

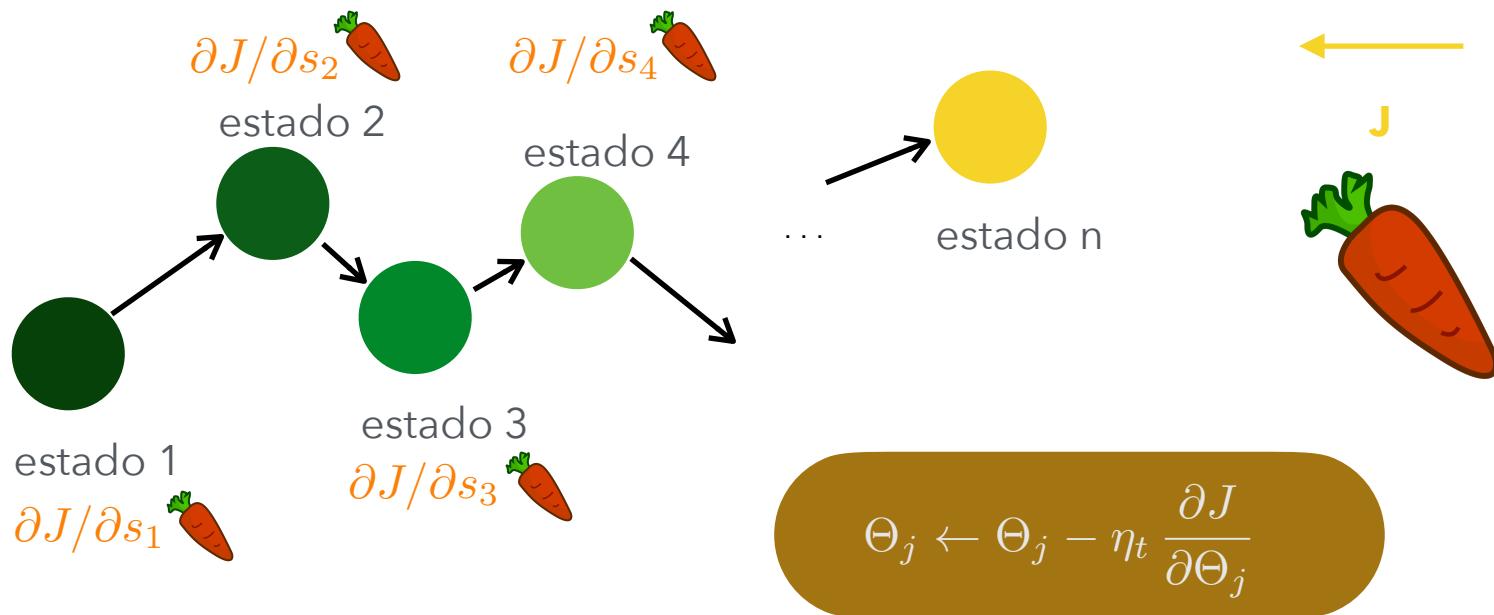
Entrenamiento Básica de Redes Neuronales

Backpropagation



Entrenamiento vía Gradientes

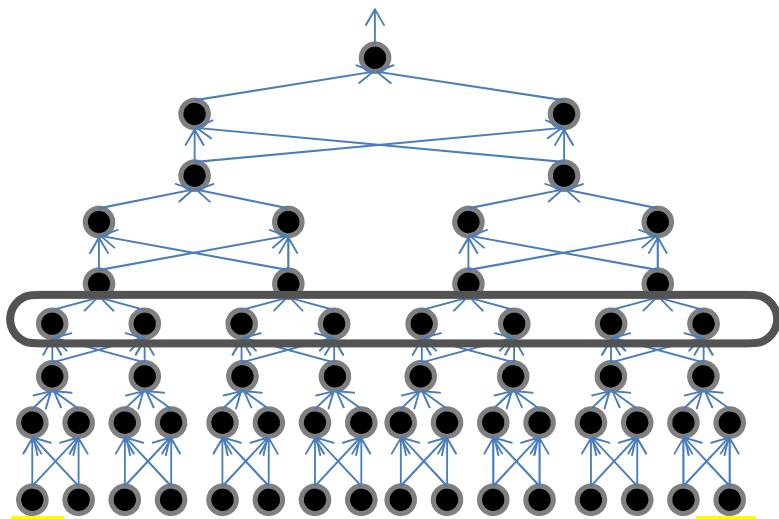
- Hemos visto que los nodos ocultos de la red se pueden entrenar usando **derivadas parciales**. Más específicamente, la idea es corregir un parámetro iterativamente en la dirección del negativo de la derivada parcial de objetivo respecto a ese parámetro.



Cálculo de las Derivadas Parciales

- **Sin un buen algoritmo, esta idea podría ser impracticable** cuando la red es muy profunda y densa. Para ver esto, intentemos hacer los cálculos, asumiendo que el objetivo es una media sobre el conjunto de entrenamiento (este es el caso en la mayoría de las aplicaciones).

$\Theta^{(\ell)}$ parámetros de la capa ℓ



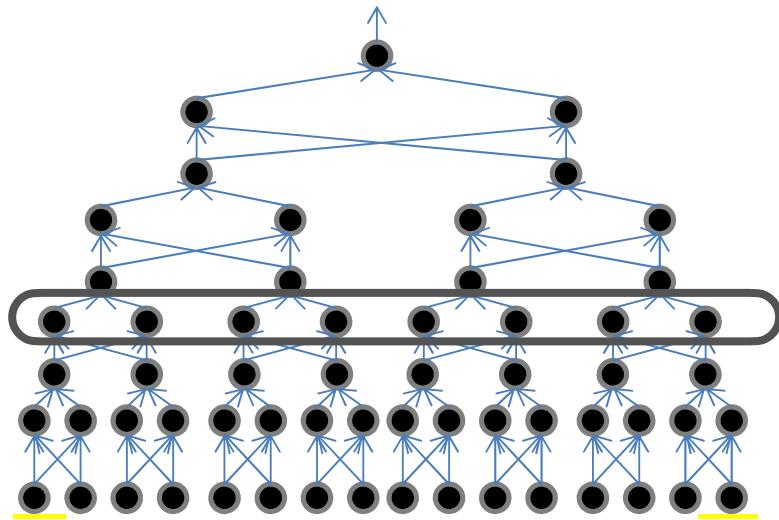
$$J(\Theta) = \frac{1}{n} \sum_{i=1}^n L(f(x^{(i)}), y^{(i)})$$

$$\frac{\partial J}{\partial \Theta_k^{(\ell)}} = \frac{1}{n} \sum_{i=1}^n \frac{\partial L(f(x^{(i)}), y^{(i)})}{\partial \Theta_k^{(\ell)}}$$

Cálculo de las Derivadas Parciales

- Notemos para simplificar la notación, que como la derivada requerida es una media sobre el conjunto de entrenamiento, nos basta saber calcular la derivada respecto de 1 ejemplo x genérico y luego vectorizamos esa función sobre todo el conjunto.

$\Theta^{(\ell)}$ parámetros de la capa ℓ



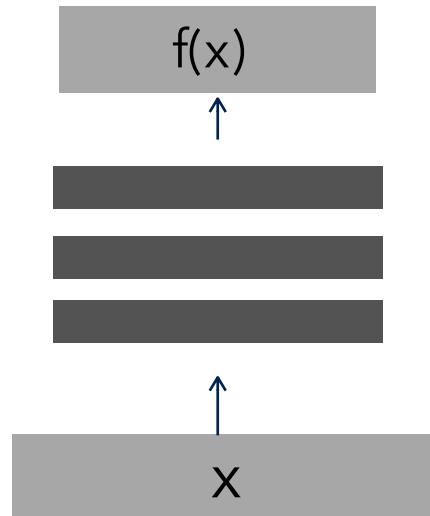
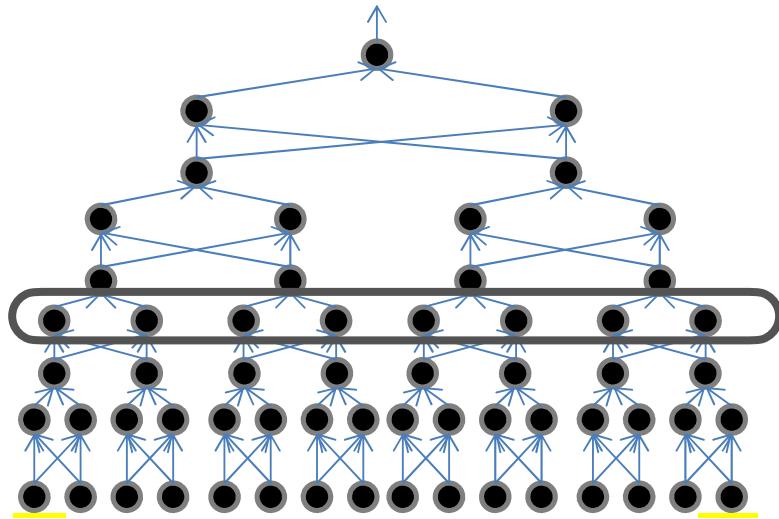
$$J(\Theta) = \frac{1}{n} \sum_{i=1}^n L(f(x^{(i)}), y^{(i)})$$

$$\frac{\partial J}{\partial \Theta_k^{(\ell)}} = \frac{1}{n} \sum_{i=1}^n \frac{\partial L(f(x^{(i)}), y^{(i)})}{\partial \Theta_k^{(\ell)}}$$

Cálculo de las Derivadas Parciales

- Notemos también que la salida de la red $f(x)$ para un determinado ejemplo genérico x , es simplemente la activación de la capa de salida $a^{(L)}$ después de pasar ese ejemplo por el grafo de computación, proceso que se denomina **forward pass**.

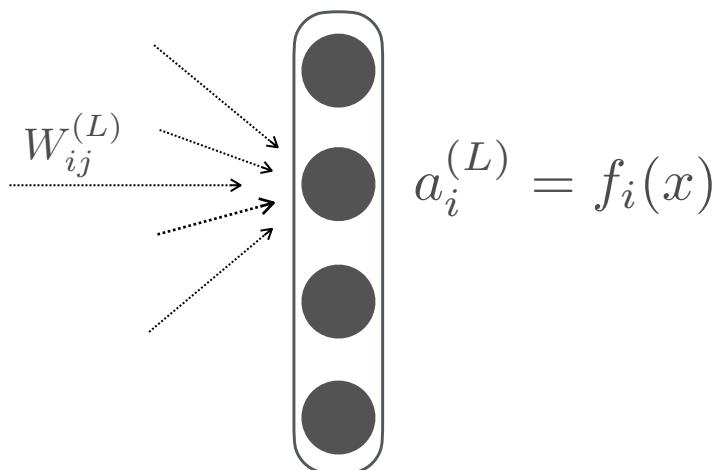
$\Theta^{(\ell)}$ parámetros de la capa ℓ



Feedback para la Salida

- Partamos entonces con el cálculo para la capa de salida. Supongamos que el parámetro que nos interesa es un cierto peso de conexión $\Theta_k^{(L)} = W_{ij}^{(L)}$ (es lo mismo para b)

$$f(x) = a^{(L)} = g(W^{(L)}a^{(L-1)} - b^{(L)})$$



$$\begin{aligned}\frac{\partial L(f(x), y)}{\partial W_{ij}^{(L)}} &= \frac{\partial L(f(x), y)}{\partial f_i(x)} \frac{\partial f_i(x)}{\partial W_{ij}^{(L)}} \\ &= \boxed{\frac{\partial L(f(x), y)}{\partial a_i^{(L)}}} \boxed{\frac{\partial a_i^{(L)}}{\partial W_{ij}^{(L)}}} \\ &\quad \boxed{2} \quad \boxed{1}\end{aligned}$$

Feedback para la Salida

- Ambas derivadas son fáciles de calcular. En el primer caso, $W_{ij}^{(L)}$ contribuye directamente a producir $f(x) = a^{(L)}$ (no hay problema de asignación de crédito). En el segundo caso, la loss L se aplica directamente a $f(x) = a^{(L)}$ (tampoco hay problema de asignación de crédito).

