CS231n: CNNs for Visual Recognition **Spring 2017**

Lecture 4

Course Coordinator: Prof. Fei-Fei Li *Scribes: Akash Gupta*

- Computational graphs

- Graphical representation for a function where nodes of the graph are steps in between for processing.
- Advantage: Allows us to use backpropagation which involves recursively using chain-rule in order to compute the gradients.
- Each node has local inputs, local gradients, output and Gradients.

Q) *Why use a computational graph to calculate a gradient when we can just calculate the derivative?*
A) Calculating the derivative of the expression becomes really difficult for complex expressions. Computational graphs solve this by representing these expressions in simpler form and then just multipying the results using the chain rule:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial q} . \frac{\partial q}{\partial x}$$

So, at a node we just want to calculate the local gradients and then during backprop we take numerical value coming from the upstream and multiply with the local gradients and sent these back to the connected nodes without caring about anything else but immediate surroundings.

Patterns in backward flow
- add gate: gradient distributor
- max gate: gradient router
- mul gate: gradient switcher

Gradients add at branches:

$$\frac{\partial L}{\partial x} = \sum_i \frac{\partial L}{\partial q_i} . \frac{\partial q_i}{\partial x}$$

Gradients for vectorized code: The difference is that each variable is now a vector so instead of derivatives there are Jacobian matrics containing derivatives of elements of one vector w.r.t. elements of the vector.

**NOTE: The gradient with respect to the variable should have the same shape as the variable.

API:

- $forward()$ - compute result of an operation and save any intermediates needed for gradient computation in memory.
- $backward()$ - apply chain rule to compute the gradient of the loss function with respect to the inputs.

** DL libraries for e.g. Caffe has layers which are basically computational nodes each having a forward pass and backward pass.

- Neural Networks

- They are basically classic functions where we have simpler functions that are hierarchically stacked on top of each other to form a more complex non-linear function.
- So this is the idea of having multiple stages of hierarchical computation.
- Generally done by stacking linear layers on top each other with non-linear functions in between.
- W1(first layer) - templates, W2(second layer) - weighted sum of templates, x(inputs) - image, h(hidden units) - score of how much of each template of W1 is present in x.
- Remember car color example for intuition.
- Neural networks are not really neural.

<u>Activation functions</u>: Sigmoid, leaky RELU, tanh, Maxout, RELU, ELU,...