backprop

December 8, 2024

1. Reverse Mode Automatic Differentiation

[ ]:
```
d)
```

[11]:
```
import torch

x = torch.tensor(3.0, requires_grad=True)
c = torch.tensor(5.0, requires_grad=True)

log_x = torch.log(x)
div= (x**2) / log_x
mult = (div + c) * (div - c)

mult.backward()

grad_x = x.grad
grad_c = c.grad

print(f"Output: {mult.item()}")
print(f"Gradient w.r.t x: {grad_x.item()}")
print(f"Gradient w.r.t c: {grad_c.item()}")
```

```
Output: 42.11137008666992
Gradient w.r.t x: 48.756893157958984
Gradient w.r.t c: -10.0
```

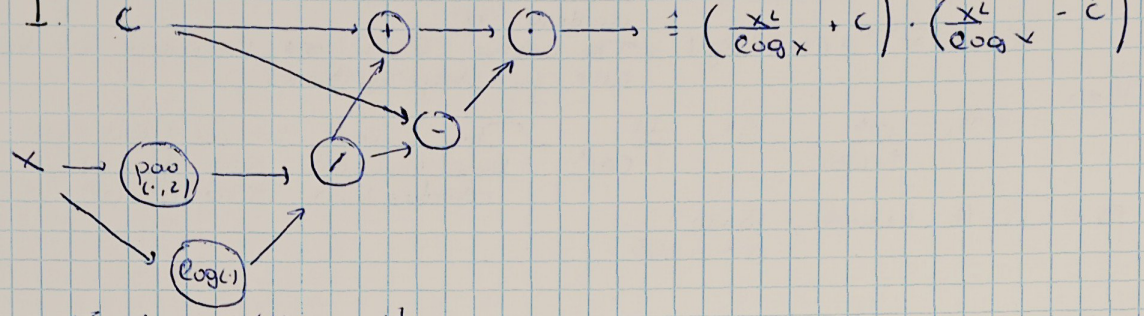
[ ]:
```

```

Exercise Sheet 7

Arsen Nuramatov
Pylyp Filippov

1.

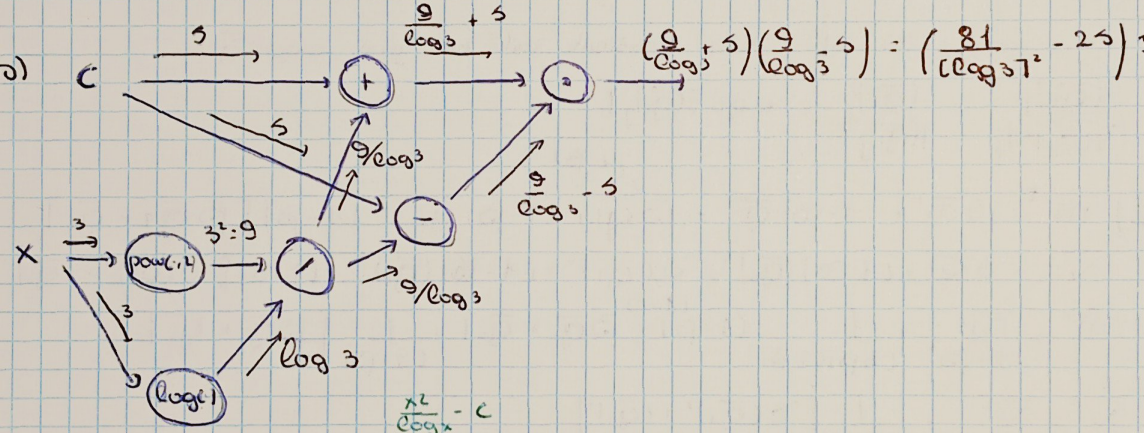


a) $\frac{d}{dx}\left[\left(\frac{x^2}{\log x}+c\right)\left(\frac{x^2}{\log x}-c\right)\right] =$

