عادي ...



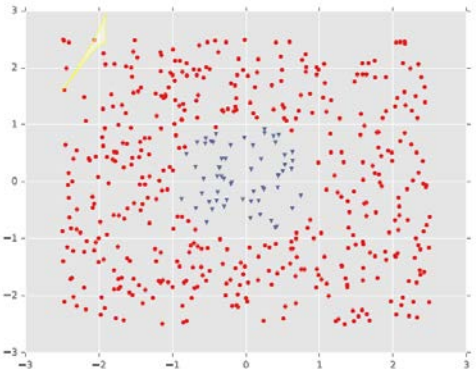




في المثال اللي فوق ده في 3 رسومات عشان الدكتور حابب يوضح ان أول صوره هي أول layer و تاني صوره هي تاني layer و آخر صوره هو الإتنين مع بعض ....

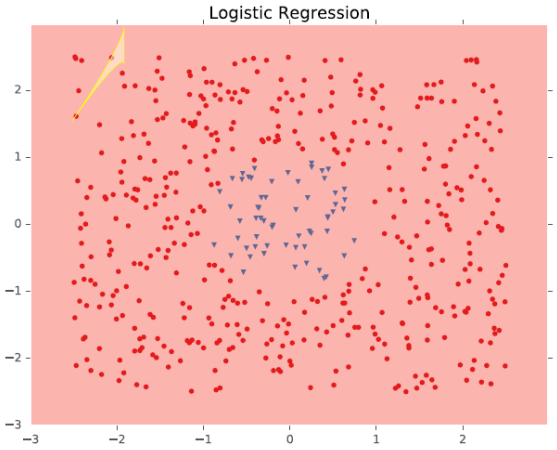
في مثال تاني ... اللوجستيك ريجريشن فشل فيه ...

