2) Back Propogation

$\mathcal{L}(\theta)$ θ parámetros

$\mathcal{L}$ función de error que depende de los parámetros

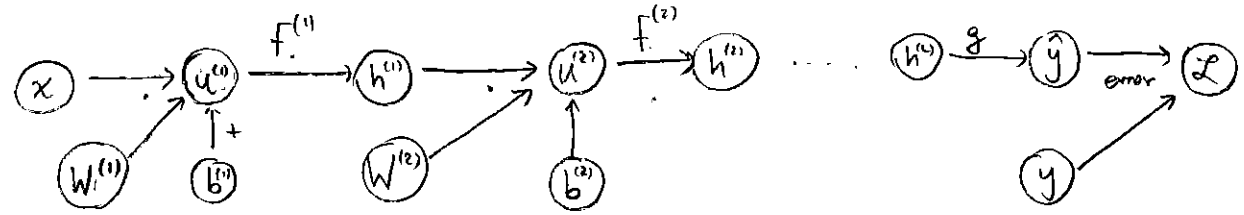
$\{(x^{(1)}, y^{(1)}), \dots (x^{(N)}, y^{(N)})\}$ $\hat{y}^{(i)} = \text{forward}(x^{(i)})$

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \text{error}(\hat{y}^{(i)}, y^{(i)})$$

queremos $\nabla_\theta \mathcal{L}(\theta) = \left(\frac{\partial \mathcal{L}}{\partial \theta_1}, \frac{\partial \mathcal{L}}{\partial \theta_2}, \dots, \frac{\partial \mathcal{L}}{\partial \theta_r} \right)$ y evaluarlo en un punto $\hat{\theta}$ para poder descender

Back propogation es simplemente un método para calcular $\nabla_\theta \mathcal{L}(\theta)$ y evaluarlo eficientemente en el computador.

Primer concepto importante: Computational graph



Forma de organizar todas las operaciones que engloban lo paso de hacia adelante y el cálculo de la pérdida.

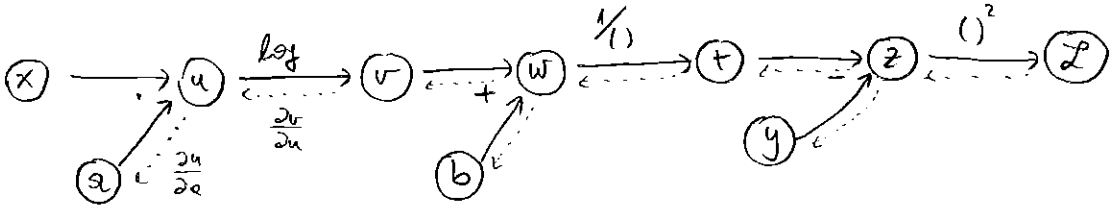
¿Como calculo $\frac{\partial \mathcal{L}}{\partial W_{ij}^{(2)}}$? $\mathcal{L} = \text{error}($

$g(f^{(L)}(f^{(L-1)}(\dots f^{(2)}(h^{(1)}W^{(2)} + b^{(2)})W^{(3)}\dots)))$

no parece una tarea fácil :(

Idea: usar agresivamente la regla de la cadena!

consideremos un computational graph simple (solo escalares, sin matrices)



$u = x \cdot a$
$v = \log(u)$
$w = v + b$
$t = 1/w$
$z = y - t$
$\mathcal{L} = z^2$

x, y inputs, a, b parámetros

¿ $\frac{\partial \mathcal{L}}{\partial a}$? idea 1) desarrollar la definición de $\mathcal{L}$

$$\mathcal{L} = \left(y - \frac{1}{b + \lg(x \cdot a)} \right)^2$$

idea 2) usar la regla de la cadena "desde atrás" recursivamente

¿ $\frac{\partial \mathcal{L}}{\partial a}$? $= \frac{\partial \mathcal{L}}{\partial u} \cdot \frac{\partial u}{\partial a} = \frac{\partial \mathcal{L}}{\partial u} \cdot x$

si tuviera $\frac{\partial \mathcal{L}}{\partial u}$ computado podría computar $\frac{\partial \mathcal{L}}{\partial a}$ fácilmente

$\frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial v} \cdot \frac{\partial v}{\partial u} = \frac{\partial \mathcal{L}}{\partial v} \cdot \frac{1}{u}$

$\frac{\partial \mathcal{L}}{\partial v} = \frac{\partial \mathcal{L}}{\partial w} \cdot \frac{\partial w}{\partial v} = \frac{\partial \mathcal{L}}{\partial w} \cdot 1$

$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial t} \cdot \frac{\partial t}{\partial w} = \frac{\partial \mathcal{L}}{\partial t} \cdot \frac{-1}{w^2}$

$\frac{\partial \mathcal{L}}{\partial t} = \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial t} = \frac{\partial \mathcal{L}}{\partial z} \cdot -1$

$\frac{\partial \mathcal{L}}{\partial z} = 2z$ listo ✓

¿ $\frac{\partial \mathcal{L}}{\partial b}$? MUY FÁCIL !

