VANISHING AND EXPLODING GRADIENTS GRADIENT CHECK

Deep Neural Networks

Session 11

Pramod Sharma
pramod.sharma@prasami.com

2

Neural Networks face problems of vanishing or exploding gradients

5/21/2024

pra-sâmi

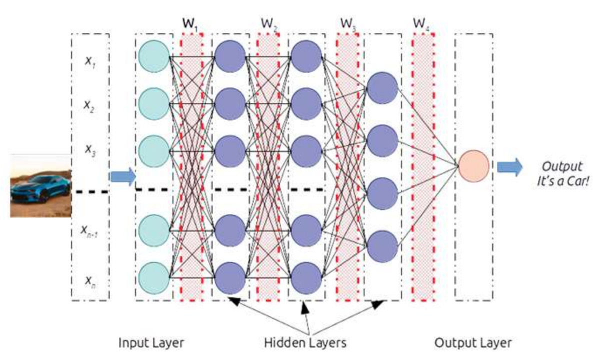
3

Deeper the network more are the chances of the gradients becoming smaller and smaller or keep growing…

4

Weight multiplied

- We know that :
  - $z = X * W + b$
  - $\hat{y} = a = \sigma(z)$
  - $a_1 = \sigma(a_0 . W_1)$



- That in multilayer network
  - $\hat{y} = \sigma(\sigma(\sigma(\sigma(\sigma(a_0 . W_1). W_2). W_3). W_4) \ldots .$
  - For explanation purpose assume σ (z) = z (say ReLU)
  - $y = W_1 . W_2 . W_3 . W_4$
  - so any change in y will result in $W_1 * W_2 * W_3 * W_4$ times y in layer 1
  - Longer the chain, more W_s will be multiplied.

5

More Layers… More problems

- Assume we have 150 layers
- Also assume our weight is say 1.1
  - => 1.1^{150} = 1.6 million
- On the other hand assume our weight is 0.9
  - => 0.9^{150} = 1.4 e^{-7}

pra-sâmi

6

It's a Severe Problem…

- No silver bullet solution….
- There is multi-prong approach to it…
- First, Initialise your weights as close to 1 as possible (not 1)
- It is found that for tanh activation function
  - Divide by $\sqrt{number\ of\ nodes\ in\ the\ previous\ layer}$
  - for Gaussian distribution it normalises the data with var =1
- Some cases : $\frac{2}{\sqrt{number\ of\ nodes\ in\ the\ previous\ layer}}$
- In ReLU , : $\frac{2}{\sqrt{number\ of\ nodes\ in\ the\ previous\ layer + numbe\ \ of\ nodes\ in\ current\ layer}}$
- Some literature, even $\frac{K}{\sqrt{number\ of\ nodes\ in\ the\ previous\ layer}}$; κ is a another parameter to tune

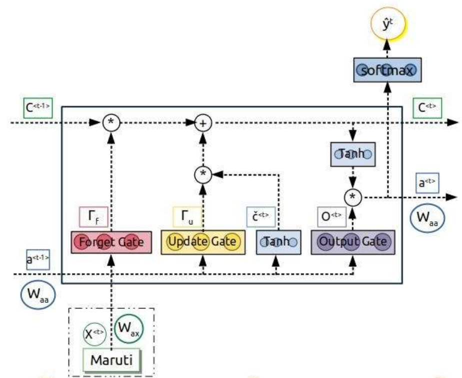
pra-sâmi

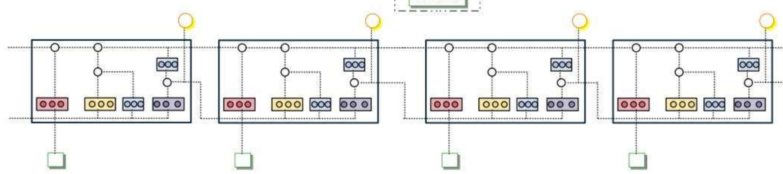
7

Change your architecture - LSTM

LSTM

$$\check{C}^{<t>} = \tanh(W_c.[a^{<t-1>}, X^{<t>}] + b_c)$$
$$\Gamma_u = \sigma(W_u.[a^{<t>}, X^{<t>}] + b_u)$$
$$\Gamma_f = \sigma(W_f.[C^{<t>}, X^{<t>}] + b_f)$$
$$\Gamma_o = \sigma(W_o.[C^{<t>}, X^{<t>}] + b_o)$$
$$C^{<t>} = \Gamma_u . \check{C}^{<t>} + \Gamma_f . C^{<t-1>}$$
$$a^{<t>} = \Gamma_o * \tanh C^{<t>}$$