Example #2: One Pocket



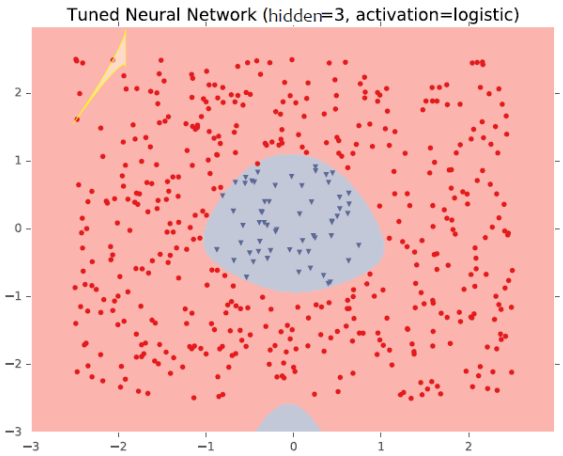
44

Example #2: One Pocket



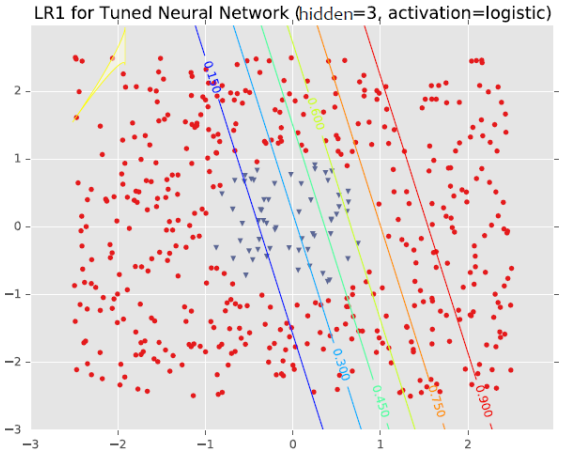
45

Example #2: One Pocket

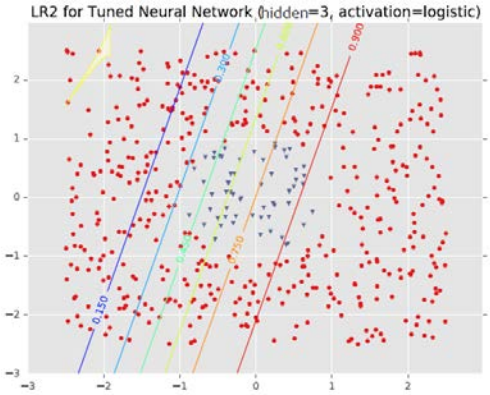


46

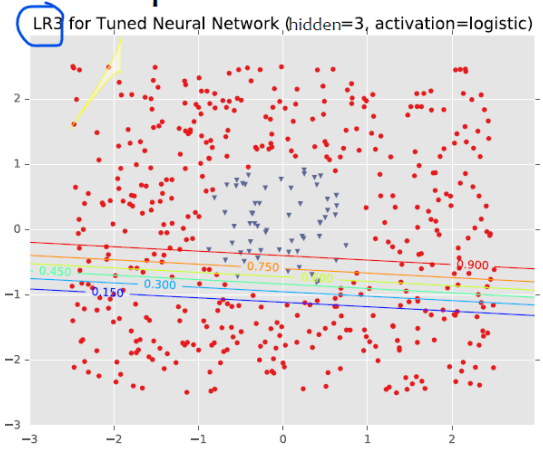
Example #2: One Pocket



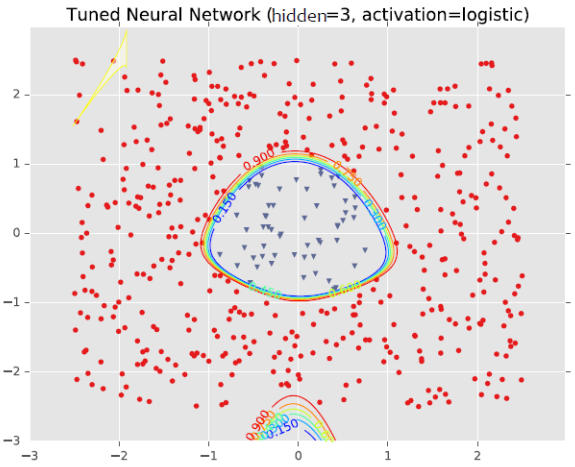
Example #2: One Pocket



Example #2: One Pocket

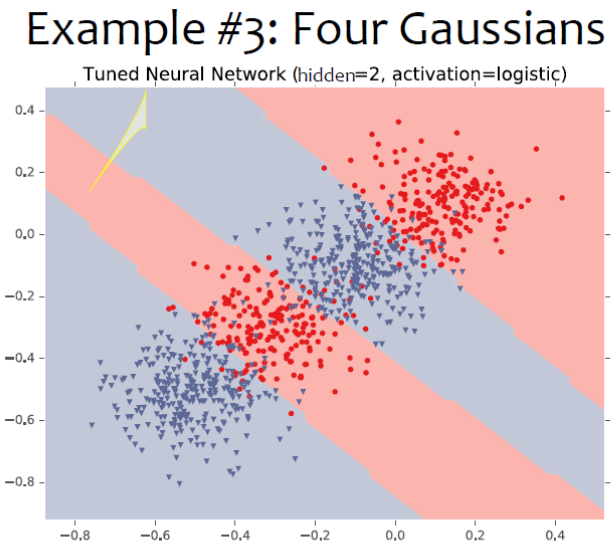


Example #2: One Pocket

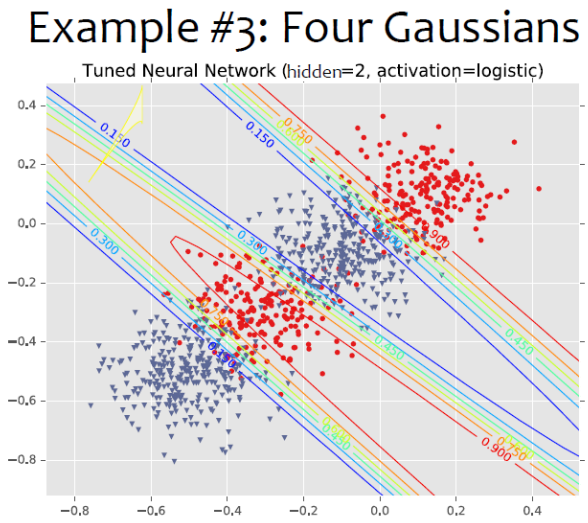


تعال نشوف مثال تالت .. الدكتور بيقول هنا الهدف من المثال ده انو يشوف ال neural network وهي بت .. fail ..

Example #3: Four Gaussians



55



58

المهم الدكتور بيتكلم علي المثال الرابع .. هو استخدم 3 hiddel units ف مره و ف مره تانيه استخدم 4 hidden units .. لقي ان ال 4 مبقاش احسن من ال 3 ... ده عشان ال objective function كان non convex و الطريقه اللي الدكتور استخدمها عشان يعمل ال training هنا كان انو عمل random initialization .. فبالتالي ال local minima في حالة ال 3 غير حالة 4 .....

طيب هنتكلم دلوقت علي ال ازاي هنحسب ال gradients .. الدكتور حط سؤالين .. اول واحد امتي هنقدر نحسب ال gradient بتاع أي نويرال نتورك و تاني سؤال هو امتي هنقدر نخلي ال gradient computation efficient .. طلع في طرق كتيره .. الدكتور بيقولك عاوزنا نفكر في طرق more general هو ان عندك فانكشن اسمها f بتاخد فيكتور طوله A وبترجع فيكتور طوله B .. فلما يبقا عندك فانكشن زي ديه .. انت بتحتاج تحسب A*B different partial derivatives .... اللي هو partial derivative of each output with respect to each input .. طيب فال setting بتاع الترينينج للنويرال نتورك .. مختلف شويه عن الماشين ليرننج .. عشان اللي عندنا هو بارمترز كتيره عشان ال A هتبقا كبيره .. بس B هتبقا بتساوي 1 ... الدكتور سأل سؤال ... ايه هي ال 1 value that is the output that we are trying to optimize .. هتبقا ال loss function اللي هو ال output of Loss(y hat, y) .. او اللي هو تقدر تقول انو ال J اللي هو هيبقا الفانكشن اللي بتاخد الأفريدج بتاع كل ال Ls .. في ال setting بتاعنا B = 1 ..

