Vanishing and Exploding Gradients
Gradient Check

Deep Neural Networks

Session 11

Pramod Sharma
pramod.sharma@prasami.com

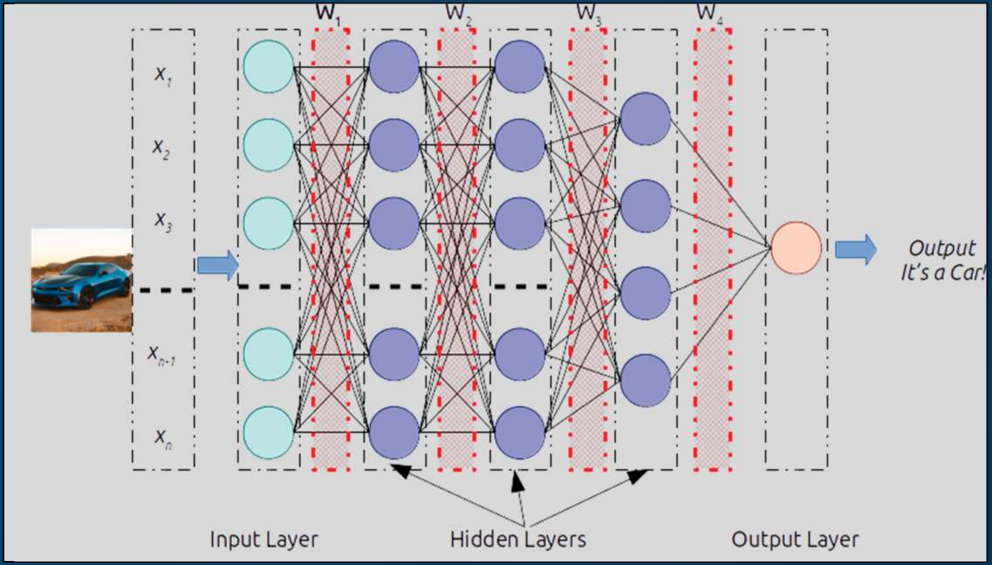
Neural Networks face problems of vanishing or exploding gradients

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

Weights multiplied…

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

- That in multilayer network
  - $\hat{y} = \sigma(\sigma(\sigma(\sigma(\sigma(a_0 . W_1). W_2). W_3). W_4) \dots .$
  - 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.



