Descenso de Gradiente Estocástico

Motivación: Coste computacional de Descenso de Grad.

- Elementos
  - N ejemplos
  - P parámetros
  - I iteraciones
- En cada iteración
  - **Cálculo de $\delta E/\delta w_i$**, i=1,..,P
    - Asumo *mejor caso*
      - Lineal en N y P: **O(NP)**
  - **Actualización de w_i**, i=1,..,M
    - Lineal en la P: **O(P)**
- **Coste total**
  - O(INP)+O(IP)
    - **O(INP)**

Iteración 1

Horas	Nota
2	1
5	3.2
7	4.5
9	6
10	4
11	4.5
13.4	5.5
14	3
15	5

$$w_1=w_1-\delta E/\delta w_1$$
$$w_2=w_2-\delta E/\delta w_2$$
$$\dots$$
$$w_p=w_p-\delta E/\delta w_p$$

Iteración 2

Horas	Nota
2	1
5	3.2
7	4.5
9	6
10	4
11	4.5
13.4	5.5
14	3
15	5

$$w_1=w_1-\delta E/\delta w_1$$
$$w_2=w_2-\delta E/\delta w_2$$
$$\dots$$
$$w_p=w_p-\delta E/\delta w_p$$

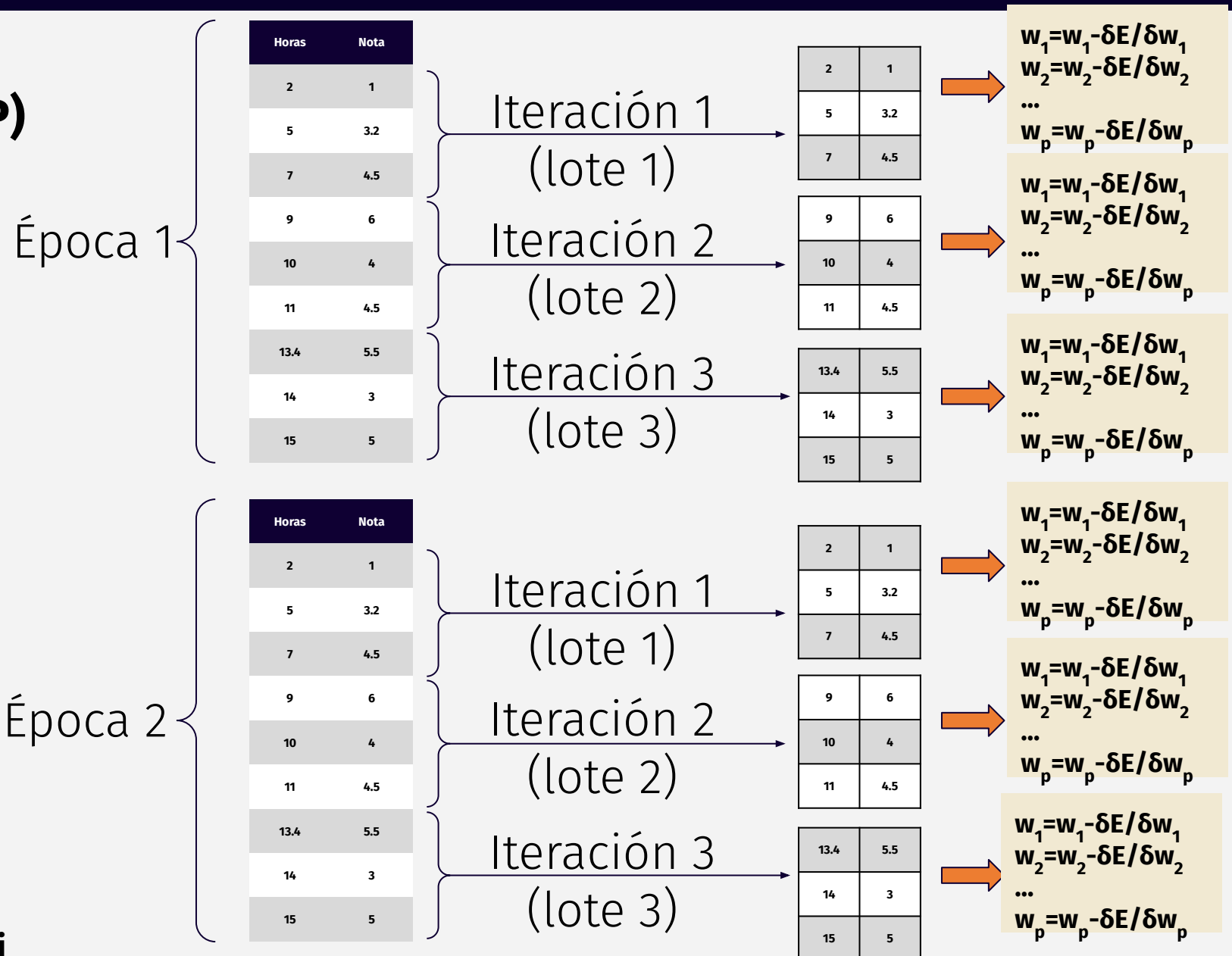
Motivación: Coste computacional de Descenso de Grad.

