Mathematical Foundations of Machine Learning
Assignment 1

David Felipe Alvear Goyes

1. Gradient of binary logistic loss.
Derive that the gradient of the cross entropy loss function used in binary logistic regression takes the form:

$$\nabla J = \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n - y_n)x_n$$

Explain briefly what x, y, and $\hat{y}$ are and how they are obtained or computed.

binary logistic regression $\rightarrow$ $\hat{y} = P(y|x;\theta) = Ber(y \mid \sigma(w^T x + b))$

$\sigma(x) = \dfrac{e^x}{1+e^x}$

$Ber(y|\mu) = \begin{cases} \mu & \text{for } y=1 \\ 1-\mu & \text{for } y=0 \end{cases}$

$\dfrac{d\sigma}{dx} = \sigma(x)(1-\sigma(x))$

Cross entropy loss $\rightarrow$ $H(p,q) = -\sum_x p(x)\log(q(x))$ $\rightarrow$ $q(x)$ estimated distribution

$\rightarrow$ Now for binary-logistic Regression $H(y,\hat{y}) = -\big(y(x)\underbrace{\log \hat{y}(x)}_{\text{term 1}} + (1-y(x))\underbrace{\log(1-\hat{y}(x))}_{\text{term 2}}\big)$

True distribution (y) Estimated distribution ($\hat{y}$)

Derivate gradient of the cross entropy loss

we have: $a_n = \theta^T x_n$ $\quad \hat{y}_n = \sigma(\theta^T x_n) \longrightarrow \dfrac{d\hat{y}_n}{d\theta_d} = \hat{y}_n(1-\hat{y}_n)X_{nd}$

$\rightarrow$ For term 1: $\nabla_\theta \log \hat{y}_n = \dfrac{1}{\hat{y}_n}\dfrac{d\hat{y}_n}{d\theta_d} = \dfrac{\cancel{\hat{y}_n}}{\cancel{\hat{y}_n}}(1-\hat{y}_n)X_n = (1-\hat{y}_n)X_n$

$\rightarrow$ For term 2: $\nabla_\theta \log(1-\hat{y}_n) = \dfrac{1}{1-\hat{y}_n}\dfrac{d(1-\hat{y}_n)}{d\theta_d} = -\dfrac{\hat{y}_n\cancel{(1-\hat{y}_n)}X_n}{\cancel{(1-\hat{y}_n)}} = -\hat{y}_n X_n$

$\nabla_\theta H(y,\hat{y}) = -\big(y\,\nabla_\theta \log\hat{y} + (1-y)\nabla_\theta \log(1-\hat{y})\big)$

$\rightarrow$ $-\big(y_n(1-\hat{y}_n)X_n - (1-y_n)\hat{y}_n X_n\big) = \big(\hat{y}_n \cancel{- y_n\hat{y}_n} - y_n \cancel{+ y_n\hat{y}_n}\big)X_n$

$$\boxed{\nabla_\Theta H(y,\hat{y}) = \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n - y_n)X_n}$$

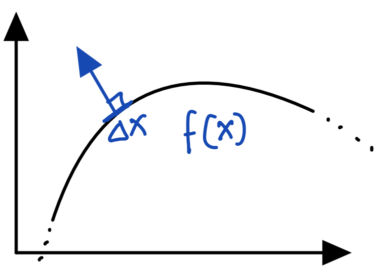
$\hat{y}_n \to$ Predicted label

$y \to$ True label

$X_n \to$ Feature value

2. Negative gradient direction.
Explain, mathematically, why the negative gradient direction is the local direction of steepest descent of the loss function.

→ Let $f(x)$, function that we want to minimize.



→ we want to find the direction u that minimize $D_u f(x)$ the directional derivative.

$$D_u f(x) = \nabla f(x)\, u$$

→ u: unit direction vector

→ using Cauchy-Schwarz inequality:

$$|\nabla f(x)\, u| \leq \|\nabla f(x)\| \cdot \|u\|$$

$$\boxed{|D_u f(x)| \leq \|\nabla f(x)\|}$$

$$\nabla f(x) . u = \|\nabla f(x)\| \|u\| \cos\Theta \leq \|\nabla f(x)\|$$

→ to maximize: $\cos\Theta = 1 \to \Theta = 0$

then $u = \dfrac{\nabla f(x)}{\|\nabla f(x)\|}$

→ to minimize: $\cos\Theta = -1$

then: $u = -\dfrac{\nabla f(x)}{\|\nabla f(x)\|}$

→ Hence $-\nabla f(x)$ is the direction of steepest descent.