Training

Approaches to Differentiation

Given $f : \mathbb{R}^A \rightarrow \mathbb{R}^B, f(\mathbf{x})$

Compute $\frac{\partial f(\mathbf{x})_i}{\partial x_j} \forall i, j$

1. Finite Difference Method
   - Pro: Great for testing implementations of backpropagation
   - Con: Slow for high dimensional inputs / outputs
   - Required: Ability to call the function f(**x**) on any input **x**
2. Symbolic Differentiation
   - Note: The method you learned in high-school
   - Note: Used by Mathematica / Wolfram Alpha / Maple
   - Pro: Yields easily interpretable derivatives
   - Con: Leads to exponential computation time if not carefully implemented
   - Required: Mathematical expression that defines f(**x**)
3. Automatic Differentiation - Reverse Mode
   - Note: Called *Backpropagation* when applied to Neural Nets
   - Pro: Computes partial derivatives of one output $f(\mathbf{x})_i$ with respect to all inputs x_j in time proportional to computation of f(**x**)
   - Con: Slow for high dimensional outputs (e.g. vector-valued functions)
   - Required: Algorithm for computing f(**x**)

large B

4. Automatic Differentiation - Forward Mode
   - Note: Easy to implement. Uses dual numbers.
   - Pro: Computes partial derivatives of all outputs $f(\mathbf{x})_i$ with respect to one input x_j in time proportional to computation of f(**x**)
   - Con: Slow for high dimensional inputs (e.g. vector-valued **x**)
   - Required: Algorithm for computing f(**x**)