- Elementos
  - N ejemplos
  - P parámetros
  - I iteraciones
- En cada iteración
  - **Cálculo de $\delta E/\delta w_i$**, i=1,..,P
    - Asumo *mejor caso*
      - Lineal en N y P: **O(NP)**
  - **Actualización de w_i**, i=1,..,M
    - Lineal en la P: **O(P)**
- **Coste total**
  - O(INP)+O(IP)
    - **O(INP)**

```
def descenso_gradiente(E,w,x,y)
    w = …
    P = len(w)
    converge=False
    while not converge:
        for i in range(P):
            δEδW[i] = derivada(E,i,x,y)
        for i in range(P):
            w[i] =  w[i] - δEδW[i]
        converge = … (depende)
    return w
```

Entrenamiento por lotes

- Coste total **O(INP)**
  - Cálculo de derivadas **O(NP)**
- **Idea**
  - Dividir datos en **lotes**
    - de **B << N** ejemplos
  - Calcular derivadas con **lotes**
    - Gradiente aproximado
    - Ruidoso
      - Pero funciona
    - Cálculo de derivadas
      - **O(BP)**
- Ejemplo con **B=3**
  - Épocas = recorridos del conjunto de datos
  - Iteraciones = act. de los $\mathbf{w_i}$



Descenso de gradiente Clásico vs Por lotes

- Asumimos
  - model.w = matriz con todos los parámetros de la red
  - δw = matriz con la derivada de respecto a model.w
- Descenso de gradiente Clásico

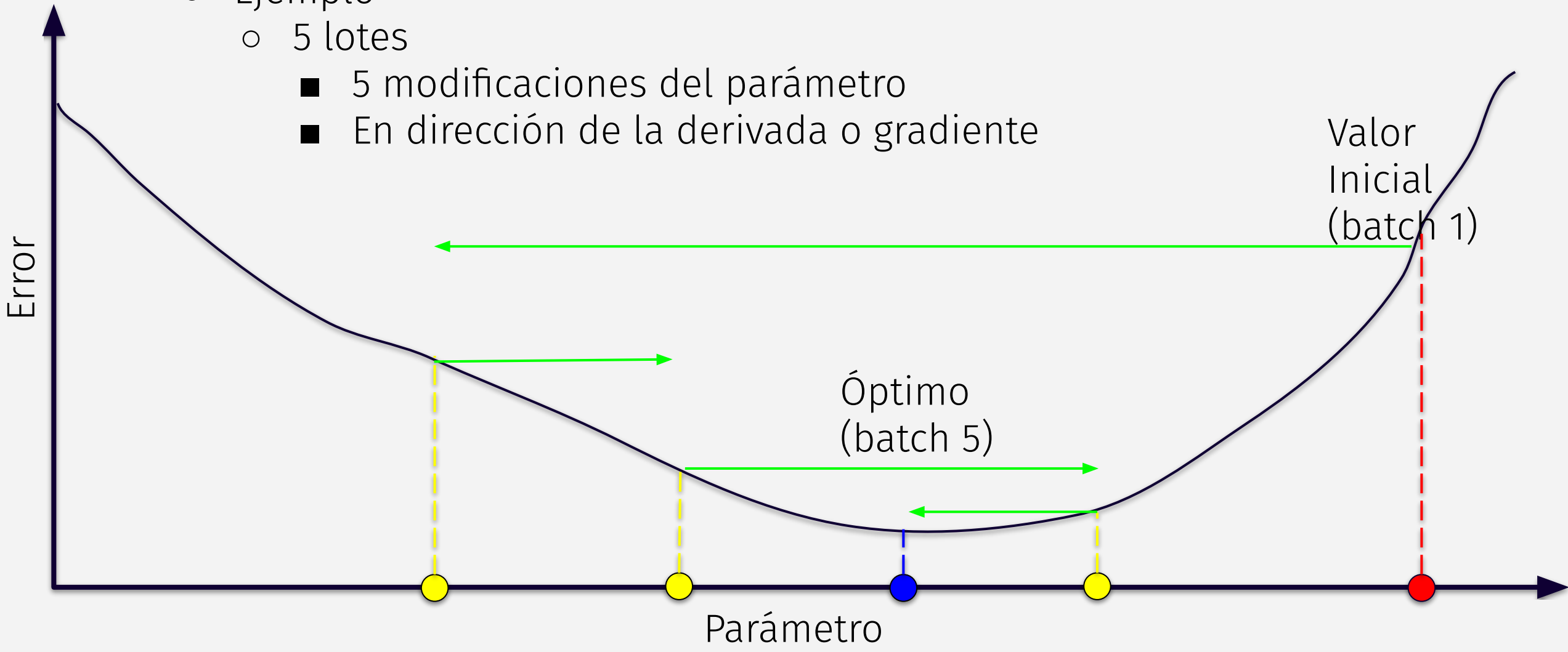
```
x,y = load_data()

lr = 0.001 # learning rate
model = ….
epochs = 1000

for i in range(epochs):
  δw = model.derivatives(x,y)
  model.w = model.w - lr * δw
```