$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial w} \cdot \frac{\partial w}{\partial b} = \frac{\partial \mathcal{L}}{\partial w} \cdot 1$

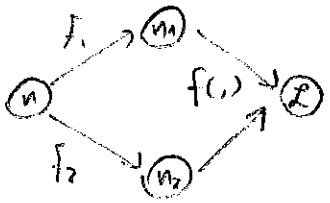
yo lo tengo ya calculado !!

Idea general: para cada nodo n en el grafo

primero calculo $\frac{\partial \mathcal{L}}{\partial n'}$ y luego lo uso para computar

$\frac{\partial \mathcal{L}}{\partial n} = \frac{\partial \mathcal{L}}{\partial n'} \cdot \frac{\partial n'}{\partial n}$ ← esta última es una derivada simple de calcular y de evaluar dados los datos computados durante la pasada de forward

¿Qué pasa si tengo algo como esto?



$n_1 = f_1(n)$

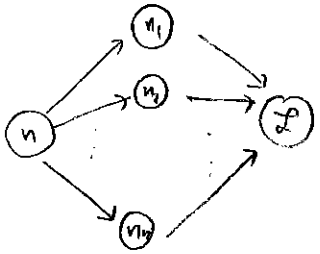
$n_2 = f_2(n)$

$\mathcal{L} = f(n_1, n_2)$

¿ $\frac{\partial \mathcal{L}}{\partial n}$?

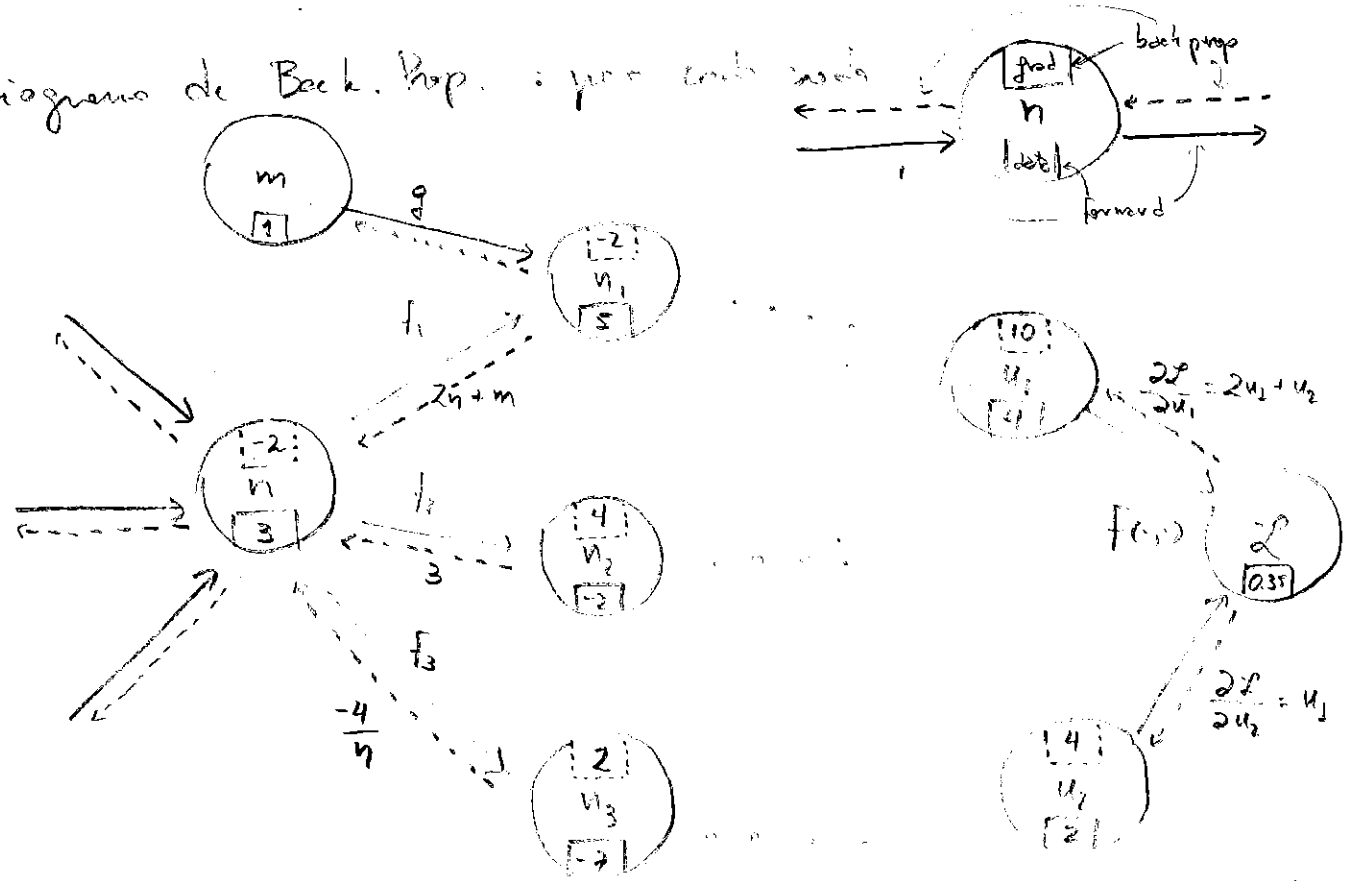
$$\frac{\partial \mathcal{L}}{\partial n} = \frac{\partial \mathcal{L}}{\partial n_1} \cdot \frac{\partial n_1}{\partial n} + \frac{\partial \mathcal{L}}{\partial n_2} \cdot \frac{\partial n_2}{\partial n}$$