large A

Backprop

Wednesday, February 26, 2020 11:58 AM

not a NN diagram

Chain Rule

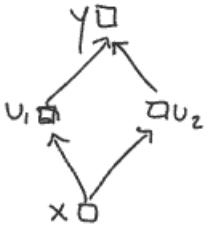
Def #1: $y = f(u)$, $u = g(x)$

Computation Graphs

Defs

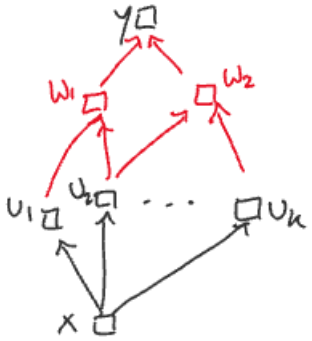
$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

Def #2: $y = f(u_1, u_2)$, $u_2 = g_2(x)$, $u_1 = g_1(x)$



$$\frac{dy}{dx} = \frac{dy}{du_1}\frac{du_1}{dx} + \frac{dy}{du_2}\frac{du_2}{dx}$$

Def #3: $y = f(\vec{u})$, $\vec{u} = g(x)$



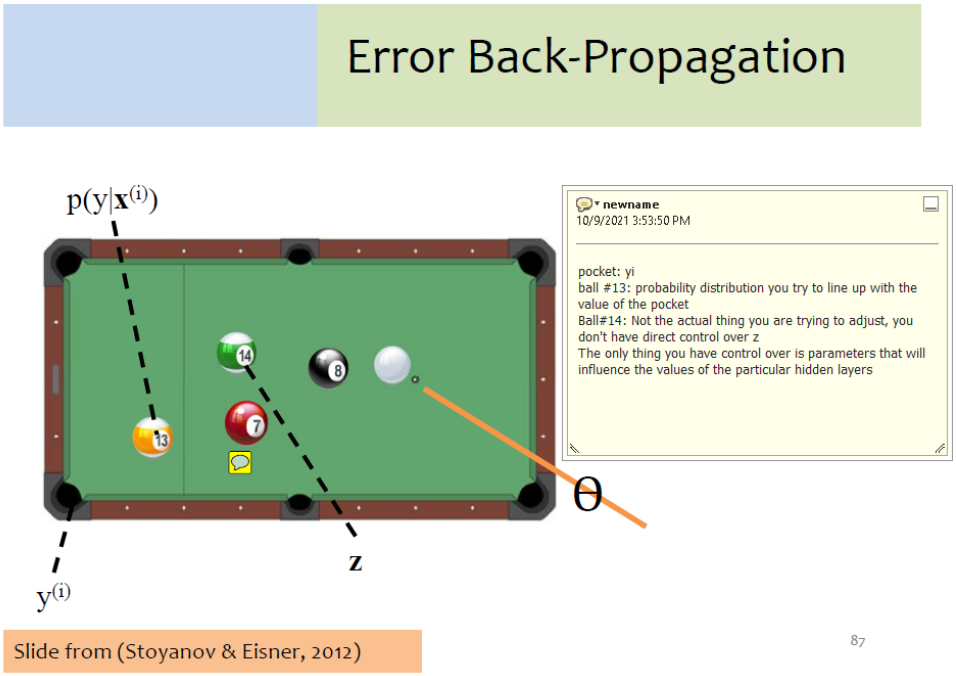
$$\frac{dy}{dx} = \sum_{k=1}^{K} \frac{dy}{du_k}\frac{du_k}{dx}$$