$= \left[\frac{\partial\left[\frac{x^2}{\log x}\right]}{\partial x^2}\frac{dx^2}{dx} + \frac{\partial\left[\frac{x^2}{\log x}\right]}{\partial \log x}\frac{d\log x}{dx}\right] \times$

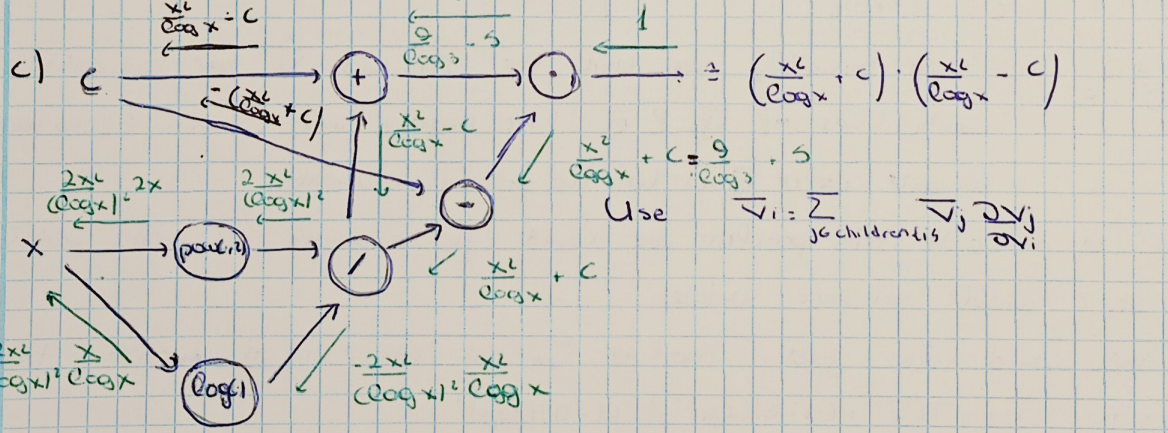
$\times \left[\frac{\partial\left[\left(\frac{x^2}{\log x}+c\right)\left(\frac{x^2}{\log x}-c\right)\right]}{\partial\left[\frac{x^2}{\log x}+c\right]}\frac{\partial\left[\frac{x^2}{\log x}+c\right]}{\partial\left(\frac{x^2}{\log x}\right)} + \frac{\partial\left[\left(\frac{x^2}{\log x}+c\right)\left(\frac{x^2}{\log x}-c\right)\right]}{\partial\left[\frac{x^2}{\log x}-c\right]}\frac{\partial\left[\frac{x^2}{\log x}-c\right]}{\partial\left(\frac{x^2}{\log x}\right)}\right]$

b)



$\left(\frac{9}{\log 3}+5\right)\left(\frac{9}{\log 3}-5\right) = \left(\frac{81}{(\log 3)^2}-25\right) \approx 42.11$

c)



$= \left(\frac{x^2}{\log x}+c\right)\cdot\left(\frac{x^2}{\log x}-c\right)$

Use $\overline{v_i} = \sum_{j \in \text{children of } i} \overline{v_j}\frac{\partial v_j}{\partial v_i}$

$\frac{d}{dx}\left[\left(\frac{x^2}{\log x}+c\right)\cdot\left(\frac{x^2}{\log x}-c\right)\right] = \frac{2x^2}{(\log x)^3}\left[2x\log x - x\right]\Big|_{x=3} = \frac{18}{(\log 3)^2}\left[6\log 3 - 3\right] \approx 48.76$

$\frac{d}{dc}\left[\left(\frac{x^2}{\log x}-c\right)\left(\frac{x^2}{\log x}+c\right)\right] = \frac{x^2}{\log x} - c - \frac{x^2}{\log x} - c = -2c\Big|_{c=5} = -10$

~~I would say that in this case symbolic differentiation wrg~~

If we compare b) with the result in a) we see that c) might look a bit easier. But at the end, we're basically doing the same thing (chain rule).