$$f(x) = a^{(L)} = g(W^{(L)}a^{(L-1)} - b^{(L)})$$

1

$$\frac{\partial a_i^{(L)}}{\partial W_{ij}^{(L)}} = g'(\cdot)a_j^{(L-1)}$$

* $g'() = g'(W^{(L)}a^{(L-1)} - b^{(L)})_i$



Feedback para la Salida

- Ambas derivadas son fáciles de calcular. En el primer caso, $W_{ij}^{(L)}$ contribuye directamente a producir $f(x) = a^{(L)}$ (no hay problema de asignación de crédito). En el segundo caso, la loss L se aplica directamente a $f(x) = a^{(L)}$ (tampoco hay problema de asignación de crédito).

$$f(x) = a^{(L)} = g(W^{(L)}a^{(L-1)} - b^{(L)})$$

2

$$\frac{\partial L(f(x), y)}{\partial a_i^{(L)}} = L'_i(f_i(x) - y_i)$$

Asumiendo

$$L(f(x), y) = \sum_i L_i(f_i(x) - y_i)$$



Feedback para la Salida

- Ambas derivadas son fáciles de calcular. En el primer caso, $W_{ij}^{(L)}$ contribuye directamente a producir $f(x) = a^{(L)}$ (no hay problema de asignación de crédito). En el segundo caso, la loss L se aplica directamente a $f(x) = a^{(L)}$ (tampoco hay problema de asignación de crédito).

$$f(x) = a^{(L)} = g(W^{(L)}a^{(L-1)} - b^{(L)})$$

2

$$\frac{\partial L(f(x), y)}{\partial a_i^{(L)}} = L'_i(f_i(x) - y_i)$$

Asumiendo

$$L(f(x), y) = \sum_i L_i(f_i(x) - y_i)$$

Ejemplo

$$\frac{\partial L(f(x), y)}{\partial a_i^{(L)}} = (f_i(x) - y_i)$$

En el caso

$$L(f(x), y) = 1/2 \sum_i (f_i(x) - y_i)^2$$



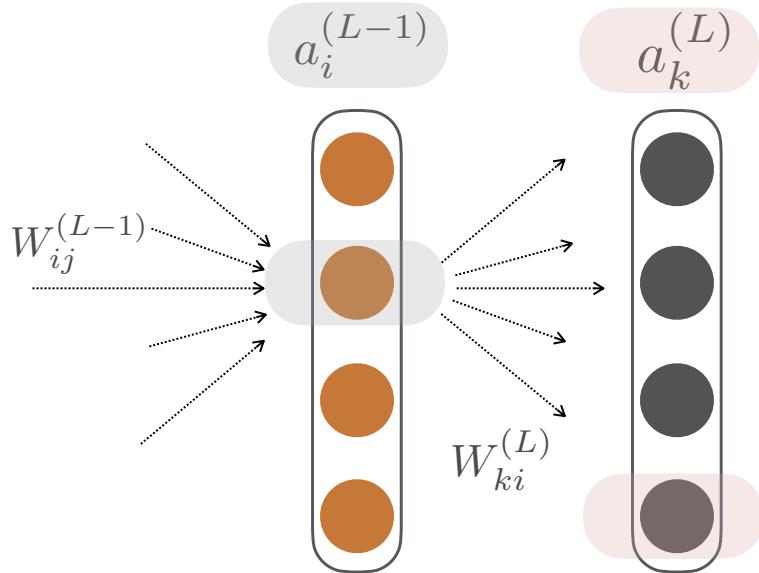
Feedback para la Última Capa Oculta

- Sigamos ahora con la última capa oculta. Supongamos s.p.d.g. que el parámetro que nos interesa es un cierto peso de conexión $\Theta_k^{(L-1)} = W_{ij}^{(L-1)}$.

$$f(x) = a^{(L)} = g(W^{(L)}a^{(L-1)} - b^{(L)})$$

$$a^{(L-1)} = g(W^{(L-1)}a^{(L-2)} - b^{(L-1)})$$

1



$$\frac{\partial L(f(x), y)}{\partial W_{ij}^{(L-1)}} = \frac{\partial L(f(x), y)}{\partial a_i^{(L-1)}} \boxed{\frac{\partial a_i^{(L-1)}}{\partial W_{ij}^{(L-1)}}}$$

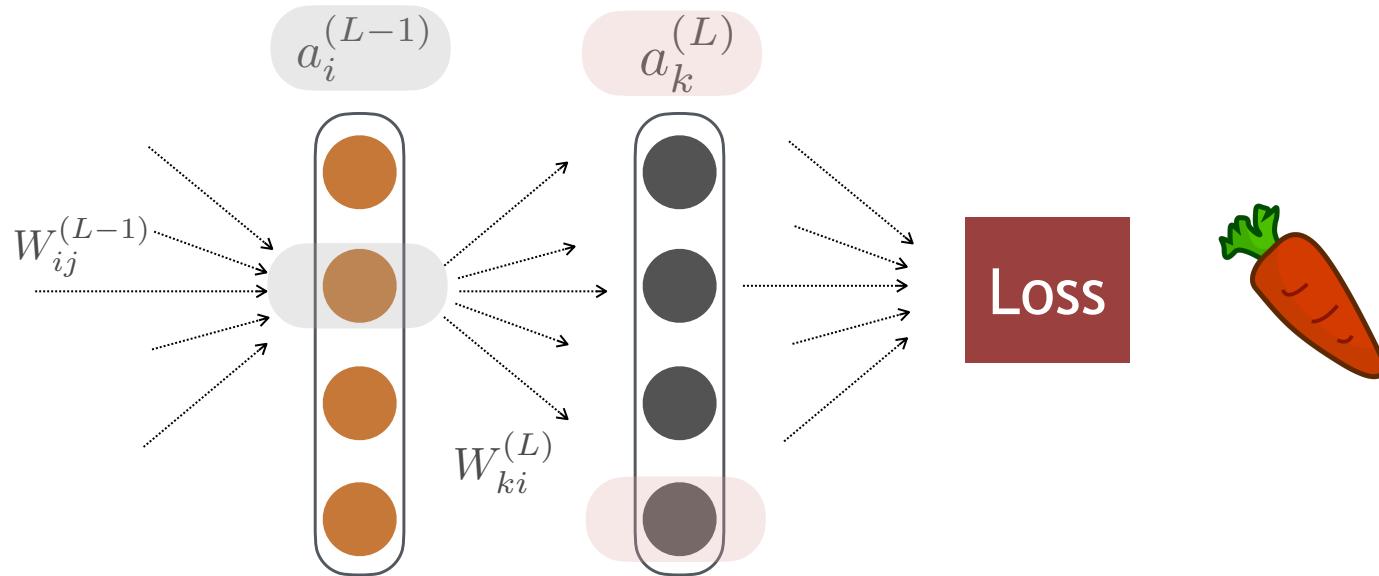
$$\frac{\partial L(f(x), y)}{\partial a_i^{(L-1)}} = \sum_k \boxed{\frac{\partial L(f(x), y)}{\partial a_k^{(L)}}} \boxed{\frac{\partial a_k^{(L)}}{\partial a_i^{(L-1)}}}$$

3

2

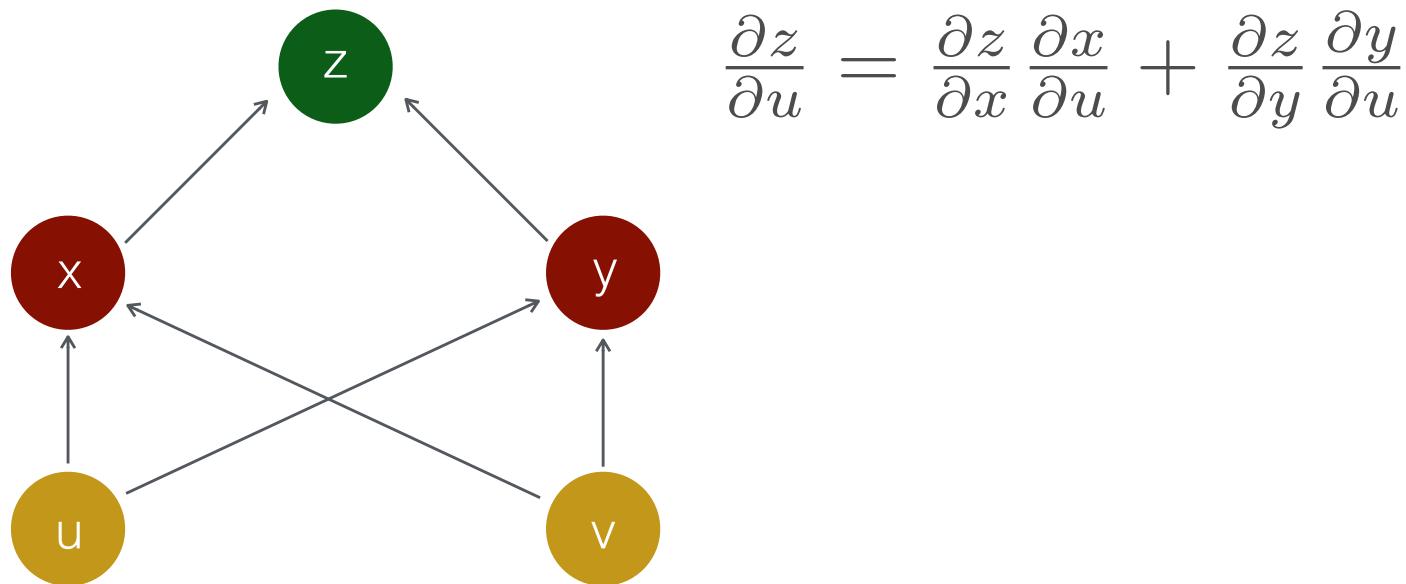
Feedback para la Última Capa Oculta

- El cálculo es más complicado porque $W_{ij}^{(L-1)}$ no contribuye directamente a producir $f(x) = a^{(L)}$. Contribuye a producir $a_i^{(L-1)}$ que participa en la activación de todas las neuronas de la capa de salida que luego se evalúan en la función objetivo.



Regla de la Cadena Multivariada

- Lo anterior es una expresión de la regla de la cadena en varias variables.



Feedback para la Última Capa Oculta

- Tenemos entonces que para la última capa oculta, el cálculo por hacer es

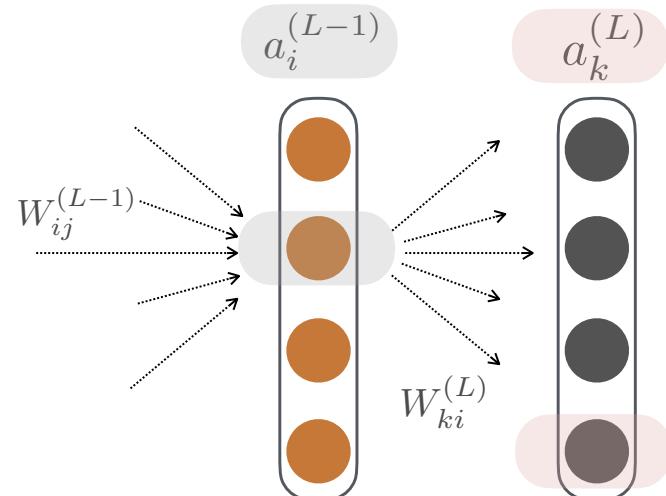
$$\frac{\partial L(f(x), y)}{\partial W_{ij}^{(L-1)}} = \frac{\partial L(f(x), y)}{\partial a_i^{(L-1)}} \boxed{\frac{\partial a_i^{(L-1)}}{\partial W_{ij}^{(L-1)}}} \quad 1$$

$$\frac{\partial L(f(x), y)}{\partial a_i^{(L-1)}} = \sum_k \frac{\partial L(f(x), y)}{\partial a_k^{(L)}} \boxed{\frac{\partial a_k^{(L)}}{\partial a_i^{(L-1)}}}$$

3

2

- La derivada (3) es exactamente la misma que antes.
- Las demás se pueden calcular como sigue ...