pra-sâmi

More Layers… More problems

- ❑ Assume we have 150 layers

- ❑ Also assume our weight is say 1.1

  => $1.1^{150} = 1.6$ million

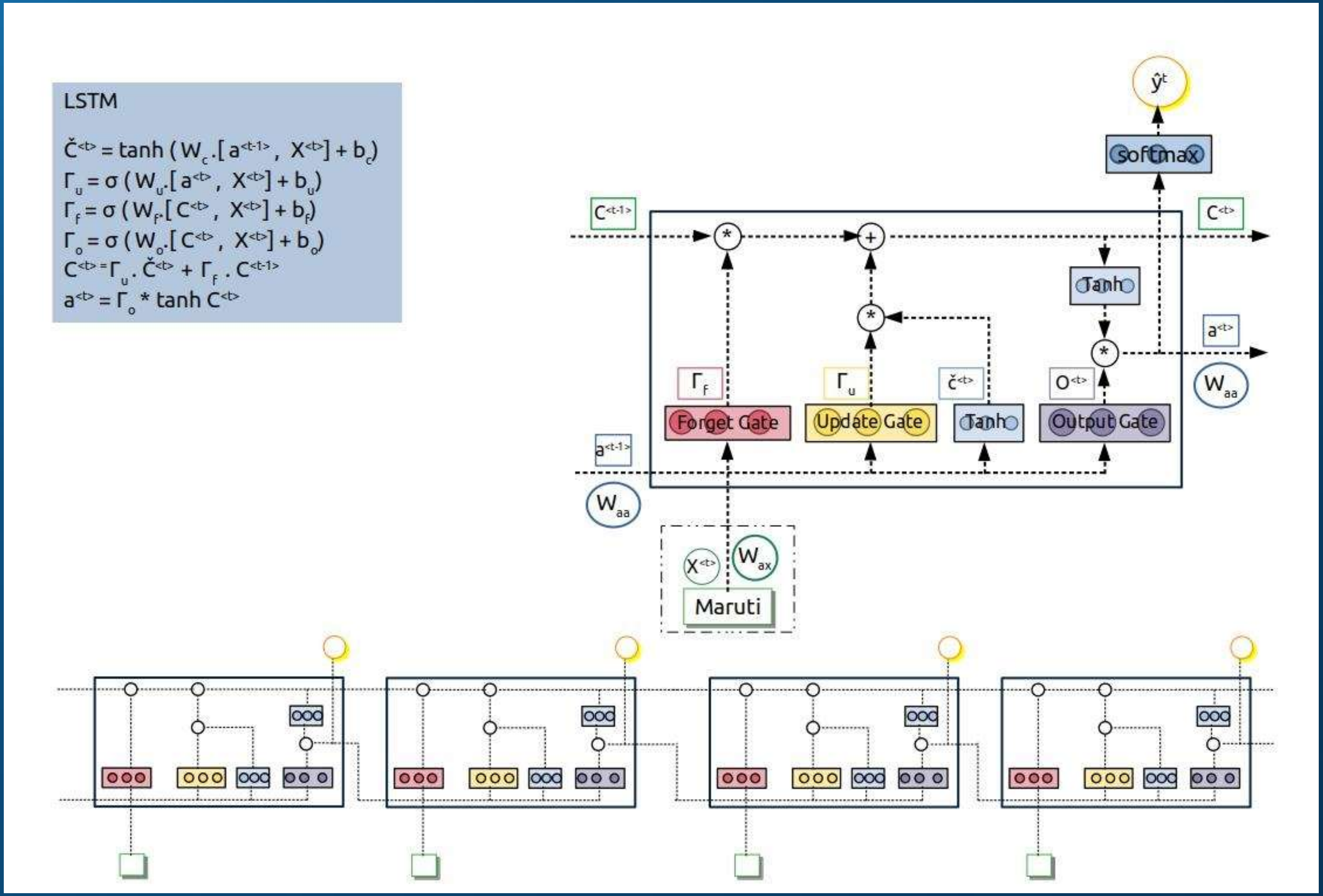
- ❑ On the other hand assume our weight is 0.9

  => $0.9^{150} = 1.4\ e^{-7}$

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 $\sqrt{number\ of\ nodes\ in\ the\ previous\ layer}$
- ❑ Some cases : $\frac{2}{\sqrt{number\ of\ nodes\ in\ the\ previous\ layer}}$
- ❑ In ReLU , : $\frac{2}{\sqrt{number\ of\ nodes\ in\ the\ previous\ layer+number\ of\ nodes\ in\ current\ layer}}$
- ❑ Some literature suggest, even $\frac{K}{\sqrt{number\ of\ nodes\ in\ the\ previous\ layer}}$; κ is a another parameter to tune

