Neural Networks and Deep Learning: Implementations Issues

Nicolas Thome

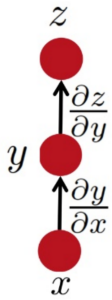
Conservatoire National des Arts et Métiers (Cnam)
Département Informatique

Challenges for Training Deep Learning Models

- Training of deep ConvNets: Gradient descent on loss function $\mathcal{L}$:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla \mathcal{L}\left(\mathbf{w}^t\right)$$

- Error-Backpropagation: way to compute $\nabla \mathcal{L}\left(\mathbf{w}^t\right)$ in neural networks
  - Analytical expression of the gradient straightforward: **chain rule**
  - **BUT:** efficient evaluation of gradient can be very tricky



$\Rightarrow$ **Efficient backprop implementation far from trivial**

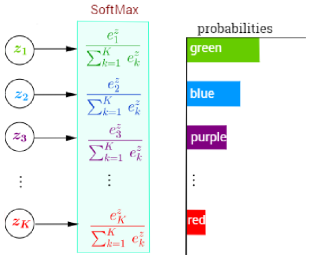
1. **Numerical Optimization Issues**
2. **Automatic Differentiation**

Numerical Optimization Issues

- Accumulating approximations (rounding) can be problematic
- Underflow: $x \approx 0$ or $x = 0$: different behaviors
  - Division by 0 $\Rightarrow$ Nan
- Overflow: large > 0 or < 0 numbers $\Rightarrow$ Nan
- Ex: softmax on $\mathbf{x} = \{x_1, x_2, ..., x_K\}$:

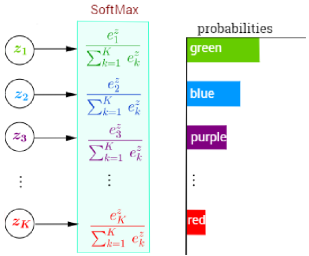
$$SM(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

- What if $x_i = C \ \forall i$, and:
  - $C \to +\infty$?
  - $C \to -\infty$?



Numerical Optimization Issues

$$SM(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

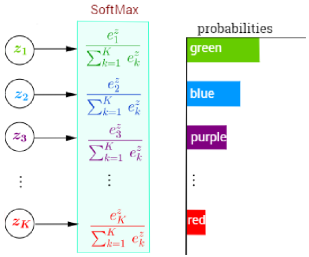


- If $x_i = C \ \forall i \ SM(\mathbf{x})_i = \frac{1}{K} \ \forall i$ expected
  - $C \to -\infty \Rightarrow e^C \to 0$: division by 0, Nan! (underflow)
  - $C \to +\infty \Rightarrow e^C \to +\infty$: Nan! (overflow)
- Numerical stabilization for denominator:

$$\mathbf{z} = \mathbf{x} - max_i(x_i) \Rightarrow SM(\mathbf{z}) = SM(\mathbf{x})$$

  - $max_i(e^{z_i}) = 1 \Rightarrow$ no overflow
  - $max_i(e^{z_i}) = 1 \Rightarrow$ no underflow

Numerical Optimization Issues



- $SM(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$, $\mathbf{z} = \mathbf{x} - max_i(x_i) \Rightarrow SM(\mathbf{z}) = SM(\mathbf{x})$
- What if we compute $log\,[SM(\mathbf{z})]$, eg cross-entrpoy loss?
  - $SM(\mathbf{z})$ can be 0 (underflow) $\Rightarrow log\,[SM(\mathbf{z})] \rightarrow -\infty$: Nan!
- Solution: stabilize $log\,[SM(\mathbf{x})_i] = x_i - log\left[\sum_{j=1}^{K} e^{x_j}\right]$
  - Same solution: $\mathbf{z} = \mathbf{x} - max_i(x_i)$

  $\Rightarrow log\,[SM(\mathbf{z})_i] = log\,[SM(\mathbf{x})_i]$?
  $\Rightarrow$ Underflow, overflow?

Computing Derivatives

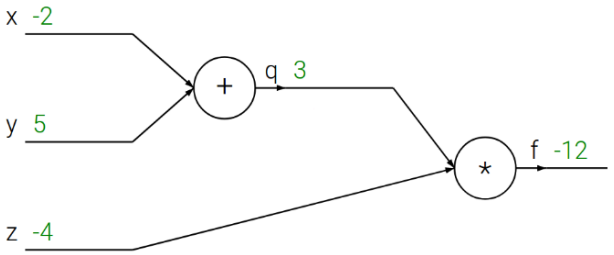
- Numerical approximation: $f'(x) \approx \frac{f(x+h)*f(x)}{h}$?
  - ⊖ Approximate, numerical issues (underflow/overflow)
- Symbolic differentiation?
  - ⊖ Rapidly: huge expressions, many duplicated terms ⇒ lack of efficiency