Feedback para la Última Capa Oculta

- Recordando que:

$$f(x) = a^{(L)} = g(W^{(L)}a^{(L-1)} - b^{(L)})$$

$$a^{(L-1)} = g(W^{(L-1)}a^{(L-2)} - b^{(L-1)})$$

2

$$\frac{\partial a_k^{(L)}}{\partial a_i^{(L-1)}} = g'(W^{(L)}a^{(L-1)} - b^{(L)})W_{ki}^{(L)}$$

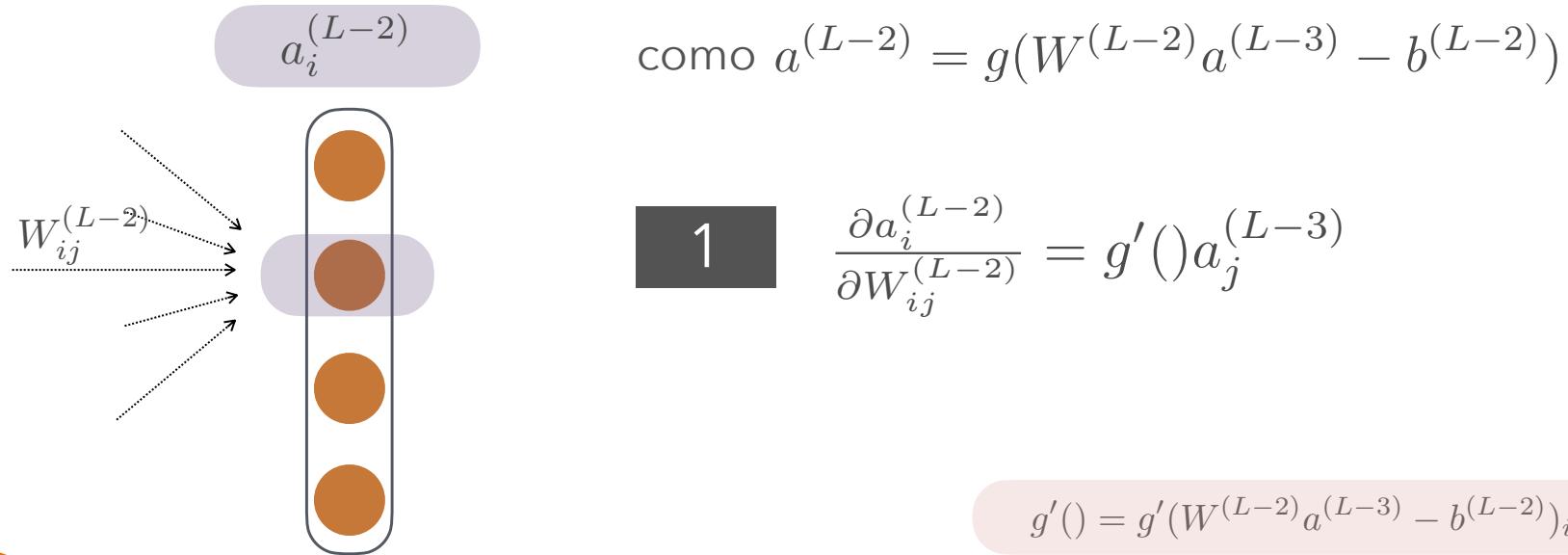
1

$$\frac{\partial a_i^{(L-1)}}{\partial W_{ij}^{(L-1)}} = g'(W^{(L-1)}a^{(L-2)} - b^{(L-1)})a_j^{(L-2)}$$

Penúltima Capa Oculta

- A medida que descendemos en profundidad, notaremos que el cálculo empieza a hacerse más complejo ...

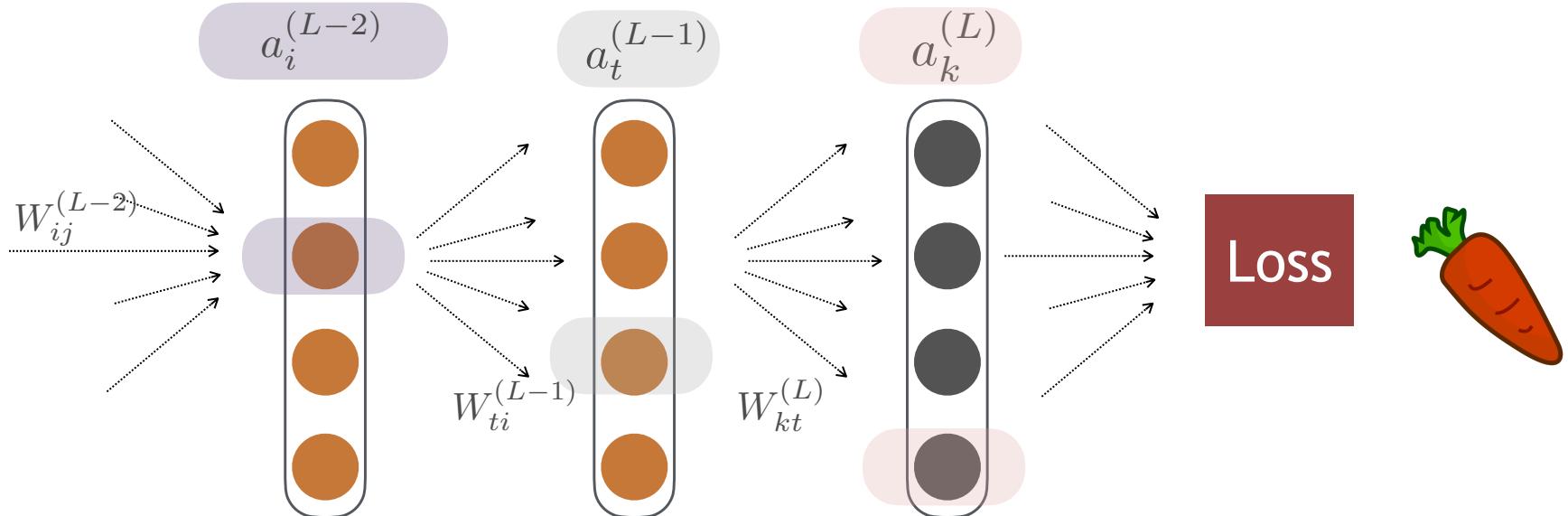
$$\frac{\partial L(f(x), y)}{\partial W_{ij}^{(L-2)}} = \frac{\partial L(f(x), y)}{\partial a_i^{(L-2)}} \frac{\partial a_i^{(L-2)}}{\partial W_{ij}^{(L-2)}}$$



Penúltima Capa Oculta

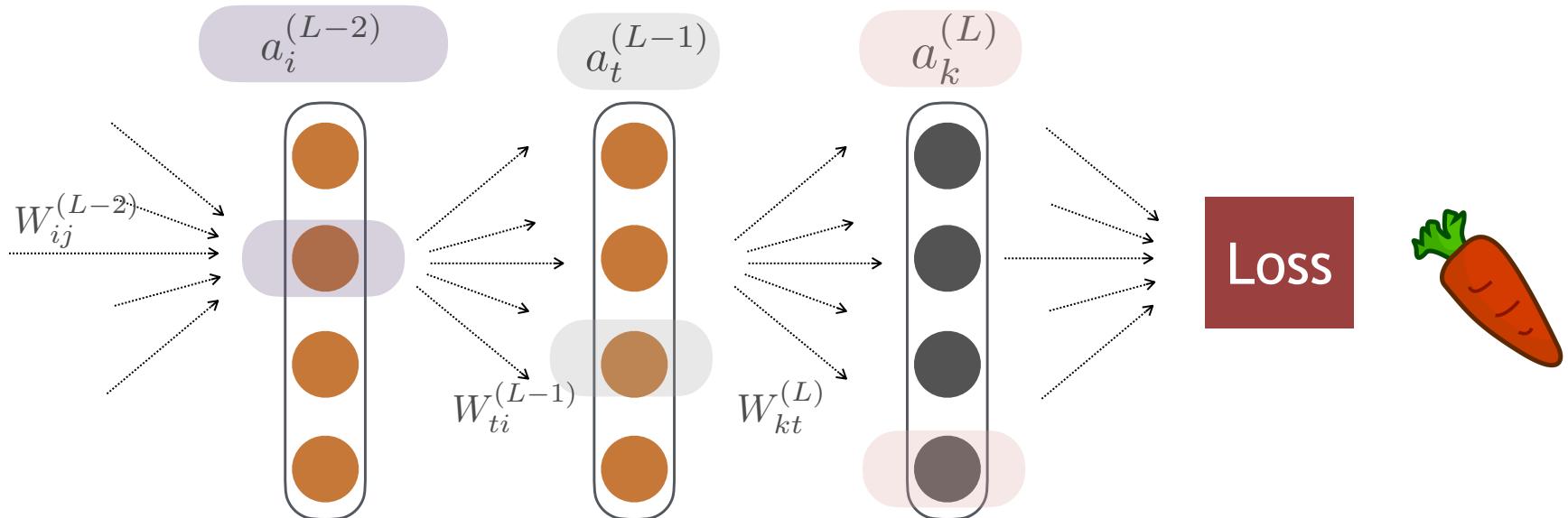
- Para calcular (2) , debemos considerar 3 niveles .

$$\frac{\partial L(f(x), y)}{\partial W_{ij}^{(L-2)}} = \frac{\partial L(f(x), y)}{\partial a_i^{(L-2)}} \frac{\partial a_i^{(L-2)}}{\partial W_{ij}^{(L-2)}}$$



Penúltima Capa Oculta

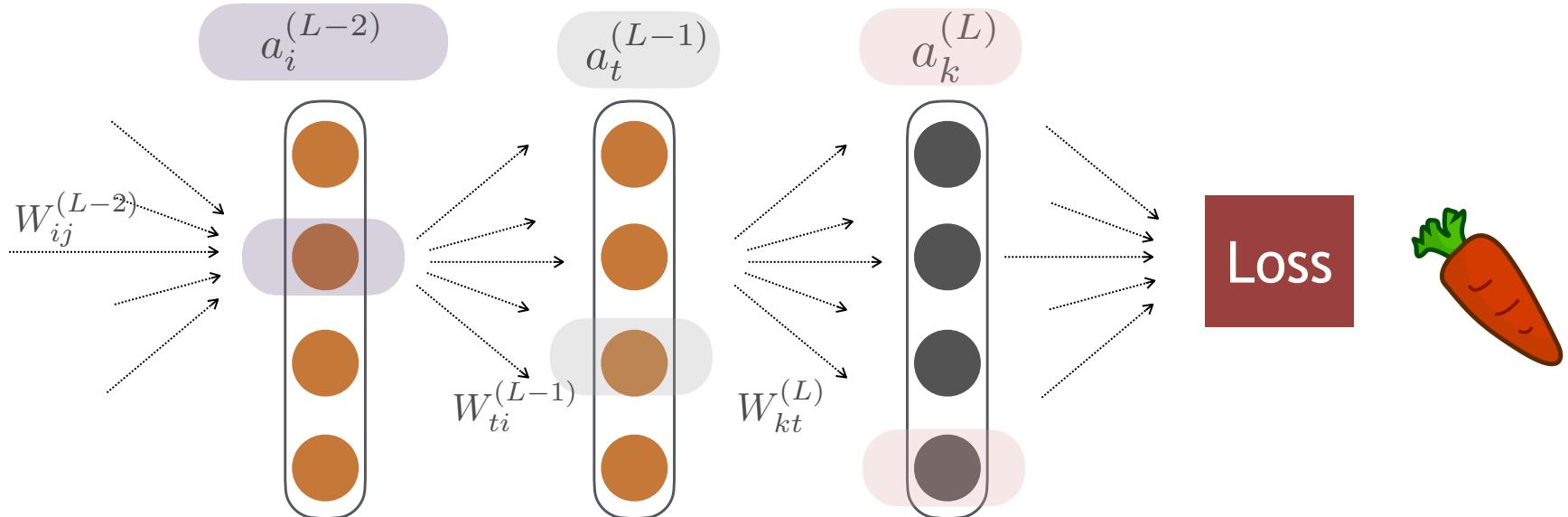
$$2 \quad \frac{\partial L(f(x), y)}{\partial a_i^{(L-2)}} = \sum_t \frac{\partial L(f(x), y)}{\partial a_t^{(L-1)}} \frac{\partial a_t^{(L-1)}}{\partial a_i^{(L-2)}}$$