Change Your Architecture - LSTM



LSTM

$\check{C}^{<t>} = \tanh(W_c.[a^{<t-1>}, X^{<t>}] + b_c)$

$\Gamma_u = \sigma(W_u.[a^{<t>}, X^{<t>}] + b_u)$

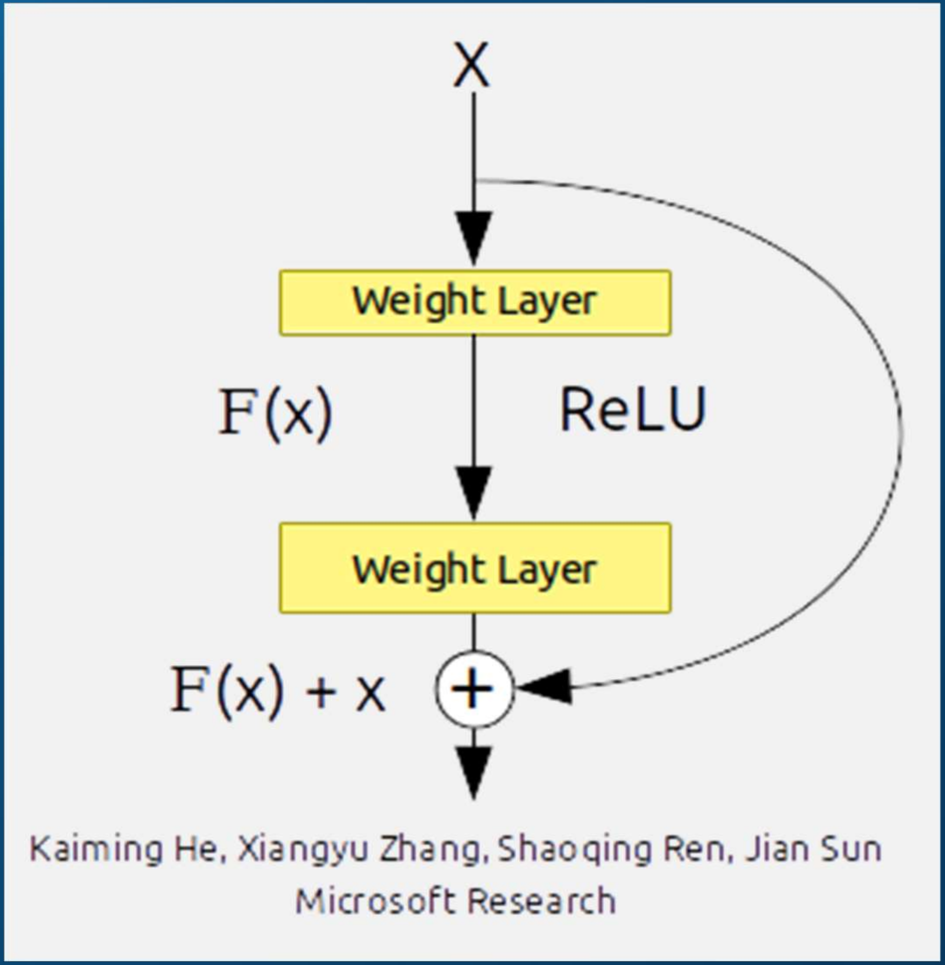
$\Gamma_f = \sigma(W_f.[C^{<t>}, X^{<t>}] + b_f)$