2. a) $\vec{w}^{t+1} = \vec{w}^t - \alpha \frac{\hat{\vec{m}}^t}{\sqrt{\hat{\vec{v}}} + \varepsilon}$ → In each iteration, we move the parameters in certain direction multiplied by the learning rate

$\vec{m}^t = \beta \vec{m}^{t-1} + (1-\beta)\vec{g}^t$ → We modify the previous direction using the gradient $\vec{g}^t$

$\vec{v}^t = \gamma \vec{v}^{t-1} + (1-\gamma)(\vec{g}^t)^2$ → The vector $\vec{v}^t$ is used to reduce the learning rate for steep gradients

Also

$\hat{\vec{m}}_t = \frac{\vec{m}^t}{1-(\beta)^t}$, $\hat{\vec{v}}^t = \frac{\vec{v}^t}{1-(\gamma)^t}$ → This is used to counteract the bias towards zero for the initialization $\vec{m}^0 = 0$, $\vec{v}^0 = 0$

$\beta, \gamma \in [0,1[$. Usually $\beta \approx 0.9$
$\gamma \approx 0.999$

b) First iteration

$\vec{m}^1 = \beta \vec{m}^0 + (1-\beta)\vec{g}^1 = (1-\beta)\vec{g}^1$

$\vec{v}^1 = \gamma \vec{v}^0 + (1-\gamma)(\vec{g}^1)^2 = (1-\gamma)|\vec{g}^1|^2$

$\hat{\vec{m}}^1 = \frac{\vec{m}^1}{1-\beta} = \frac{1}{1-\beta}(1-\beta)\vec{g}^1 = \vec{g}^1$

$\hat{\vec{v}}^1 = \frac{(1-\gamma)(\vec{g}^1)^2}{1-\gamma} = (\vec{g}^1)^2$

$\frac{\hat{\vec{m}}^1}{\sqrt{\hat{\vec{v}}} + \varepsilon} = \frac{\vec{g}^1}{|\vec{g}^1| + \varepsilon} = \frac{\vec{g}^1}{\sqrt{(\vec{g}^1)^2} + \varepsilon} = \frac{\vec{g}^1}{|\vec{g}^1| + \varepsilon} \approx \frac{\vec{g}^1}{|\vec{g}^1|} = \frac{\vec{g}^1}{\sqrt{(\vec{g}^1)^2}}$ ($\varepsilon \ll |\vec{g}^1|$)

Bearing in mind that the operations are element-wise, we conclude that the components of the gradient will be just $\text{sign}(\vec{g}^1)$

$\frac{(\vec{g}^1)_j}{(\sqrt{(\vec{g}^1)^2})_j} = \frac{(\vec{g}^1)_j}{|(\vec{g}^1)_j|} = \text{sign}[(\vec{g}^1)_j]$

c) $\vec{m}^2 = \beta \vec{m}^1 + (1-\beta)\vec{g}^2 = \beta(1-\beta)\vec{g}^1 + (1-\beta)\vec{g}^2 = [1-\beta][\beta \vec{g}^1 + \vec{g}^2]$

$\vec{v}^2 = \gamma \vec{v}^1 + (1-\gamma)(\vec{g}^2)^2 = \gamma(1-\gamma)(\vec{g}^1)^2 + (1-\gamma)(\vec{g}^1)^2 = [1-\gamma][\gamma(\vec{g}^1)^2 + (\vec{g}^1)^2]$

$\hat{\vec{m}}^2 = \frac{\vec{m}^2}{1-\beta^2} = \frac{1}{(1+\beta)(1-\beta)}(1-\beta)[\beta \vec{g}^1 + \vec{g}^2] = \frac{1}{(1+\beta)}[\beta \vec{g}^1 + \vec{g}^2]$