Penúltima Capa Oculta

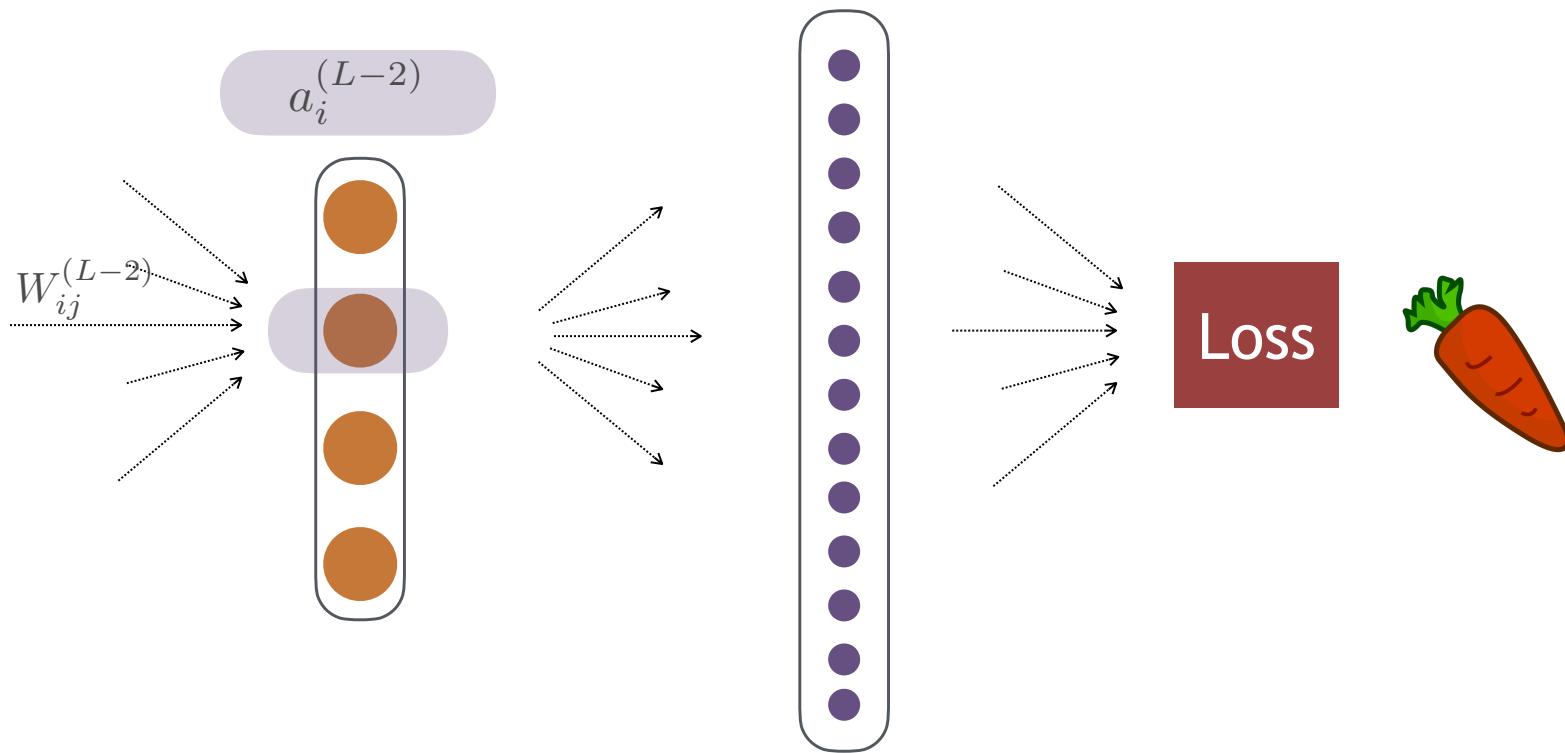
2

$$\begin{aligned}\frac{\partial L(f(x), y)}{\partial a_i^{(L-2)}} &= \sum_t \frac{\partial L(f(x), y)}{\partial a_t^{(L-1)}} \frac{\partial a_t^{(L-1)}}{\partial a_i^{(L-2)}} \\ &= \sum_t \frac{\partial a_t^{(L-1)}}{\partial a_i^{(L-2)}} \sum_k \frac{\partial L(f(x), y)}{\partial a_k^{(L)}} \frac{\partial a_k^{(L)}}{\partial a_t^{(L-1)}}\end{aligned}$$



Penúltima Capa Oculta

- Si cada capa de la red tiene M neuronas, un parámetro de la capa L-2 influye en el error de M^2 formas que corresponden a todos los caminos posibles en el grafo que parten desde la unidad i y llegan a la loss. Todo ellos debiesen ser considerados para obtener el crédito final.



Lema de Agregación

Consideremos un grado de computación con nodos $y(i), i \in I$ y arcos (dirigidos) $Z \subset I \times I$. Supongamos que entre el nodo a y el nodo b existe un conjunto de caminos no vacío P . Entonces,

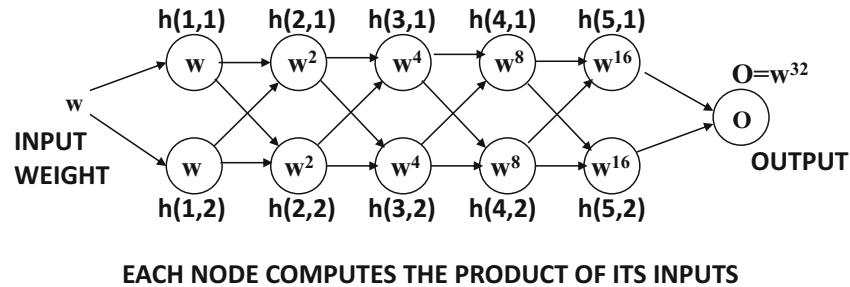
$$\frac{\partial o}{\partial a} = \sum_{p \in P} \prod_{(i,j) \in p} \frac{\partial y(j)}{\partial y(i)}$$

Ver por ejemplo: Charu C. Aggarwal - Neural Networks and Deep Learning. Springer (2018)



Lema de Agregación

- Ejemplo (los nodos muestran el resultado del cálculo en un grafo de computación donde los nodos multiplican sus inputs)



$$\begin{aligned}\frac{\partial o}{\partial w} &= \sum_{j_1, j_2, j_3, j_4, j_5 \in \{1, 2\}^5} \underbrace{\prod_{w} h(1, j_1)}_{w} \underbrace{\prod_{w^2} h(2, j_2)}_{w^2} \underbrace{\prod_{w^4} h(3, j_3)}_{w^4} \underbrace{\prod_{w^8} h(4, j_4)}_{w^8} \underbrace{\prod_{w^{16}} h(5, j_5)}_{w^{16}} \\ &= \sum_{\text{All 32 paths}} w^{31} = 32w^{31}\end{aligned}$$

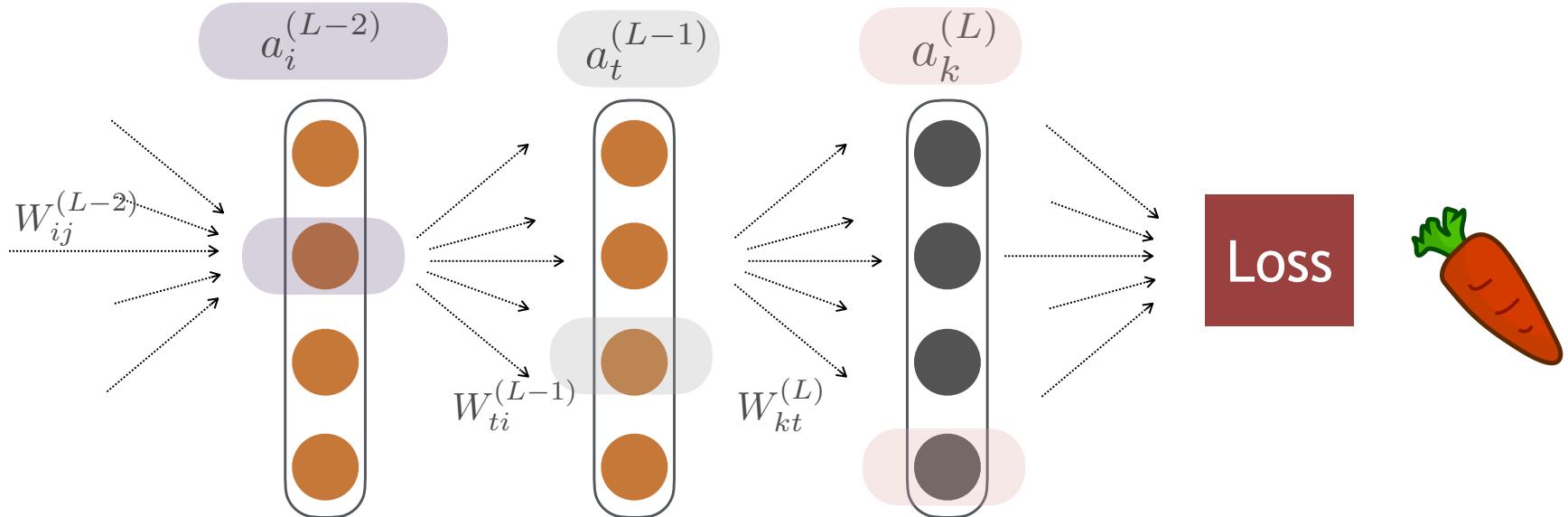
Ver por ejemplo: Charu C. Aggarwal - Neural Networks and Deep Learning. Springer (2018)



Lema de Agregación

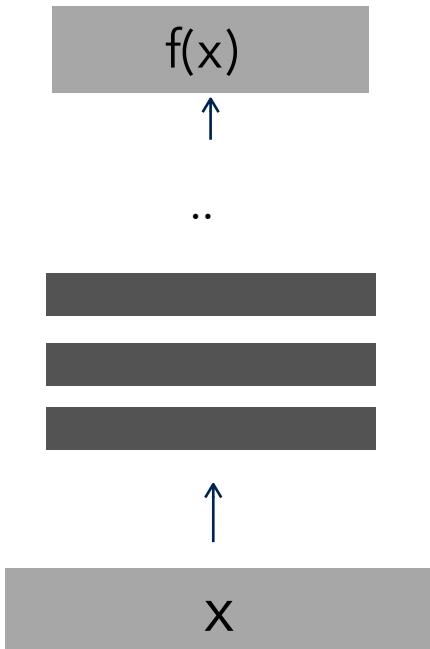
- Del lema, anterior, el cálculo de la derivada que teníamos involucra (como habíamos observado) 8 términos diferentes. Cada sumando tendría 3 términos que se multiplican.

$$2 \quad \frac{\partial L(f(x), y)}{\partial a_i^{(L-2)}}$$



Lema de Agregación

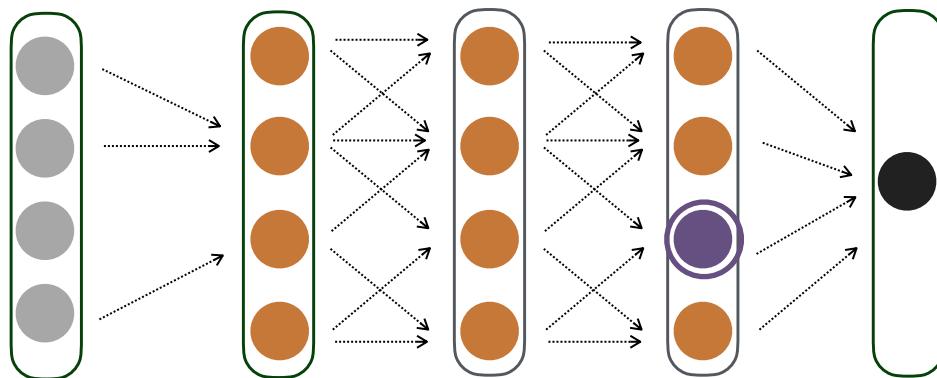
- Sin un buen algoritmo, la idea de las derivadas parciales puede ser **impracticable** si la red es muy profunda y densa.



- **En una red FF de L capas y M neuronas por cada, calcular 1 derivada parcial para la primera capa oculta podría costar $\mathcal{O}(M^L)$ operaciones aritméticas.**
- Cómo resolvemos el problema? Sólo hacia 1982-1986 empezó a ser claro como!