★ Holds for any intermediate quantities $\vec{u}$

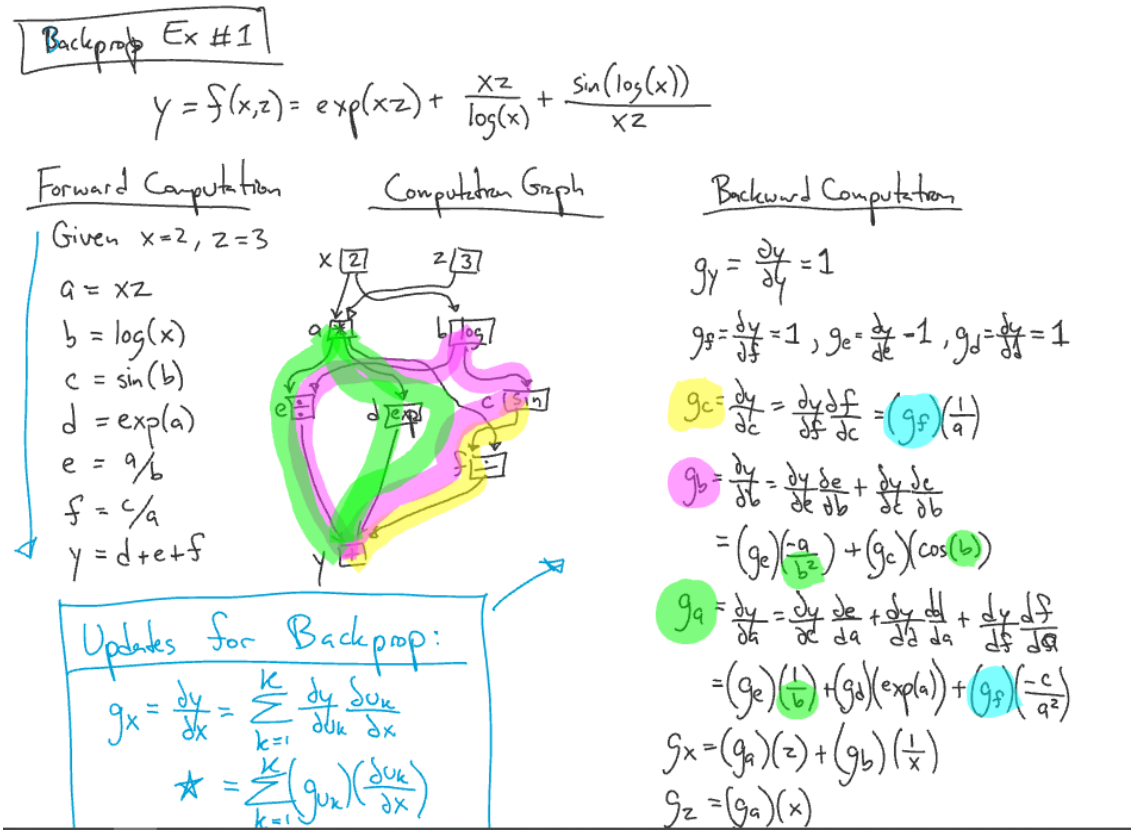
في التعريف رقم 3 .. الدكتور أكّد علي حتة انك حتي لما هتضيف ال w1 and w2 في النص .. هي مش هتفرق فحاجه لإنك في الآخر بتقول y = f(u) .. فهو انت متعرفش ال f اصلا بتبقا عامله ازاي ...

الدكتور رجع علي السلايدز وقال ان الباك بروبجيشن ما هو إلا repeated application of the chain rule فالناس اللي بتفضل تصيح .. لو سمحت كفايه صياح .. هو بس بيبقا شكلو رخم شويه لما بيبقا عندك complicated computation graph .. المهم خد بالك برضو ان ال computation graph غير ال Neural Network ..

تعال ناخد شوية intuition عن اللي بيحصل في الباك بروبجيشن ..



بص علي مثال البلياردو اللي في المحاضره ... الدكتور دلوقت بيتكلم علي ازاي ان الباك بروب بيعمل نفس الحاجه عشان يآبديت البارمترز بتاعت النويرال نتورك ...