$$\boxed{|D_u f(x)| \geq -\|\nabla f(x)\|}$$

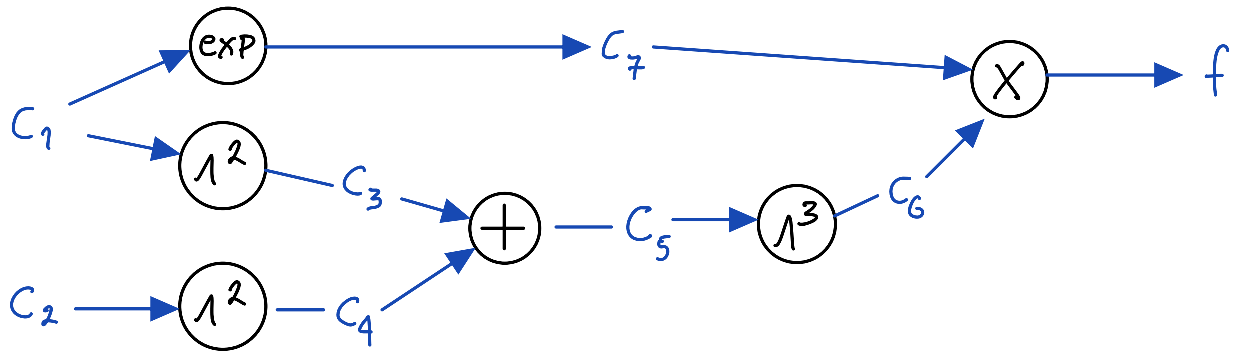
3. Automatic differentiation.

We consider the 2-variable scalar function

$$f(x) = \exp(x_1)\,(x_1^2 + x_2^2)^3$$

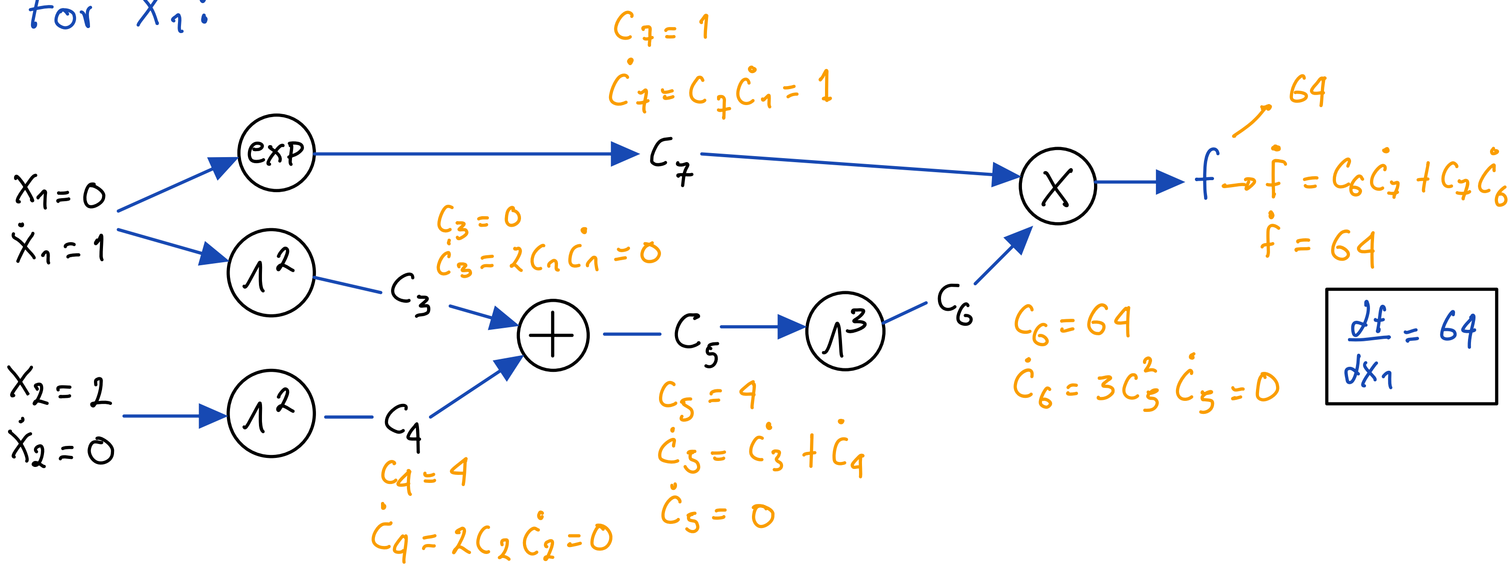
- generate the computational graph of this expression (graph starts with $c_1 = x_1$, $c_2 = x_2$, and ends with $f = c_7$)
- use forward mode differentiation to compute $\nabla f(0, 2)$. Show your intermediate values on the graph.
- use reverse mode differentiation to compute the same gradient. Show your intermediate values (values of the adjoint variables $\bar{c}_i = \partial f / \partial c_i$).
- use the `backward()` method of pytorch to verify your answers.
- comment briefly on the computational effectiveness of forward vs reverse modes of computing these derivatives.