BackPropagation

- La idea clave del back-propagation es que si la profundidad nos permite reutilizar cálculos durante el **forward pass**, nos debe poder permitir también hacerlo durante el **backward pass** (proceso de cálculo de las derivadas parciales y ajuste de los pesos)



- Durante el forward pass, no re-calculemos las activaciones de los nodos de capas anteriores. Una vez realizados, esos cálculos se mantienen para permitir la recursión entre capas:

$$a^{(\ell)} = g(W^{(\ell)}a^{(\ell-1)} - b^{(\ell)})$$

BackPropagation

- Buscaremos generar una recursión para el cálculo de las derivadas parciales. En los cálculos preliminares que hicimos observamos este patrón:

$$\frac{\partial L(f(x), y)}{\partial W_{ij}^{(\ell)}} = \frac{\partial L(f(x), y)}{\partial a_i^{(\ell)}} \frac{\partial a_i^{(\ell)}}{\partial W_{ij}^{(\ell)}}$$

21

como $a^{(\ell)} = g(W^{(\ell)}a^{(\ell-1)} - b^{(\ell)})$

1 $\frac{\partial a_i^{(\ell)}}{\partial W_{ij}^{(\ell)}} = g'()a_j^{(\ell-1)}$

*

$$g'() = g'(W^{(\ell)}a^{(\ell-1)} - b^{(\ell)})_i$$



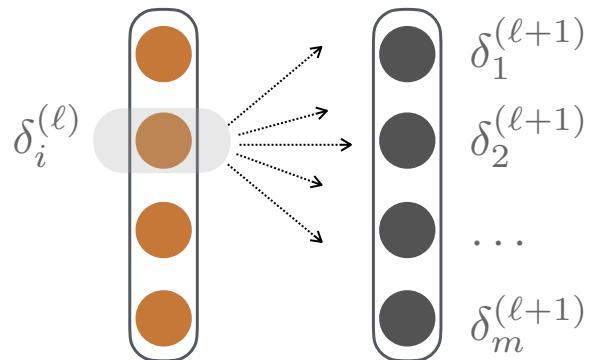
BackPropagation

- Diseñaremos la recursión para calcular la derivada (2), que denotaremos $\delta_i^{(\ell)}$

$$\frac{\partial L(f(x), y)}{\partial W_{ij}^{(\ell)}} = \boxed{\frac{\partial L(f(x), y)}{\partial a_i^{(\ell)}}} \frac{\partial a_i^{(\ell)}}{\partial W_{ij}^{(\ell)}}$$

2

- Asumiremos que cuando necesitamos hacer el cálculo en la capa ℓ , la capa sucesiva ha ya realizado el cálculo y está disponible en los nodos de las red (los cálculos para la capa de salida se deben hacer como mostramos al comienzo)



BackPropagation

- Por la regla de la cadena obtenemos,

$$\delta_i^{(\ell)} = \frac{\partial L(f(x), y)}{\partial a_i^{(\ell)}} = \sum_t \frac{\partial L(f(x), y)}{\partial a_t^{(\ell+1)}} \frac{\partial a_t^{(\ell+1)}}{\partial a_i^{(\ell)}} = \sum_t \delta_t^{(\ell+1)} \frac{\partial a_t^{(\ell+1)}}{\partial a_i^{(\ell)}}$$

- Como $a^{(\ell+1)} = g(W^{(\ell+1)}a^{(\ell)} - b^{(\ell+1)})$

$$\frac{\partial a_t^{(\ell+1)}}{\partial a_i^{(\ell)}} = g'()W_{ti}^{(\ell+1)}$$

- Y obtenemos la recursión que queríamos

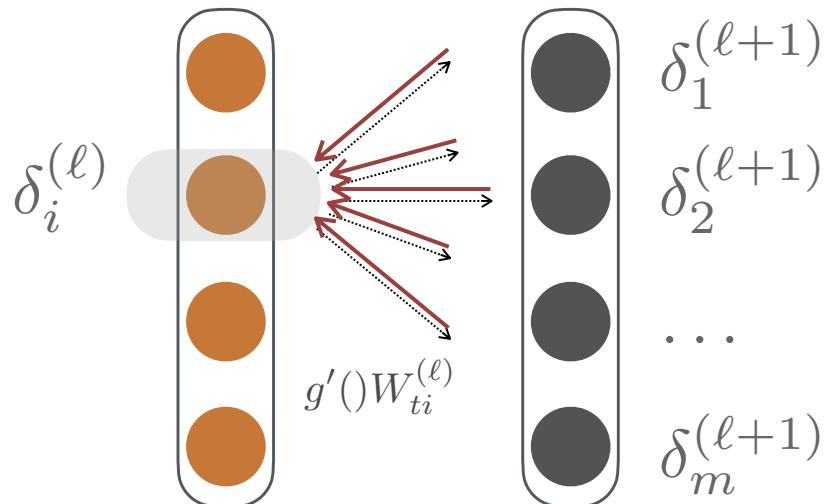
$$\delta_i^{(\ell)} = \sum_t \delta_t^{(\ell+1)} g'(p^{(\ell+1)}) W_{ti}^{(\ell+1)}$$

$$p^{(\ell+1)} = W^{(\ell+1)}a^{(\ell)} - b^{(\ell+1)}$$

BackPropagation

- El cálculo de la señal de error de cualquier capa, se puede obtener combinando linealmente las señales de error de la capa sucesiva. Esto es casi exactamente equivalente a invertir las conexiones y operar la red al revés:

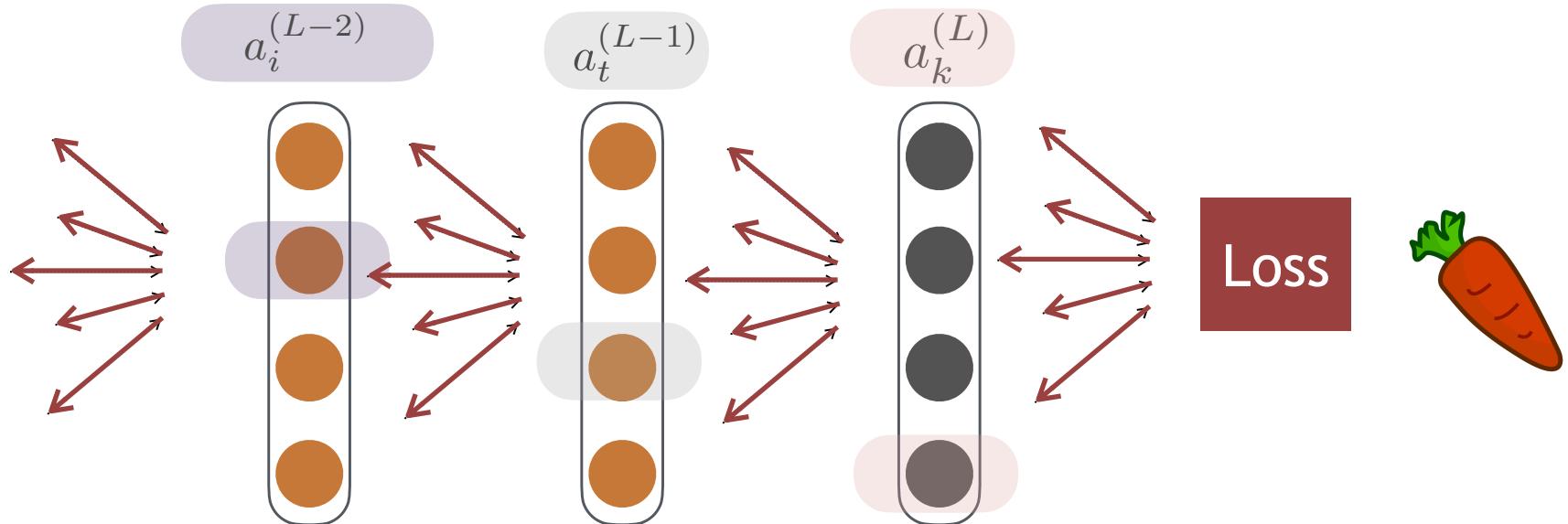
$$\delta_i^{(\ell)} = \sum_t \delta_t^{(\ell+1)} g'(p^{(\ell+1)}) W_{ti}^{(\ell+1)}$$



$$* p^{(\ell+1)} = W^{(\ell+1)} a^{(\ell)} - b^{(\ell+1)}$$

BackPropagation

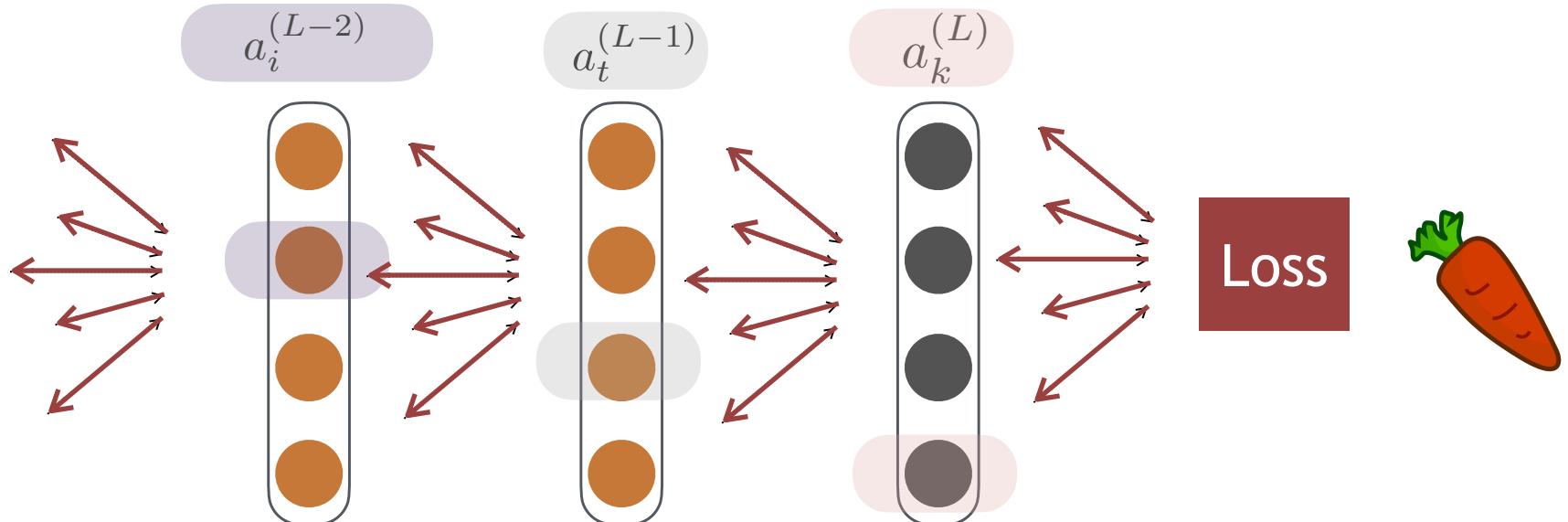
- Calculando las derivadas de la capa ℓ de esta forma, necesitamos hacer $\mathcal{O}(M)$ operaciones aritméticas en vez de $\mathcal{O}(M^\ell)$! Esta complejidad hace el procedimiento factible computacionalmente.



(Los cálculos para la capa de salida son la base de la recursión. Se hacen como mostramos al comienzo).

BackPropagation

- Además de eficiente, este procedimiento es intrínsecamente distribuido: las neuronas pueden operar en paralelo y requiere sólo información local (de sus vecinas).