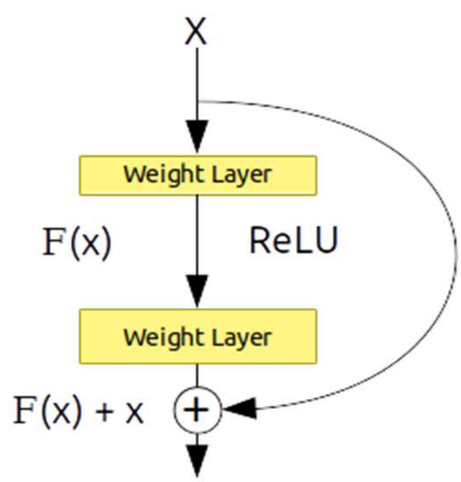


5/21/2024

pra-sâmi

8

As in ResNet



Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
Microsoft Research

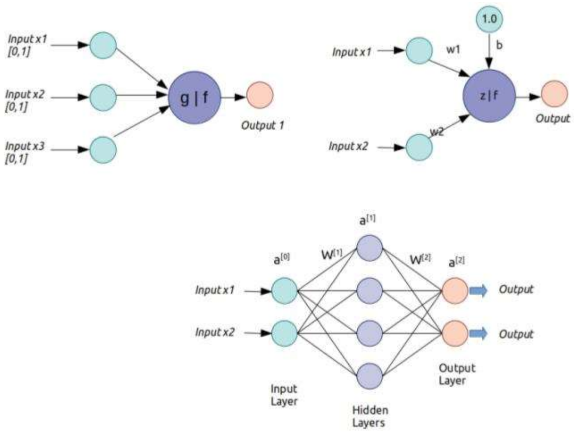
5/21/2024

pra-sâmi

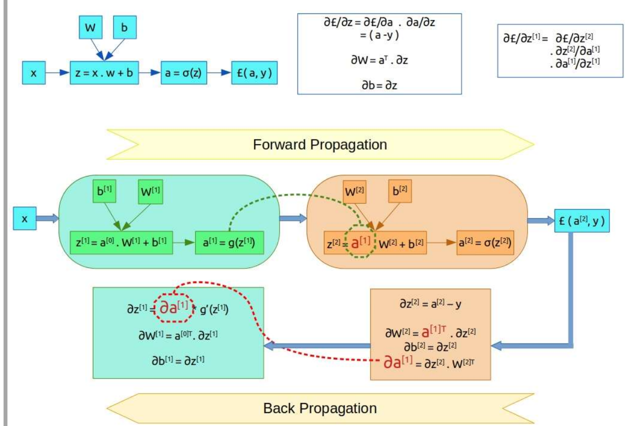
9

Gradient Checks

- Recap



- The math



10

Loss Function and its Derivative

- The loss for our prediction $\hat{y}$ with respect to the true labels y is given by:

  L($\hat{y}$, y) = − y . log $\hat{y}$− (1 − y) .log (1−$\hat{y}$)

- For all samples:

  J($\hat{y}$, y)= $-\frac{1}{m}\sum_{i\in m} y_i \log \hat{y}_i - (1-y_i).\log(1-\hat{y}_i)$

  Where $\hat{y}$ = σ (a. W + b)