$\Gamma_o = \sigma(W_o.[C^{<t>}, X^{<t>}] + b_o)$

$C^{<t>} = \Gamma_u . \check{C}^{<t>} + \Gamma_f . C^{<t-1>}$

$a^{<t>} = \Gamma_o * \tanh C^{<t>}$

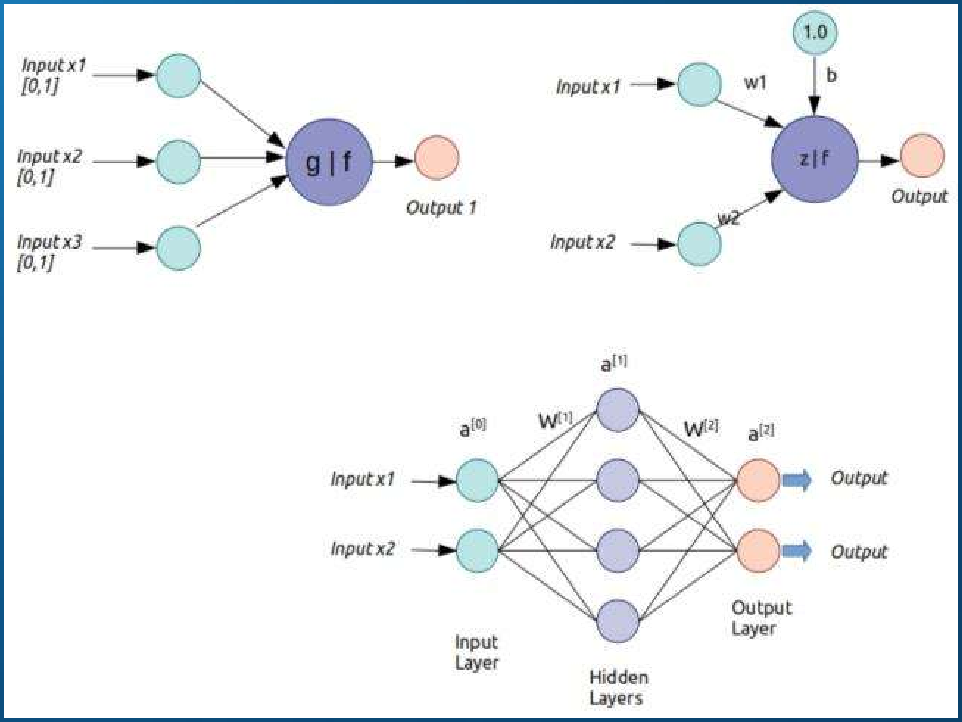
As in ResNet


X
Weight Layer
F(x)
ReLU
Weight Layer
F(x) + x
+
Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
Microsoft Research