y generalizo



$$\frac{\partial \mathcal{L}}{\partial n} = \sum_{i=1}^{k} \frac{\partial \mathcal{L}}{\partial n_i} \cdot \frac{\partial n_i}{\partial n}$$

Diagrama de Back. Prop. : para cada nodo



parámetros son "hojas" de este grafo (solo ellos se actualizan cuando [illegible] entre nodos)

Algoritmo de backpropagation (recursivo)

Necesitas:
- G : computational graph (DAG), $\mathcal{L}$ variable objetivo
- data[n] : para cada nodo $n \in G$ almacena el valor computado en forward (incluyendo inputs, parámetros y nodos intermedios)

Calcula:
- grad[n] para cada parámetro

grad[$\mathcal{L}$] := 1

rbprop(n):
  si grad[n] ≠ NULL retorna grad[n]
  si grad[n] = NULL
    sean $n_1, n_2, \ldots, n_k$ nodos que dependen de n en G
    grad[n] := $\sum_{i=1}^{k}$ rbprop(n_i) · $\frac{\partial n}{\partial n_i}$ (data)

para cada parámetro p, ejecuta rbprop(p)

esto es lo que Tensorflow / Theano Pytorch hacen automático

"$\frac{\partial n}{\partial n_i}$" es la única parte no trivial del anterior código y es lo que ayudan a hacer las librerías (TF, Theano, Pytorch)

Algoritmo Back Propagation (iterativo)

Necesita:
- G, $\mathcal{L}$, data (igual que antes)

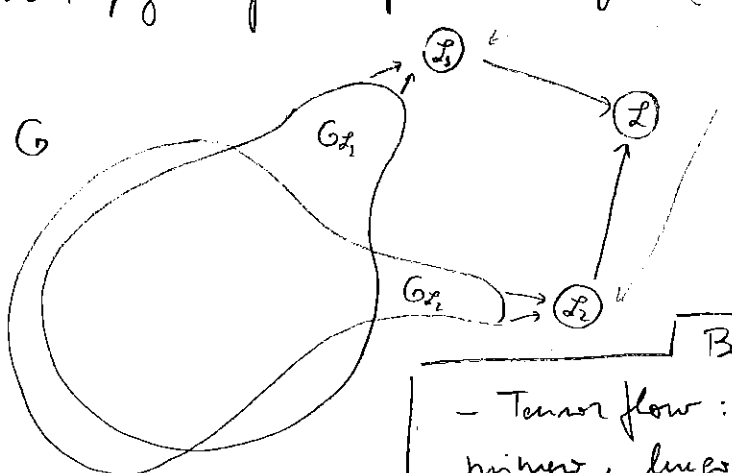
Calcula:
- grad[n] para cada n en G

bprop:

```
grad[L] := 1 , grad[n] := 0 para n ≠ L
mientras G tenga nodos sin marcar:
    n := nodo en G con todas sus aristas salientes marcadas
    para cada (n', n) ∈ G:
        grad[n'] += grad[n] · ∂n/∂n' (data)
        marca (n', n)
    marca n
```

(sigue siendo lo único no trivial)

Back Propagation puede pensarse más general



dos funciones objetivo (pérdida) podríamos hacer bprop sobre $G_{\mathcal{L}_1}$ y luego sobre $G_{\mathcal{L}_2}$, incluso podríamos acumular los gradientes

Back Propagation en la práctica.

- Tensorflow: se define el grafo de computación primero, luego se "compila" lo que calcula las derivadas parciales y se almacenan en otro grafo y luego se hace bprop.
- Pytorch: el grafo está implícito en las operaciones realizadas sobre datos/parámetros, las derivadas se calculan "on the fly" cuando se hace el bprop. (mucho más versátil!!!)