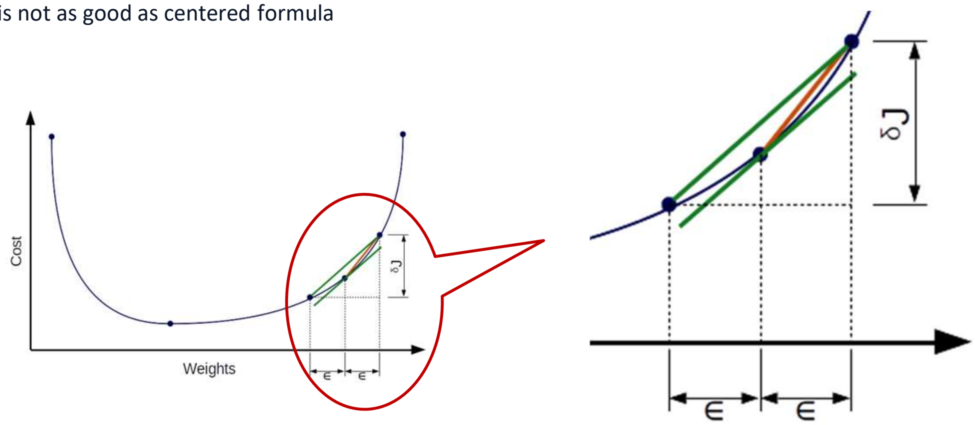
- Therefore, we can say that:

  J($\hat{y}$, y) = J (W, b) = J ($W_1, W_2, W_3, \ldots. W_n, b_1, b_2, b_3, \ldots. b_n$)

11

Calculation of derivative

- Use the centered formula
  - The formula you may have seen for the finite difference approximation when evaluating the numerical gradient is not as good as centered formula



12

Gradient Checking

- Also called “Grad Check”
- Do it to verify the model’s math(Debug) only
  - Too heavy for training, switch off once the model is verified.
- For all values of Ws and Bs, we can calculate:

$\delta\theta_{approx}$ = J(W_1, ..., $W_i + \varepsilon$, ... W_n, b_1, b_2, b_3, ... b_n) - J(W_1, ..., $W_i - \varepsilon$, ... W_n, b_1, b_2, b_3, ... b_n) / (2 * ε)

- To check if $\delta\theta_{approx}$ and $\delta\ \theta$ are close

$\frac{\|\delta\theta_{approx} - \delta\ \theta\|_2}{\|\delta\theta_{approx}\|_2 + \|\delta\ \theta\|_2}$ is very small

- For ε = 1e-7
  - Relative error > 1e-2 usually means the gradient is probably wrong
  - 1e-2 > relative error > 1e-4 should make you feel uncomfortable
  - 1e-4 > relative error is usually okay for objectives with kinks. But if there are no kinks (e.g. use of tanh nonlinearities and softmax), then 1e-4 is too high.
  - 1e-7 and less you should be happy

13

Grad Check Steps

- Recall: Our model has all weights and biases stored
  - Model ={ "W_1": … , "b_1": …, "W_2": … , "b_2": …, … … "W_n": … , "b_n": …}
  - We have implemented our forward prop and back prop
- Step 1 : Pick model and convert all weights and biases into a vector θ
- Step 2: Similarly pick δW and δb and convert to a vector $\delta\theta$
- Step 3: for each of the value in the vector θ
  - Make copy of θ and $\delta\theta$
  - Increase θ_i to $\theta_i + \varepsilon$
  - Calculate J + (Cost with increased θ)
  - Similarly calculate J- (Cost with decreased θ)
  - Use J+ and J- to calculate if $\delta\theta_{approx}$
  - Calculate $\delta\theta$ as usual
  - Find error

5/21/2024 pra-sâmi

14

Grad Check

- Phew… too lengthy calculations….
- Good News!
  - Both **Tensorflow** and **Torch** have **autograd** implementation for us. So in real implementation we will be using those functions for gradient checking
- Caution!
  - This check is resource hungry
  - Once the verification is done, comment/switch off the code
- Deeper the network → the higher the relative errors
  - For the input data for a 10-layer network, a relative error of 1e-2 might be okay because the errors build up on the way
  - Conversely, an error of 1e-2 for a single differentiable function likely indicates incorrect gradient

5/21/2024 pra-sâmi

15

Learning

- So far, we've discussed the static parts of a Neural Networks:
  - How we can set up the network connectivity,
  - The data
  - The loss function

- Time to look at the dynamics:
  - The process of learning the parameters
  - Finding good hyper-parameters

5/21/2024

pra-sâmi

16

Next Session…
The process of learning the parameters…
Finding good hyper-parameters…

5/21/2024

pra-sâmi

17

5/21/2024

pra-sâmi