Resumen de lo Obtenido

$$\delta_i^{(\ell)} = \sum_t \delta_t^{(\ell+1)} g'(p^{(\ell+1)}) W_{ti}^{(\ell+1)}$$

- **Esta recursión se inicializa en la capa de salida. Por ejemplo para la loss cuadrática tenemos**

$$\delta_i^{(L)} = f_i(x) - y_i$$

$$p^{(\ell+1)} = W^{(\ell+1)} a^{(\ell)} - b^{(\ell+1)}$$



Resumen de lo Obtenido

$$\delta_i^{(\ell)} = \sum_t \delta_t^{(\ell+1)} g'(p^{(\ell+1)}) W_{ti}^{(\ell+1)}$$

||

$$\frac{\partial L(f(x), y)}{\partial W_{ij}^{(\ell)}} = \boxed{\frac{\partial L(f(x), y)}{\partial a_i^{(\ell)}}} \boxed{\frac{\partial a_i^{(\ell)}}{\partial W_{ij}^{(\ell)}}}$$

||

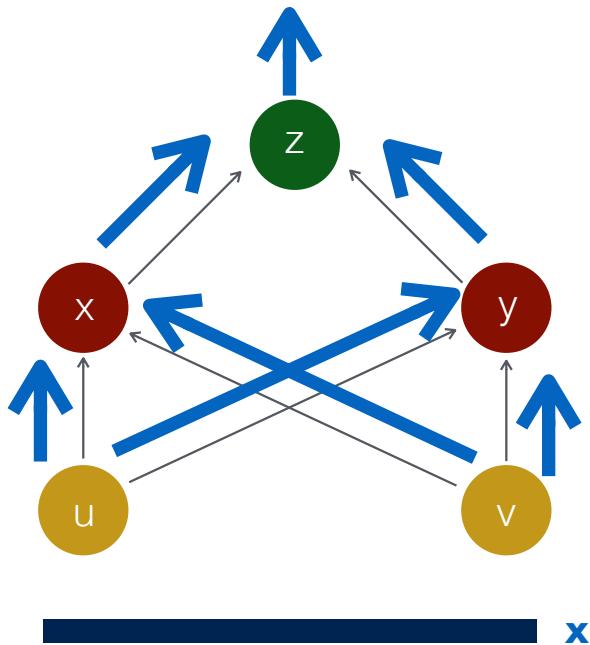
$$g'(p^{(\ell)}) a^{(\ell+1)}$$

$$p^{(\ell+1)} = W^{(\ell+1)} a^{(\ell)} - b^{(\ell+1)}$$

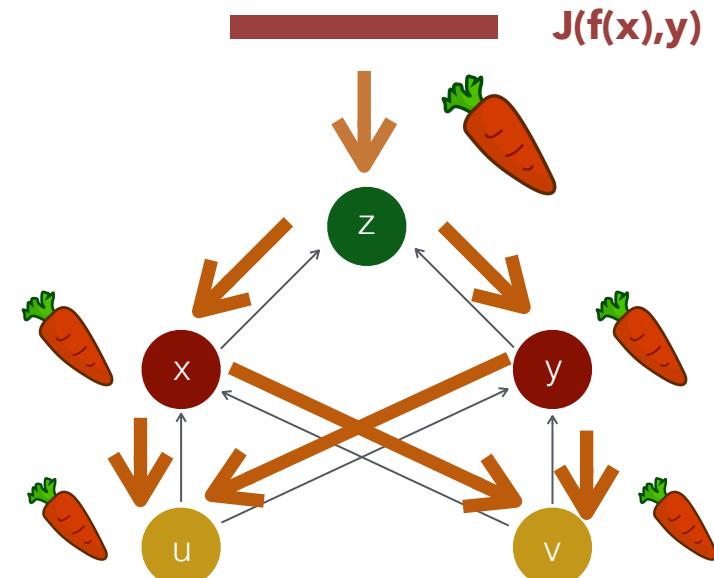


BackProp Estocástico

- Entonces, hemos descrito un procedimiento para calcular las derivadas de cada capa de la red considerando *1 ejemplo* de entrenamiento. Este proceso se denomina **backward pass** y puede sólo ser ejecutado luego de calcular la salida de la red para ese ejemplo (**forward pass**).



Forward Pass = Predict



Backward Pass = Correct

BackProp Estocástico

- En la mayoría de las aplicaciones, el objetivo de la red es una media sobre el conjunto de entrenamiento. Por ejemplo:

$$J(\Theta) = \frac{1}{n} \sum_{i=1}^n L(f(x^{(i)}), y^{(i)})$$

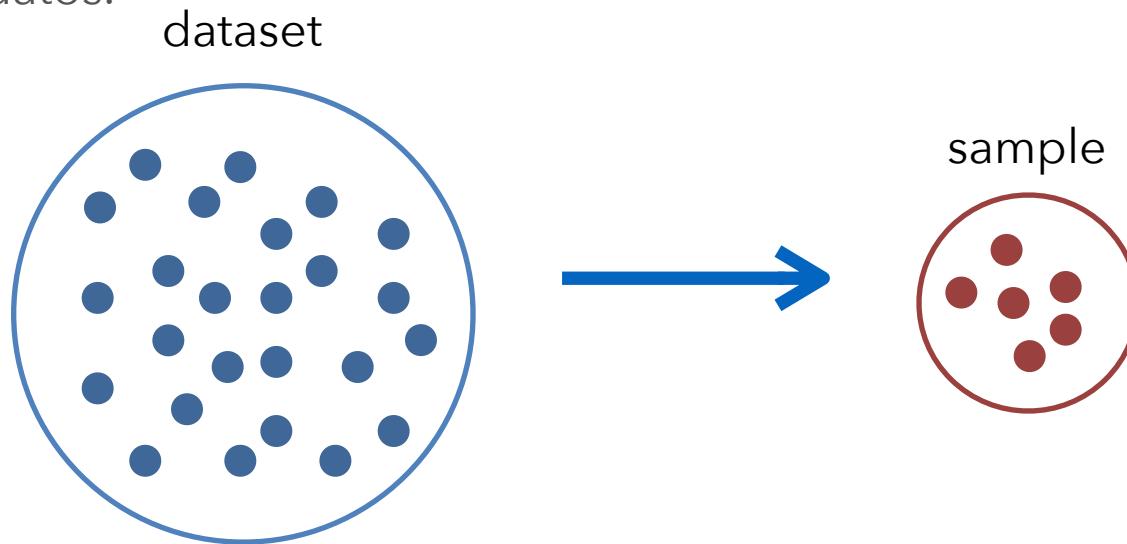
- Por lo tanto, el método ingenuo de entrenar la red consistiría en aplicar un forward pass y un backward pass por cada dato/ejemplo, sumar las derivadas parciales obtenidas y sólo después de esta agregación actualizar los pesos de la red.

$$\frac{\partial J}{\partial \Theta_k^{(\ell)}} = \frac{1}{n} \sum_{i=1}^n \frac{\partial L(f(x^{(i)}), y^{(i)})}{\partial \Theta_k^{(\ell)}}$$



BackProp Estocástico

- En la práctica, este procedimiento es ineficiente, sobre todo si el conjunto de ejemplos es grande. Por la **ley de los grandes números**, la media que necesitamos calcular puede **estimarse** usando un pequeño **lote o batch** de b datos.

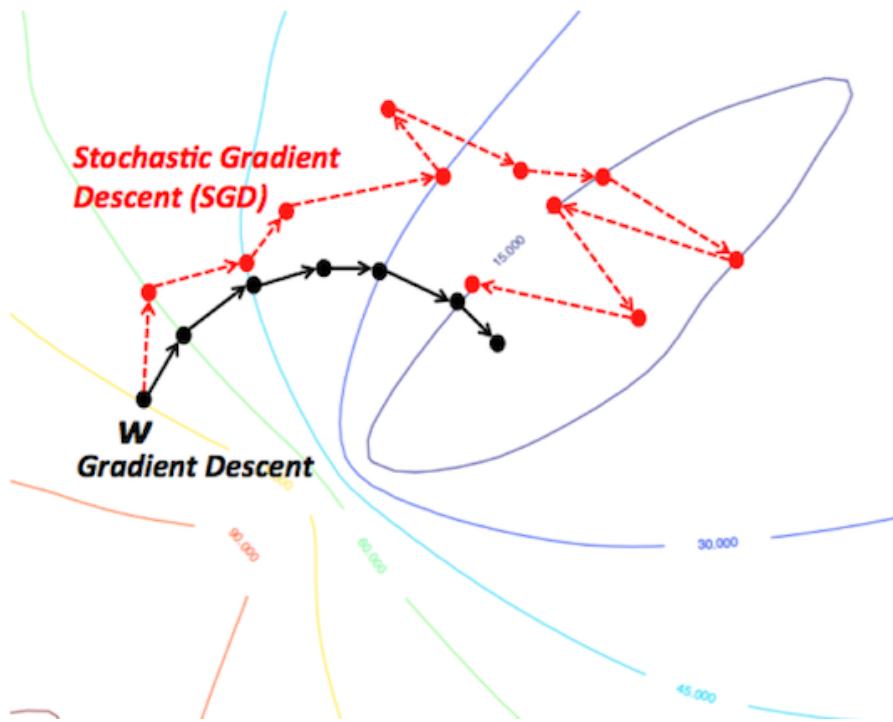


$$\frac{\partial J}{\partial \Theta_k^{(\ell)}} = \frac{1}{n} \sum_{i=1}^n \frac{\partial L(f(x^{(i)}), y^{(i)})}{\partial \Theta_k^{(\ell)}}$$

$$\frac{\partial J}{\partial \Theta_k^{(\ell)}} \approx \frac{1}{b} \sum_{i^*=1}^b \frac{\partial L(f(x^{(i^*)}), y^{(i^*)})}{\partial \Theta_k^{(\ell)}}$$

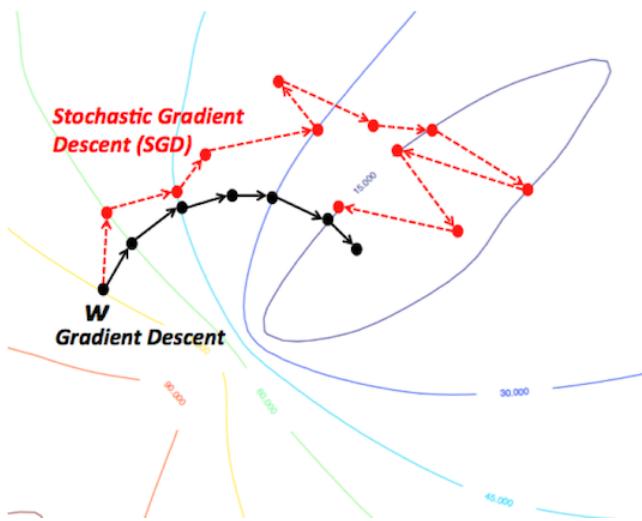
BackProp Estocástico

- Usando este procedimiento, obtenemos sólo una aproximación de gradiente verdadero, lo que los pasos de optimización sean menos precisos (en ocasiones incluso pueden empeorar el valor del objetivo)



BackProp Estocástico

- Sin embargo, en un determinado tiempo de computación, podemos hacer muchísimos más pasos de optimización. Si calcular una derivada sobre 1 ejemplo nos cuesta $\mathcal{O}(c)$ y asumimos que hacer sobre m ejemplos cuesta $\mathcal{O}(mc)$, la versión estocástica propuesta puede hacer $\mathcal{O}(m/b)$ pasos de optimización en el tiempo en que el algoritmo no estocástico hace 1.



Ejemplo:

$$m=1.000.000, b=100$$

¿conviene hacer 1 paso exacto
ó 10.000 aproximados?

BackProp Estocástico

- Operar con pequeños lotes o mini-batches en vez de todo el conjunto suele ser mejor en la práctica. Parte de las razones por las que esto sucede es que el objetivo de entrenamiento es sólo una aproximación del objetivo real.

$$J(\Theta) = \frac{1}{n} \sum_{i=1}^n L(f(x^{(i)}), y^{(i)}) \quad \approx \quad \mathbb{E} (L(f(x), y))$$

Error de
Entrenamiento

Error de Predicción
(novel data)

- Optimizar demasiado para una aproximación de objetivo real puede resultar en un peor error de predicción en la práctica.

BackProp Estocástico

- Esta idea ha sido revisada múltiples veces en la literatura.