Neural Network Training

- Consider a 2-hidden layer NN
- params are $\theta = [\alpha^{(1)}, \alpha^{(2)}, \beta]$
- SGD Training

  Iterate until convergence:

  (1) Sample $i \in \{1, \ldots, N\}$

  (2) Compute gradient by backprop

$$g_{\alpha^{(1)}} = \nabla_{\alpha^{(1)}} J^{(i)}(\theta)$$

$$g_{\alpha^{(2)}} = \nabla_{\alpha^{(2)}} J^{(i)}(\theta)$$

$$g_{\beta} = \nabla_{\beta} J^{(i)}(\theta)$$

  (3) Step opposite the gradient

$$\alpha^{(1)} \leftarrow \alpha^{(1)} - \gamma g_{\alpha^{(1)}}$$

$$\alpha^{(2)} \leftarrow \alpha^{(2)} - \gamma g_{\alpha^{(2)}}$$

$$\beta \leftarrow \beta - \gamma g_{\beta}$$

$\theta \leftarrow \theta - \gamma g_{\theta}$

$J^{(i)}(\theta) = \ell(h_\theta(\vec{x}^{(i)}), y^{(i)})$

Recall:

$$\nabla J(\vec{a}, \vec{b}) = \nabla_{\vec{a},\vec{b}} J(\vec{a}, \vec{b})$$

$$\nabla_{\vec{a}} J(\vec{a}, \vec{b}) = \begin{bmatrix} \partial J/\partial a_1 \\ \partial J/\partial a_2 \\ \vdots \\ \partial J/\partial a_K \end{bmatrix}$$

left out terms

$\beta \leftarrow \beta - \gamma g_\beta$

Backprop Ex #2: for NN

left out intercept terms

Given: ① Dec. fn. $\hat{y} = h_\theta(\vec{x}) = \sigma\left((\alpha^{(3)})^T \sigma\left((\alpha^{(2)})^T \sigma\left((\alpha^{(1)})^T \vec{x}\right)\right)\right)$

β $\vec{z}^{(2)}$ $\vec{z}^{(1)}$

② loss fn. $J = \ell(\hat{y}, y^*) = y^* \log(\hat{y}) + (1-y^*)\log(1-\hat{y})$ ← cross entropy

① Forward

Given $\vec{x}, \alpha^{(1)}, \alpha^{(2)}, \alpha^{(3)}, y^*$

$\vec{z}^{(0)} = \vec{x}$

for $i = 1, 2, 3$:

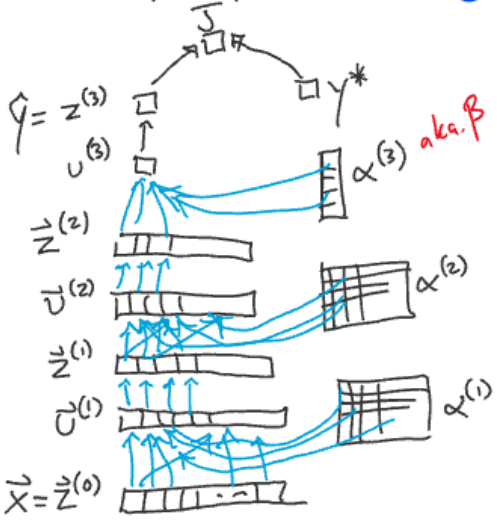
$\vec{u}^{(i)} = (\alpha^{(i)})^T \vec{z}^{(i-1)}$

$\vec{z}^{(i)} = \sigma(\vec{u}^{(i)})$

$\hat{y} = z^{(3)}$

$J = \ell(\hat{y}, y^*)$

② Comp Graph



③ Backward

$g_J = [1]$

$g_{\hat{y}} = \frac{y^*}{\hat{y}} - \frac{(1-y^*)}{(1-\hat{y})}$

HW5

for $i = 3, 2, 1$:

$g_{\vec{u}^{(i)}} = \ldots$

$g_{z^{(i-1)}} = \ldots$

$g_{\alpha^{(i)}} = \ldots$

$g_x = g_{z^{(0)}}$