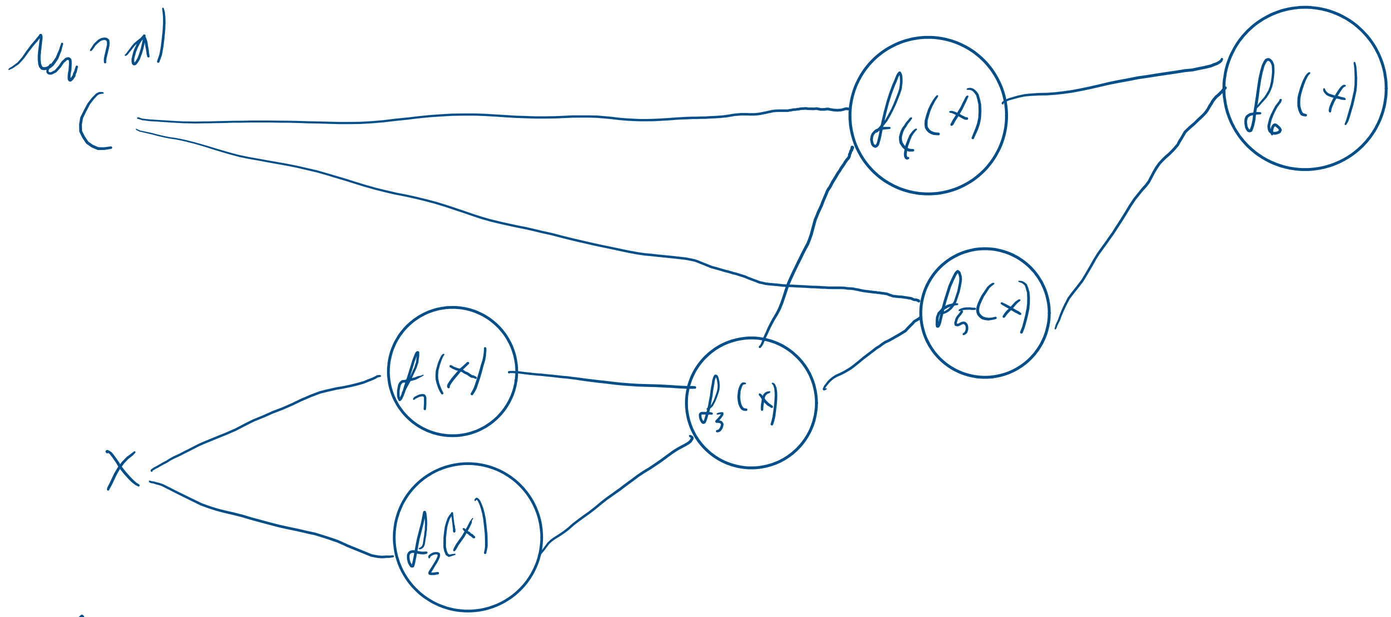
Sheet 7 Tim, Julian, Jan

Dienstag, 19. November 2024 11:03



$$\frac{\partial f_1(x)}{\partial x} = 2x \qquad \frac{\partial f_2(x)}{\partial x} = \frac{1}{x}$$

$$\frac{\partial f_3}{\partial x} = \frac{\partial \left(\frac{f_1(x)}{f_2(x)} \right)}{\partial x} = \frac{\partial f_1(x)}{\partial x} \cdot \frac{1}{f_2(x)} + \frac{1}{f_2^2(x)} \frac{\partial f_2(x)}{\partial x} \cdot f_1(x)$$

$$\frac{\partial f_4(x)}{\partial x} = \frac{\partial f_3(x)}{\partial x} + \frac{\partial c}{\partial x} \qquad \frac{\partial f_5(x)}{\partial x} = \frac{\partial f_3(x)}{\partial x} - \frac{\partial c}{\partial x}$$

$$\frac{\partial f_6(x)}{\partial x} = \frac{\partial f_4(x)}{\partial x} f_5(x) + \frac{\partial f_5(x)}{\partial x} f_4(x)$$

b) $x = 3 \quad c = 5$

$f_1(3) = 9$ $\qquad$ $f_2(3) = \log(3) \approx 1{,}099$

$f_3(3) = \frac{9}{\log(3)} \approx 8{,}192$ $\qquad$ $f_4(3,5) = \frac{9}{\log(3)} + 5 \approx 13{,}192$