The Tradeoffs of Large Scale Learning

Léon Bottou
NEC laboratories of America
Princeton, NJ 08540, USA
leon@bottou.org

Olivier Bousquet
Google Zürich
8002 Zurich, Switzerland
olivier.bousquet@m4x.org



Leon Bottou

Train faster, generalize better:
Stability of stochastic gradient descent

Moritz Hardt*

Benjamin Recht[†]

Yoram Singer[†]

February 9, 2016

Abstract

We show that parametric models trained by a stochastic gradient method (SGM) with few iterations have vanishing generalization error. We prove our results by arguing that SGM is algorithmically stable in the sense of Bousquet and Elisseeff. Our analysis only employs elementary tools from convex and continuous optimization. We derive stability bounds for both convex and non-convex optimization under standard Lipschitz and smoothness assumptions.

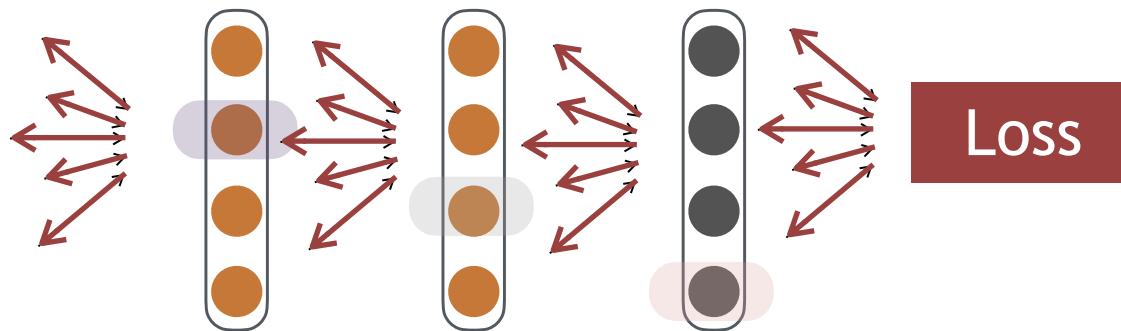


Moritz Hardt



Vectorización

- La versión estocástica de Backprop requiere ejecutar b veces un forward-pass, b veces un back-ward pass, agregar (mediar) los resultados y finalmente corregir todos los parámetros entrenables.
- Naturalmente, estas operaciones se pueden hacer localmente y con sincronizaciones sólo debidas a la naturaleza recursiva del algoritmo.
- Aún así, en la práctica no resulta conveniente ejecutar el forward pass y el backward pass para 1 ejemplo a la vez. **Ambas operaciones se pueden vectorizar** y aplicar a todo el **lote o batch** de ejemplos simultáneamente



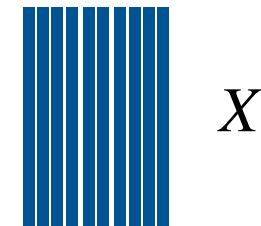
Vectorización

- Por ejemplo, el forward pass, puede escribirse considerando una matriz de ejemplos X (cada columna es un ejemplo):

$$f(X) = A^{(L)}$$

$$A^{(\ell)} = g(W^{(\ell)} A^{(\ell-1)} - B^{(\ell)}) \quad \forall \ell = 1, 2, \dots, L$$

$$A^{(0)} = X$$



- Aquellos que se sientan incómodos usando las columnas como índices de los ejemplos (en ML usamos casi siempre las filas) pueden preferir la siguiente versión:

$$f(X) = A^{(L)}$$

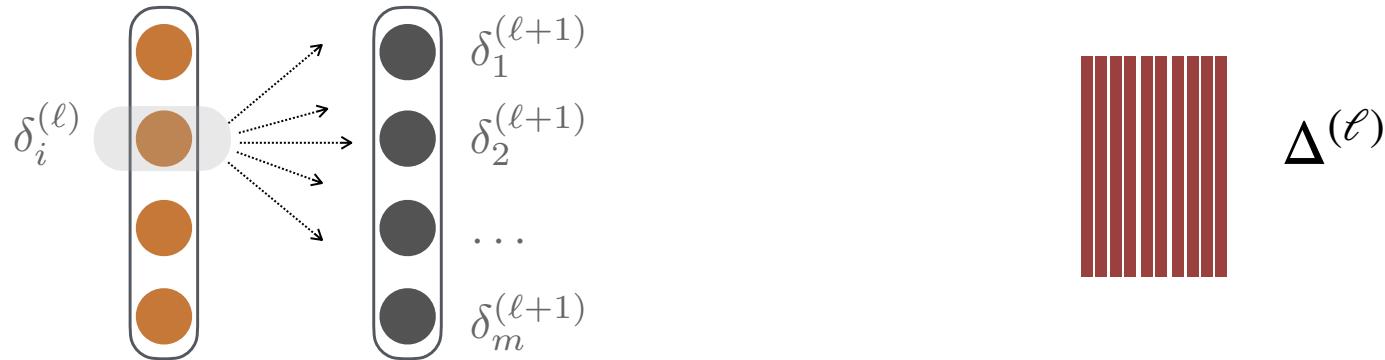
$$A^{(\ell)} = g(A^{(\ell-1)} W^{(\ell)T} - B^{(\ell)T}) \quad \forall \ell \in [L]$$

$$A^{(0)} = X$$



Vectorización

- Una forma de vectorizar el backward pass es organizar las señales de error también en una matriz



$$\delta_i^{(\ell)} = \sum_t \delta_t^{(\ell+1)} g'() W_{ti}^{(\ell)}$$

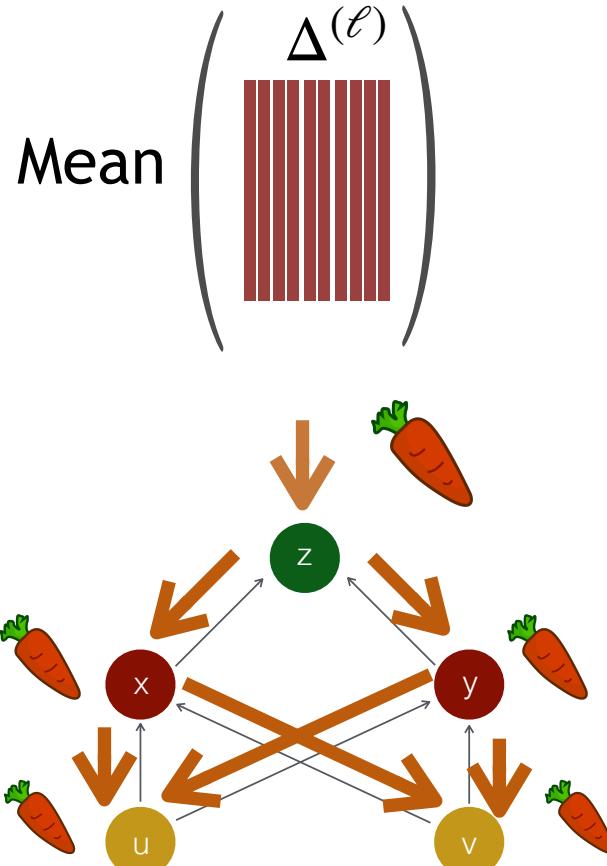
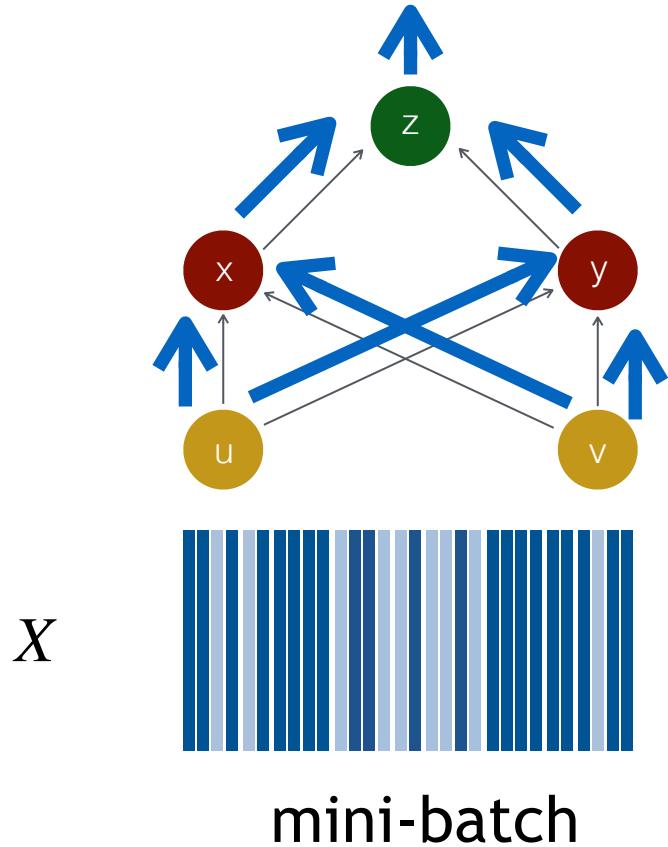
$$\Delta^{(\ell)} = W^{(\ell+1)T} G' \Delta^{(\ell+1)}$$

$$G' = \text{diag}(g'(W^{(\ell+1)} a^{(\ell)} - b^{(\ell+1)}))$$

* $g'() = g'(W^{(\ell+1)} a^{(\ell)} - b^{(\ell+1)})_t$

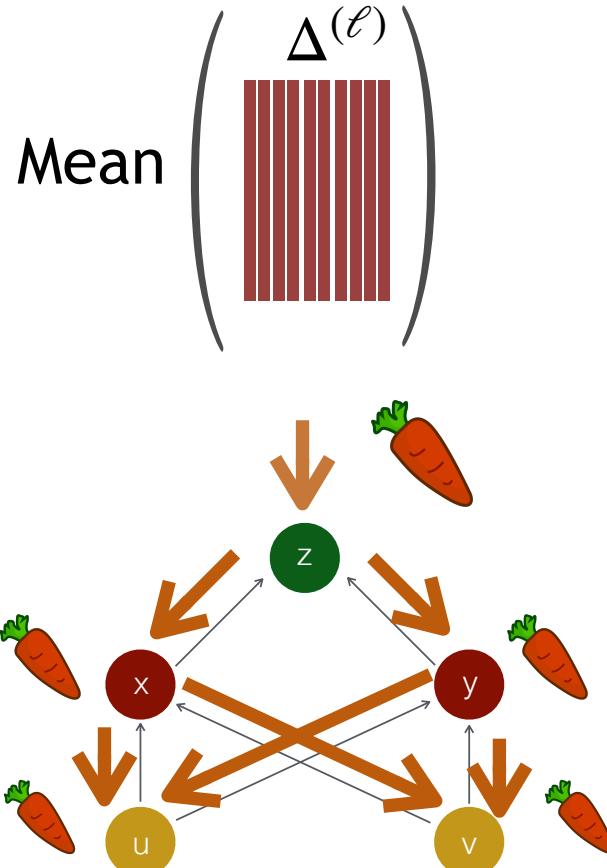
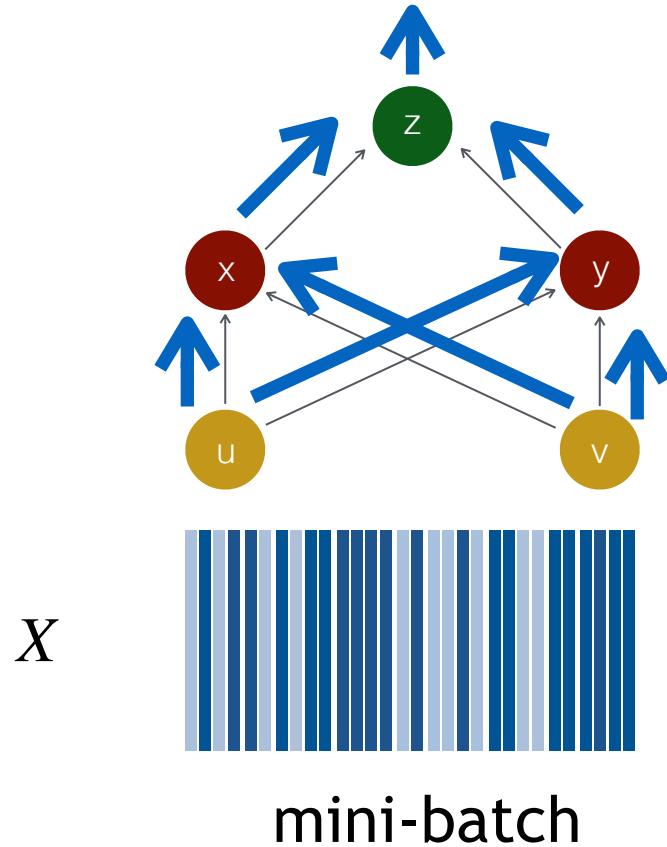
Selección de los Mini-Batches