$f(x)$	$f'(x)$	$f'(x)$
$64x(1-x)(1-2x)^2(1-8x+8x^2)^2$	$128x(1-x)(-8+16x)(1-2x)^2(1-8x+8x^2)+64(1-x)(1-2x)^2(1-8x+8x^2)^2-64x(1-2x)^2(1-8x+8x^2)^2-256x(1-x)(1-2x)(1-8x+8x^2)^2$	$64(1-42x+504x^2-2640x^3+7040x^4-9984x^5+7168x^6-2048x^7)$

- **Automatic differentiation**
  - **Interleave symbolic differentiation and simplification steps**
  - **Symbolic differentiation at the elementary operation level keep intermediate numerical results**

Automatic Differentiation (AD)

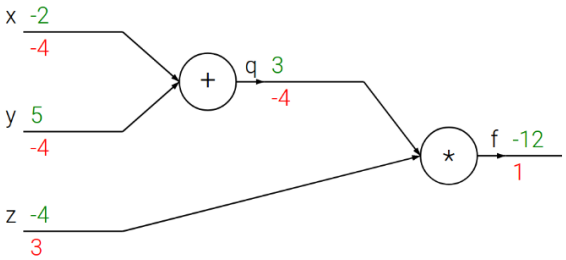
- Computation graph: core abstraction for computing gradient with backprop
- Ex[a]: $f(x, y, z) = (x + y) * z$
- Ultimate goal: compute numerical values $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$, $\frac{\partial f}{\partial z}$
- $x = -2$, $y = 5$, $z = -4$, forward prop $\Rightarrow$ $q = 3$, $f = -12$



[a]From Stanford course: http://cs231n.github.io/optimization-2/

Computation Graph & backprop

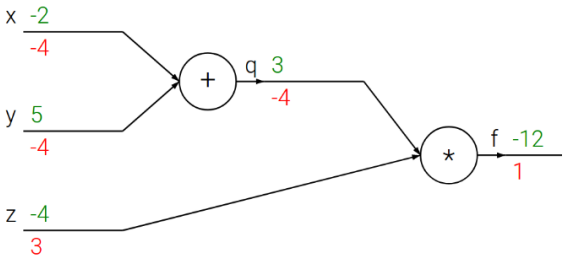
- $\frac{\partial f}{\partial f} = 1$, back-prop $\Rightarrow \frac{\partial f}{\partial q}, \frac{\partial f}{\partial z}$
- $f = q \star z \Rightarrow \frac{\partial f}{\partial q} = z = -4, \boxed{\frac{\partial f}{\partial z} = q = 3}$, back-prop $\Rightarrow \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}$
- $q = x + y \Rightarrow \boxed{\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x} = -4 \star 1 = -4}$ $\boxed{\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y} = -4 \star 1 = -4}$



Computation Graph (CG) & backprop

- Automatic differentiation with CG: Forward + backward pass
  - Backward pass: recursively compute derivatives from top → bottom
  - **Dynamic programming**, big speed-up wrt naive forward-mode differentiation
  - ex: Forward forward-mode differentiation from x: $\frac{\partial x}{\partial x} = 1$, $\frac{\partial q}{\partial x} = 1$, $\frac{\partial f}{\partial x} = -4$

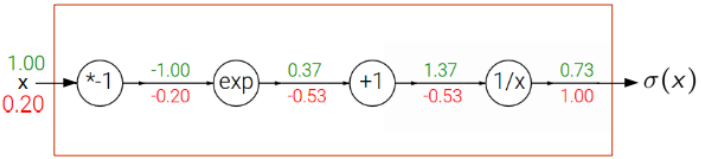
⇒ **N nodes: N forward passes *vs* 1 forward + 1 backward for backprop**



Computation Graph (CG) & backprop

- Symbolic differentiation only at atomic operation level, *e.g.* binary arithmetic operators[a], exp, log, trigonometric functions
- Include block with known derivative which is well numerically behaved
- Ex: sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$, $\sigma'(x) = \sigma(x)\,[1-\sigma(x)]$

$x = 1.0 \Rightarrow \sigma(x) = 0.73,\ \sigma'(x) = 0.2$



[a] multiplication, addition, subtraction, division, *etc*

Implementation Issues: Conclusion

- **Efficient Implementation of back-prop in deep learning: very tricky! Computing efficient & robust derivatives?**
  - AD: combines atomic symbolic differentiation with recursive back-propagation of numerical gradients
- **Software / libraries implementing these concepts?**
  ⇒ following!