Computational Graph

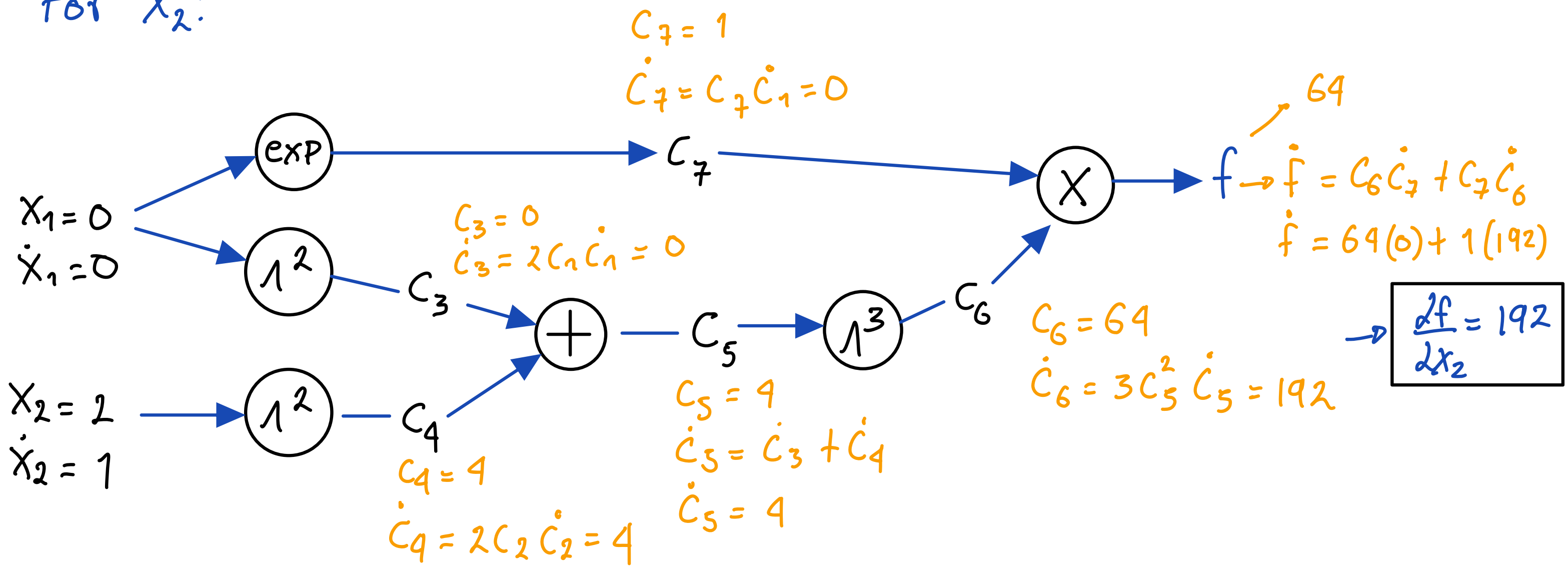


→ Compute $\nabla f(0,2)$ → $x_1 = 0$, $x_2 = 2$ using Forward Mode Differentiation

For x_1:

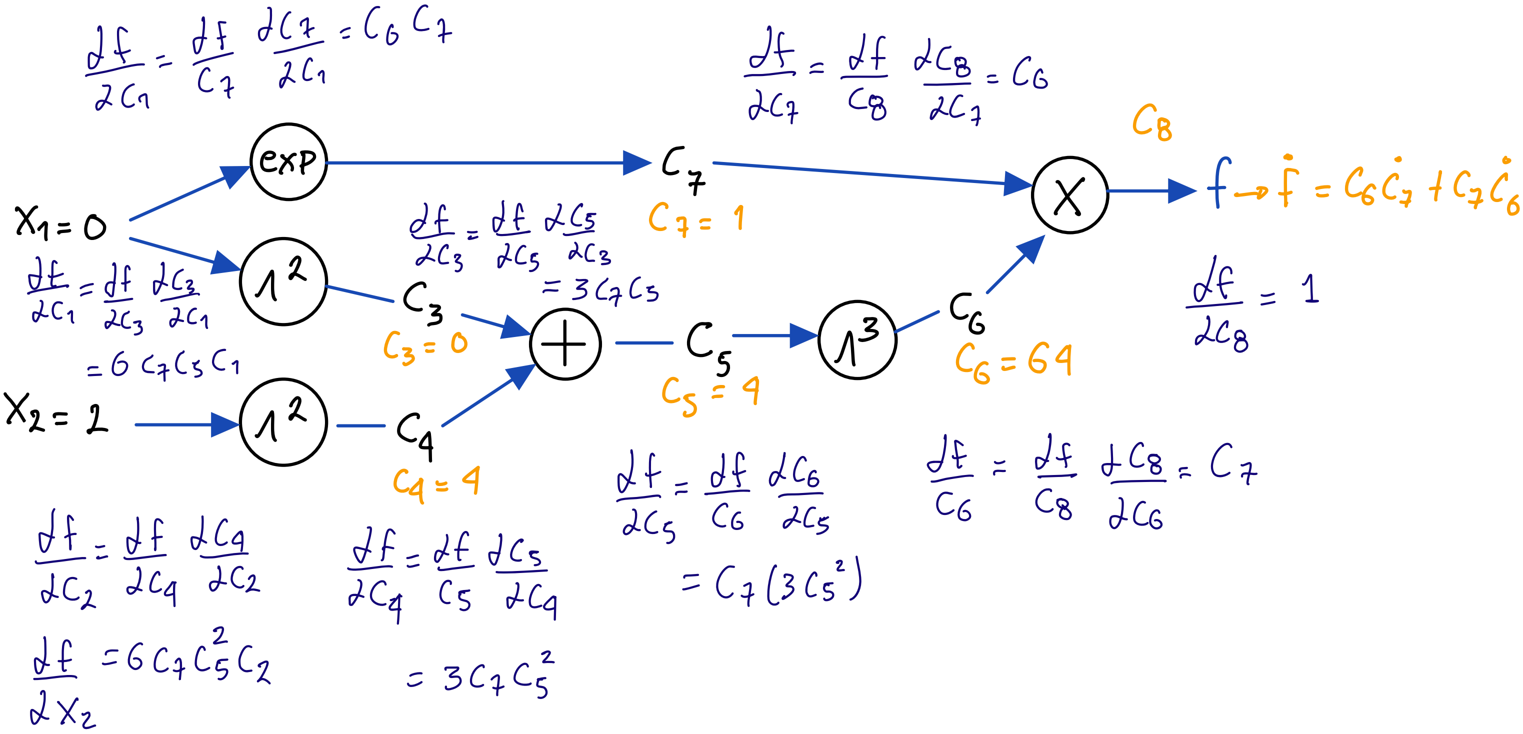


For x_2:



Then: using Forward Mode differentiation $\nabla f(0,2) = [64, 192]$

→ Compute $\nabla f(0,2)$ → $x_1 = 0$, $x_2 = 2$ using Reverse Mode Differentiation



→ Then:

$$\frac{df}{dx_1} = \frac{df}{C_1} + \frac{df}{C_1} = \frac{df}{dC_3}\frac{dC_3}{dC_1} + \frac{df}{dC_7}\frac{dC_7}{dC_1} = 64$$

$$\frac{df}{dx_2} = \frac{df}{dC_4}\frac{dC_4}{dC_2} = 6C_7C_5^2C_2 = 192$$

```python
import torch
# Define variables
x1 = torch.tensor(0.0, requires_grad=True)
x2 = torch.tensor(2.0, requires_grad=True)

# Compute function
f = torch.exp(x1) * (x1 ** 2 + x2 ** 2) ** 3
f.backward()

# Get gradients
print(x1.grad)
print(x2.grad)
```

```
tensor(64.)
tensor(192.)
```

- The forward mode is computational expensive when high dimensional input cases. It's needed to run as many forward passes as there are input variables.

- The reverse mode compute all the gradients in the backward Pass, more efficient than the forward mode. However it requires to store the values of the forward pass. needing more memory.