$f_5(3,5) = \frac{9}{\log(3)} - 5 \approx 3{,}192$ $\qquad$ $f_6(3,5) = \left(\frac{9}{\log(3)} + 5\right)\left(\frac{9}{\log(3)} - 5\right) \approx 42$

c) 1. $\frac{\partial f_6}{\partial f_5} = f_4 \approx 13{,}192$ $\qquad$ $\frac{\partial f_6}{\partial f_4} = f_5 \approx 3{,}192$

2. $\frac{\partial f_5}{\partial f_3} = 1$ $\qquad$ $\frac{\partial f_4}{\partial f_3} = 1$

$$\frac{\partial f_6}{\partial f_3} = \frac{\partial f_6}{\partial f_4} \frac{\partial f_4}{\partial f_3} + \frac{\partial f_6}{\partial f_5} \frac{\partial f_5}{\partial f_3} \approx 16{,}384$$

3. $\frac{\partial f_3}{\partial f_1} = \frac{1}{f_2}$ $\qquad$ $\frac{\partial f_3}{\partial f_2} = \frac{f_1}{f_2^2}$

$\frac{\partial f_6}{\partial f_3} \frac{\partial f_3}{\partial f_1} \approx 14{,}91$ $\qquad$ $\frac{\partial f_6}{\partial f_3} \frac{\partial f_3}{\partial f_2} \approx -121{,}63$

$\frac{\partial f_1}{\partial x} = 2x = 6$ $\qquad$ $\frac{\partial f_2}{\partial x} = \frac{1}{x} = \frac{1}{3}$

$$\frac{\partial f_6}{\partial f_1} \frac{\partial f_1}{\partial x} + \frac{\partial f_6}{\partial f_2} \frac{\partial f_2}{\partial x} \approx 48{,}8$$

$\Rightarrow$ backward is easier, no explicit derivations required

A2 (1) $w^{t+1} = w^t - \alpha \frac{\hat{m}^t}{\sqrt{\hat{v}} + \varepsilon}$

(2) $m^t = \beta m^{t-1} + (1-\beta) g^t$

(3) $v^t = \gamma v^{t-1} + (1-\gamma)(g^t)^2$

(4) $\hat{m}^t = \frac{m^t}{1-(\beta)^t}$, (5) $\hat{v}^t = \frac{v^t}{1-(\gamma)^t}$,

(1) update of the weights

$\frac{\alpha}{\sqrt{\hat{v}}+\varepsilon}$ rescales the learning rate

ε avoids division by 0

(2) updates momentum by involving the previous gradient

(3) reduces the learning rate for steep gradients

(4),(5) avoids bias through initalization for $m^0 = 0$ and $v^0 = 0$

b) $m^1 = (1-\beta) g^1$ $\qquad v^1 = (1-\gamma)(g^1)^2$

$\hat{m}^1 = g^1$ $\qquad \hat{v}^1 = (g^1)^2$

$$w^1 = w^0 - \alpha \frac{g^1}{\sqrt{(g^1)^2}} = w^0 - \alpha \left(\frac{g_1^1}{|g_1^1|}, \frac{g_2^1}{|g_2^1|}, \dots \right)^T = w^0 - \alpha \operatorname{sign} g^1$$

component wise

c) $m^2 = \beta(1-\beta) g^1 + (1-\beta) g^2$

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

$v^2 = \gamma(1-\gamma)(g^1)^2 + (1-\gamma)(g^2)^2$

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

$$w^2 = w^1 - \lambda \, \text{sign}\, g^1 - \alpha \frac{(\beta g^1 + g^2)\sqrt{1+\gamma}}{(1+\beta)\sqrt{\gamma(g^1)^2 + (g^2)^2}}$$

d) Train more than one epoche!

e) Yes, it makes a difference. Including $\|w\|_2^2$ in the loss affects the gradient computation, causing regularization to interact with Adam's adaptive learning rate.
In contrast, AdamW applies weight decay directly during update, decoupling regularization from optimization. This leads to more stable training and better generalization, making AdamW the preffered approach.

Exercise07_Kusch_Reckermann_Weinreich_code

December 5, 2024

1 7.1

1.1 (d)

[2]:
```python
import torch
```

[4]:
```python
x = torch.tensor(3.0, requires_grad=True)
c = torch.tensor(5.0, requires_grad=True)

f1 = x**2
f2 = torch.log(x)
f3 = f1 / f2
f4 = f3 + c
f5 = f3 - c
f6 = f4 * f5

# Backward computation
f6.backward()

# Print gradients
print(f"df6/dx: {x.grad}")
```

```
df6/dx: 48.756893157958984
```