- La versión vectorizada y estocástica de BP operaría como sigue:



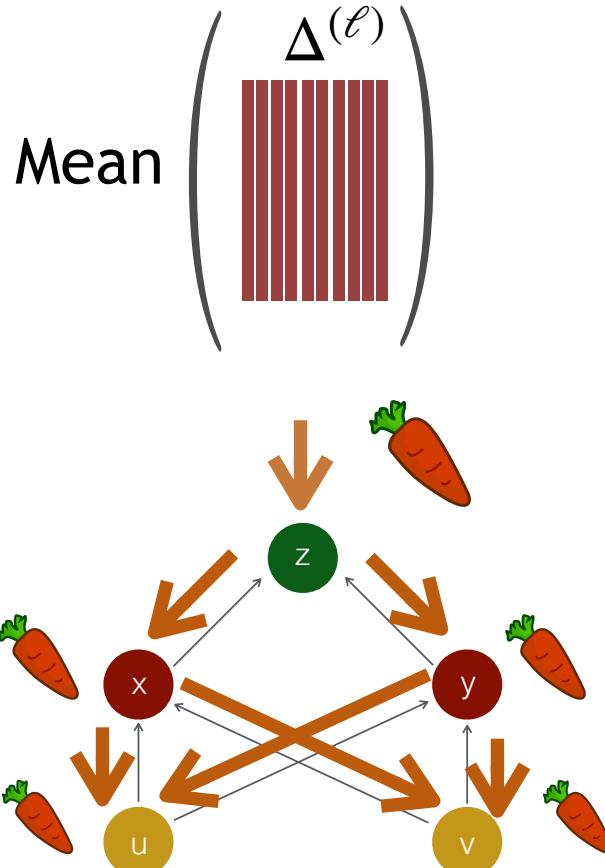
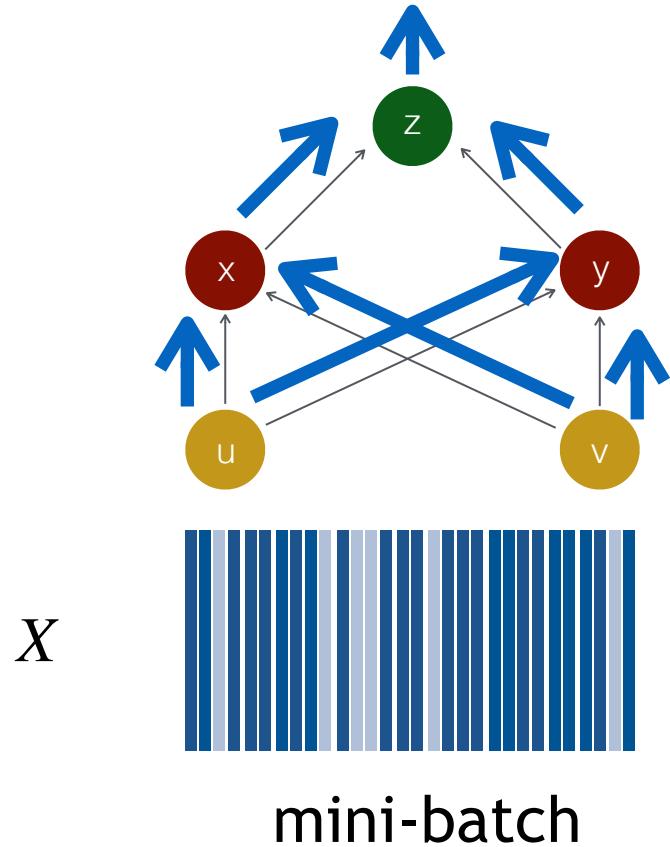
Selección de los Mini-Batches

- La versión vectorizada y estocástica de BP operaría como sigue:



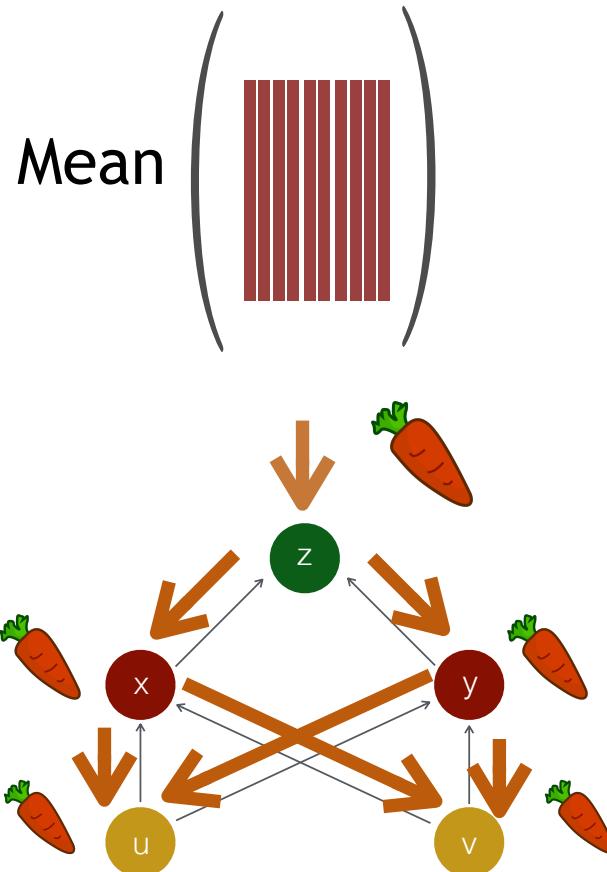
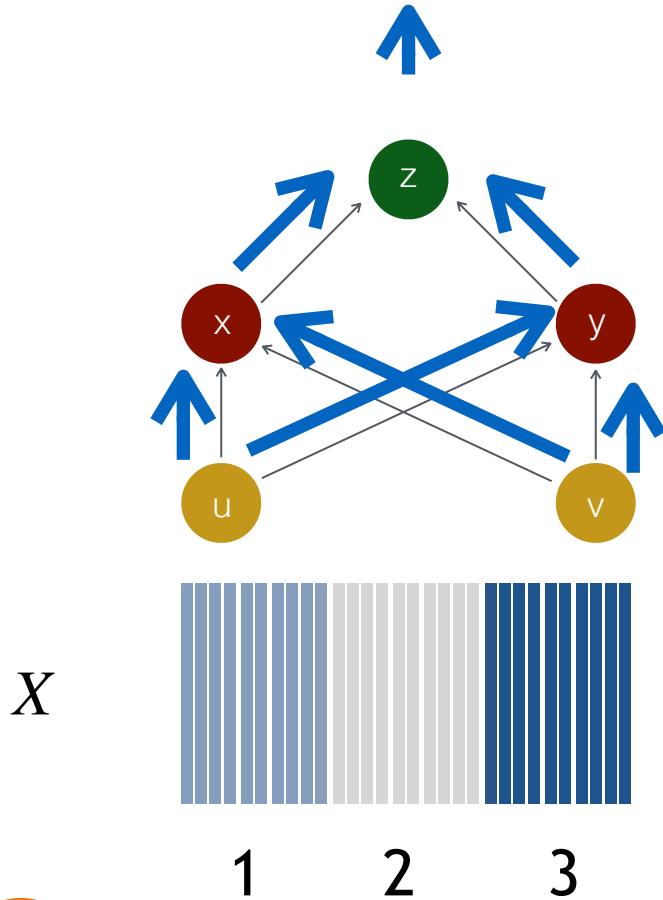
Selección de los Mini-Batches

- La versión vectorizada y estocástica de BP operaría como sigue:



Selección de los Mini-Batches

- En la práctica la selección de los mini-batches se hace permutando los datos en cada época de entrenamiento y luego seleccionando en modo ordenado.



Diferenciación Automática

- Dada la importancia de las derivadas en el entrenamiento de una red, mayoría de los lenguajes/entornos de programación modernos para implementar estos modelos incorporan herramientas que permiten obtener esas derivadas de modo completamente automático.
- Estas herramientas no se basan, como se podría esperar, en diferencias finitas

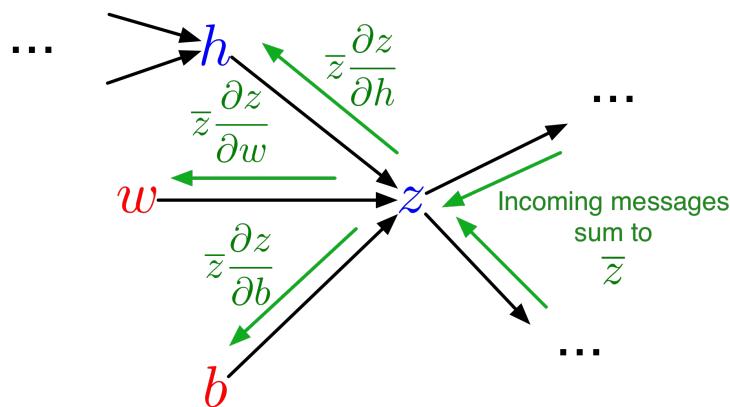
$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x + \Delta h) - f(x)}{\Delta h}$$

- Los métodos computacionalmente eficientes de ese tipo suelen ser inestables y tener grandes errores de aproximación que se propagan en la red. los métodos más sofisticados (orden superior) suelen ser costosos computacionalmente.



Diferenciación Automática

- Los sistemas modernos (e.g. autograd) se basan en el concepto de grafo de computación y calculan las derivadas usando back-propagation (o variantes de este procedimiento) sobre el grafo, exactamente como hemos hecho con nuestra red.
- El grafo de computación que corresponde a una función contiene sólo operaciones primitivas del lenguaje cuyas derivadas se sabe calcular. Las derivadas de la función se construyen usando la regla de la cadena.



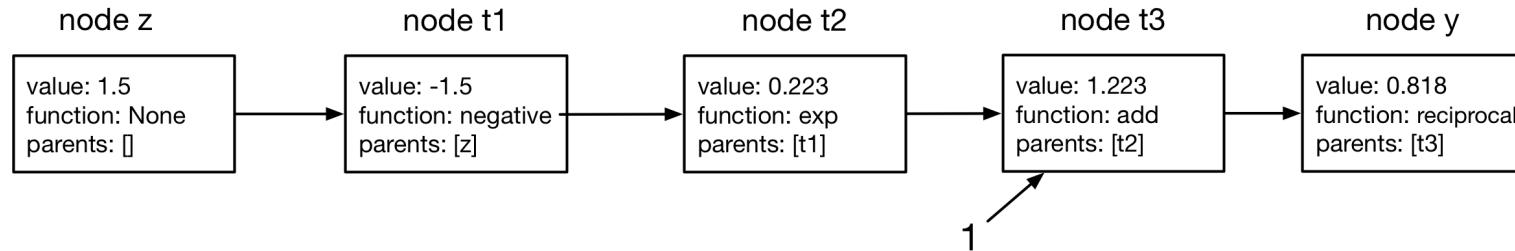
Diferenciación Automática

- Ejemplo:

```
def logistic(z):
    return 1. / (1. + np.exp(-z))

# that is equivalent to:
def logistic2(z):
    return np.reciprocal(np.add(1, np.exp(np.negative(z)))))

z = 1.5
y = logistic(z)
```



<https://github.com/HIPS/autograd#end-to-end-examples>

Entonces ...

- Backpropagation es un método para implementar de modo computacionalmente eficiente la idea de asignar crédito/feedback a las unidades del grafo usando derivadas parciales. El método reduce la complejidad de exponencial a lineal en la profundidad.
- La idea es calcular las señales de error para la capa de salida y proceder luego de manera recursiva mediante un procedimiento denominado backward pass que es casi equivalente a operar la red invirtiendo la dirección de los arcos que se usó durante el forward pass.
- En la práctica, se opera usando pequeños lotes de ejemplos y explotando la posibilidad de vectorizar los cálculos.
- Tarea: Estudie experimentalmente los efectos de tamaño de mini-batch tanto sobre la eficiencia como sobre el error de predicción de la red.