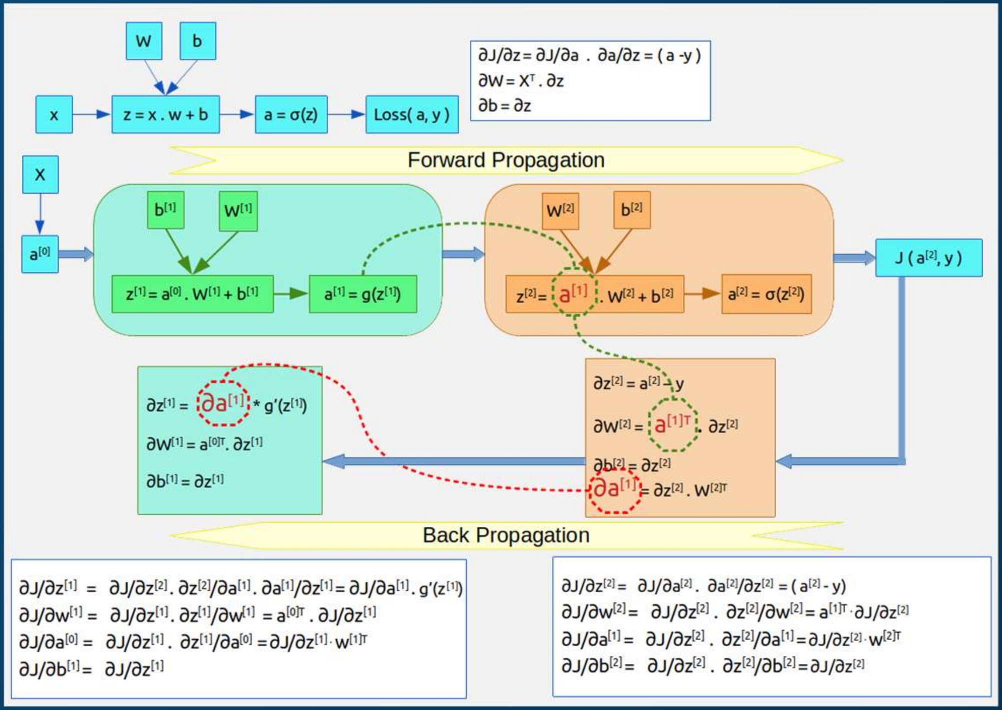
Gradient Check

Gradient Checks

- Recap



- The math


W
b
x
$z = x . w + b$
$a = \sigma(z)$
Loss(a, y)
$\partial J/\partial z = \partial J/\partial a \ . \ \partial a/\partial z = (a - y)$
$\partial W = X^T . \partial z$
$\partial b = \partial z$
Forward Propagation
X
$a^{[0]}$
$b^{[1]}$
$W^{[1]}$
$z^{[1]} = a^{[0]} . W^{[1]} + b^{[1]}$
$a^{[1]} = g(z^{[1]})$
$W^{[2]}$
$b^{[2]}$
$z^{[2]} = a^{[1]} . W^{[2]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$J (a^{[2]}, y)$
$\partial z^{[1]} = \partial a^{[1]} * g'(z^{[1]})$
$\partial W^{[1]} = a^{[0]T} . \partial z^{[1]}$
$\partial b^{[1]} = \partial z^{[1]}$
$\partial z^{[2]} = a^{[2]} - y$
$\partial W^{[2]} = a^{[1]T} . \partial z^{[2]}$
$\partial b^{[2]} = \partial z^{[2]}$
$\partial a^{[1]} = \partial z^{[2]} . W^{[2]T}$
Back Propagation
$\partial J/\partial z^{[1]} = \partial J/\partial z^{[2]} . \partial z^{[2]}/\partial a^{[1]} . \partial a^{[1]}/\partial z^{[1]} = \partial J/\partial a^{[1]} . g'(z^{[1]})$
$\partial J/\partial w^{[1]} = \partial J/\partial z^{[1]} . \partial z^{[1]}/\partial w^{[1]} = a^{[0]T} . \partial J/\partial z^{[1]}$
$\partial J/\partial a^{[0]} = \partial J/\partial z^{[1]} . \partial z^{[1]}/\partial a^{[0]} = \partial J/\partial z^{[1]} \cdot w^{[1]T}$
$\partial J/\partial b^{[1]} = \partial J/\partial z^{[1]}$
$\partial J/\partial z^{[2]} = \partial J/\partial a^{[2]} . \partial a^{[2]}/\partial z^{[2]} = (a^{[2]} - y)$
$\partial J/\partial w^{[2]} = \partial J/\partial z^{[2]} . \partial z^{[2]}/\partial w^{[2]} = a^{[1]T} \cdot \partial J/\partial z^{[2]}$
$\partial J/\partial a^{[1]} = \partial J/\partial z^{[2]} . \partial z^{[2]}/\partial a^{[1]} = \partial J/\partial z^{[2]} \cdot w^{[2]T}$
$\partial J/\partial b^{[2]} = \partial J/\partial z^{[2]} . \partial z^{[2]}/\partial b^{[2]} = \partial J/\partial z^{[2]}$

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:

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

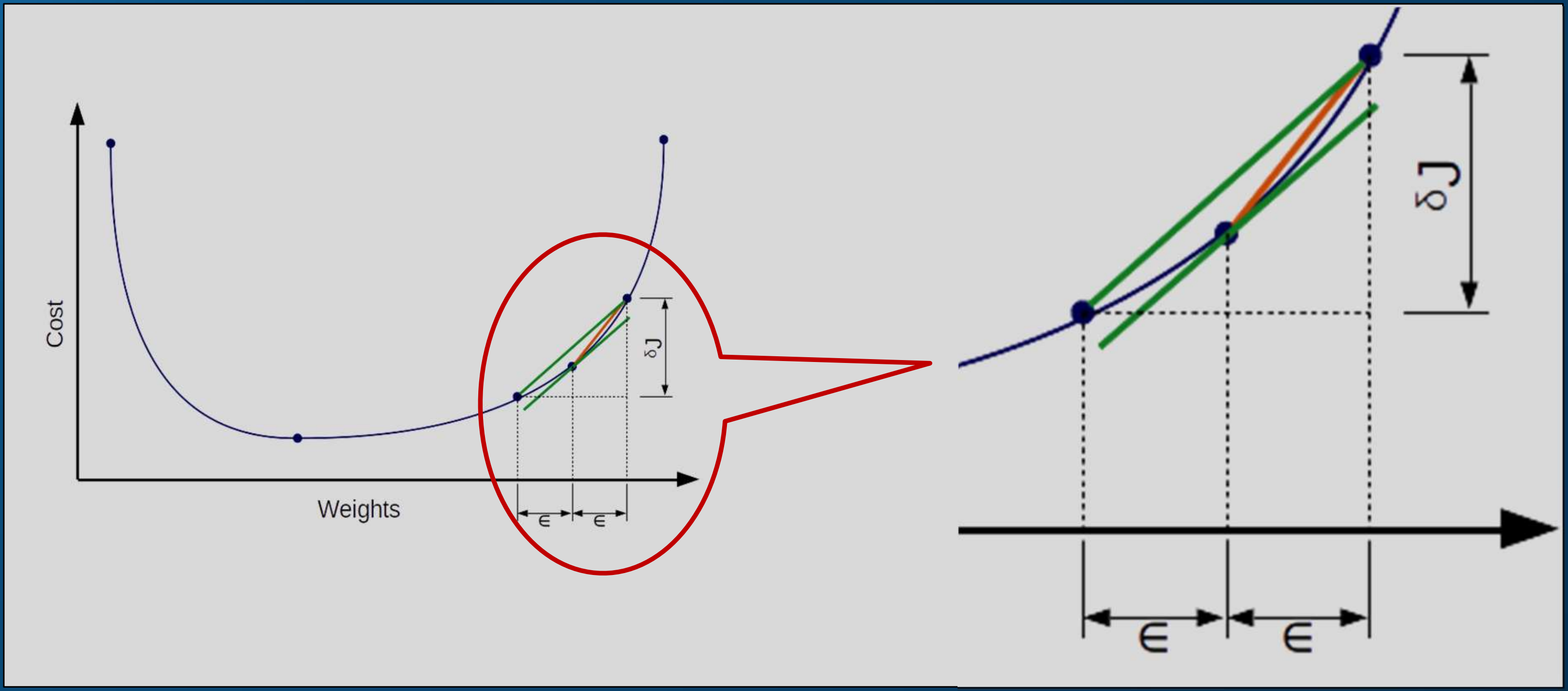
  Where $\hat{y} = \sigma (a . W + b)$

- Therefore, we can say that:

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

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



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

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 $\delta\theta_{approx}$
  - Calculate $\delta\theta$ as usual
  - Find error

Remember to Turn off Dropout/Augmentations

- When performing gradient check, remember to turn off any non-deterministic effects in the network, such as dropout, random data augmentations, etc.

- Otherwise these can clearly introduce huge errors when estimating the numerical gradient

- Downside ➔ you wouldn't be gradient checking them
  - It might be that dropout isn't back-propagated correctly)

Use Double Precision

- A common pitfall is using single precision floating point to compute gradient check. It is often that case that you might get high relative errors (as high as 1e-2) even with a correct gradient implementation

- There are cases when relative errors plummet from 1e-2 to 1e-8 by switching to double precision

Stick Around Active Range of Floating Point

- What Every Computer Scientist Should Know About Floating-Point Arithmetic https://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html
- In neural nets it is common to normalize the loss function over the batch
- If gradients per datapoint are very small, then *additionally* dividing them by the number of data points starts to give very small numbers
- Always print the raw numerical/analytic gradient,
  - Make sure that the numbers you are comparing are not extremely small (e.g. roughly 1e-10 and smaller in absolute value
  - If so, you may want to temporarily scale your loss function up by a constant to bring them to a “nicer” range where floats are more dense - ideally on the order of 1.0, where your float exponent is 0.

Kinks in the Objective Function

- One source of inaccuracy to be aware of during gradient checking is the problem of *kinks*
- Kinks refer to non-differentiable parts of an objective function
  - Such as ReLU max(0,x), or the SVM loss, Maxout neurons, etc.
- Consider gradient checking the ReLU function at x=−1e-6
- Since x<0, the analytic gradient at this point is exactly zero.
- However, the numerical gradient would suddenly compute a non-zero gradient because f(x + h) might cross over the kink (e.g. if h>1e−6) and introduce a non-zero contribution.
- You might think that this is a pathological case, but in fact this case can be very common.
- Keeping track of the identities of all "winners" in a function of form max(x, y); That is, was x or y higher during the forward pass. If the identity of at least one winner changes when evaluating f(x + h) and then f(x − h), then a kink was crossed and the numerical gradient will not be exact

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

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

pra-sâmi

THANK YOU