$\hat{\vec{v}}^2 = \frac{\vec{v}^2}{1-\gamma^2} = \frac{1}{(1+\gamma)}[\gamma(\vec{g}^1)^2 + (\vec{g}^1)^2]$

$\frac{\hat{\vec{m}}^1}{\sqrt{\hat{\vec{v}}} + \varepsilon}$ → We basically see that the $\hat{\vec{m}}^2$ and $\hat{\vec{v}}^2$ are weighted combinations of $\vec{g}^1$ & $\vec{g}^2$. The bigger are β and γ, the more weight were given to the previous gradient

d) To mitigate this problem we could for example set $\vec{m}^0$ and $\vec{v}^0$ to some small non-zero values

We could also just modify $\vec{v}^0$ to some initial constant big enough so $\vec{v}^1 = \gamma \vec{v}^0 + (1-\gamma)(\vec{g}^1)^2 \neq (1-\gamma)(\vec{g}^1)^2$

basically $|\gamma \vec{v}^0| \approx |(1-\gamma)(\vec{g}^1)^2|$

e) Yes, there is a difference.

$\mathcal{L}_{Tot}(w) = \mathcal{L}_{orig.}(w) + \lambda \|w\|_2^2$ } The penalty explicitly contributes to the gradient.

$\frac{\partial \mathcal{L}_{tot}}{\partial w} = \frac{\partial \mathcal{L}_{orig.}}{\partial w} + 2\lambda w$

$\vec{w}^{t+1} = \vec{w}^t - \alpha \frac{\hat{\vec{m}}^t}{\sqrt{\hat{\vec{v}}} + \varepsilon} - \alpha \lambda \vec{w}^t$ } In this case we scale down the weight during each step, independently of the gradient.

We can conclude that the second option is better (weight decay applied to weights) because it is applied uniformly across all parameters, while in the in L-2 regularization the reg. term is affected by β dumis adaptive learning rates. Basically it is better to decouple the weight decay from gradient update

3 Receptive Field of VGG16

(a)

$$r_l = (r_{l+1} - 1) * s_{l+1} + k_{l+1}$$

Where

- r_l – the size of the receptive field of a pixel on layer l
- s_l – stride
- k_l – kernel size

Convolutional layers in VGG16 have $k = 3,\ s = 1$, while max pooling layers can be represented as convolutional layers with $k = 2,\ s = 2$ with constant weights.

Layer	Receptive field
18. MaxPool (Output):	$r_{18} = 1$
17. Conv:	$r_{17} = (r_{18} - 1) * 2 + 2 = r_{18} * 2 = 2$
16. Conv:	$r_{16} = (r_{17} - 1) * 1 + 3 = r_{17} + 2 = 4$
15. Conv:	$r_{15} = 6$
14. MaxPool:	$r_{14} = 8$
13. Conv:	$r_{13} = 16$
12. Conv:	$r_{12} = 18$
11. Conv:	$r_{11} = 20$
10. MaxPool:	$r_{10} = 22$
9. Conv:	$r_9 = 44$
8. Conv:	$r_8 = 46$
7. Conv:	$r_7 = 48$
6. MaxPool:	$r_6 = 50$
5. Conv:	$r_5 = 100$
4. Conv:	$r_4 = 102$
3. MaxPool:	$r_3 = 104$
2. Conv:	$r_2 = 208$
1. Conv:	$r_1 = 210$
0. Input:	$r_0 = 212$

(b)

```
s = lambda x, y: x**2 + y
n_params_conv = 3 * (64 * 2 + 128 * 2 + 256 * 3 + 512 * 3 + 512 * 3)
n_params_fc = (512 * 7 * 7) * (4096) + 4096**2 + 4096 * 1000
print("Total parameters:", "%.5E" % (n_params_conv + n_params_fc))
print("Ratio:", "%.5E" % (n_params_conv / n_params_fc))
```

Total parameters: 1.23646E+08 Ratio: 1.02496E-04