- Descenso de gradiente por lotes

```
x,y = load_data()

lr = 0.001 # learning rate
model = ….
epochs = 1000

n = x.shape[0]
batch_size = 32
batches = n // batch_size

for i in range(epochs):
  for batch in range(batches):
    batch_x,batch_y = get_batch(x,y,batch)
    δw = model.derivatives(batch_x,batch_y)
    model.w = model.w - lr * δw
```

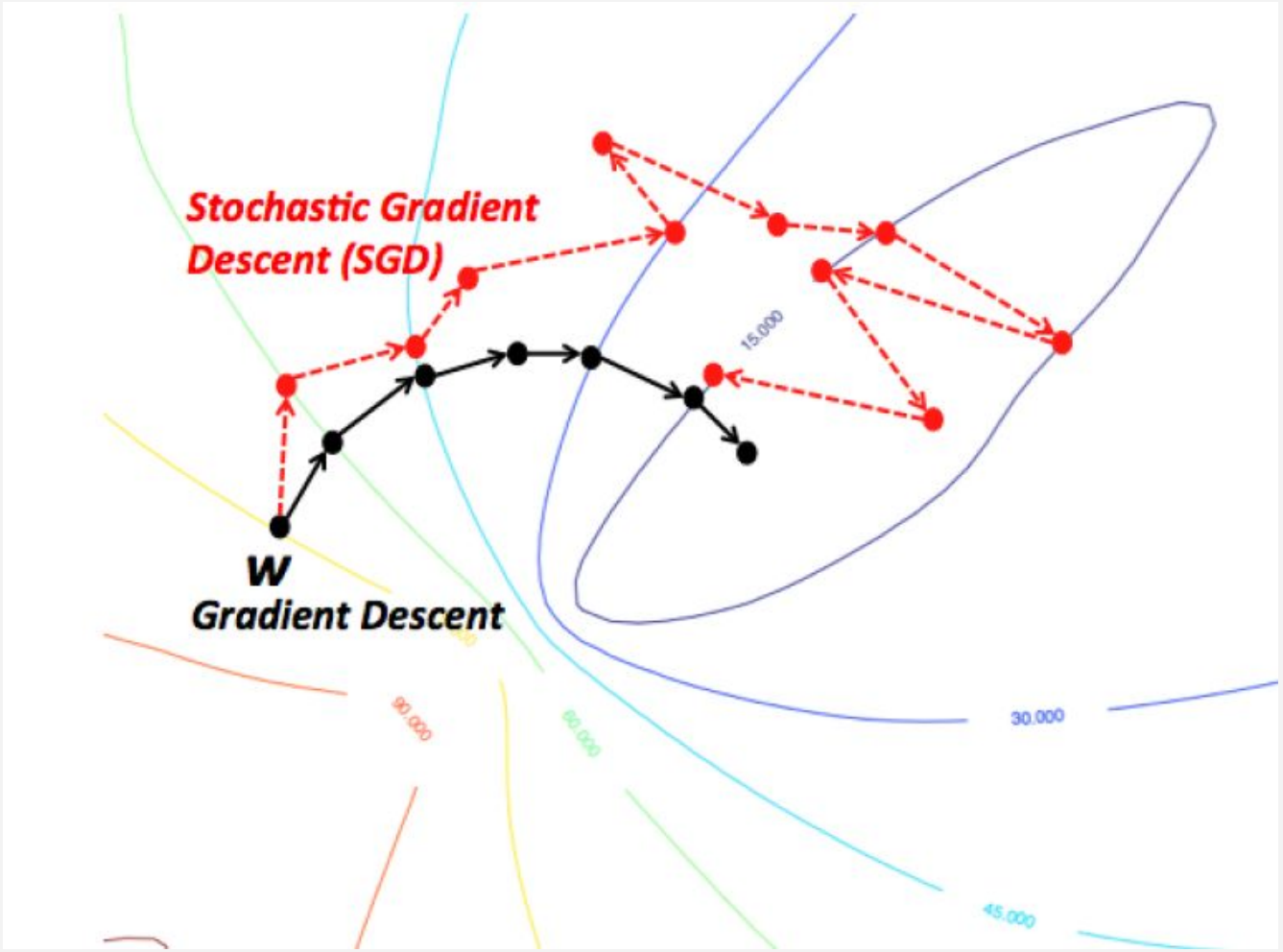
Descenso de gradiente por lotes

- Ejemplo
  - 5 lotes
    - 5 modificaciones del parámetro
    - En dirección de la derivada o gradiente



Coste computacional: Descenso de gradiente estocástico

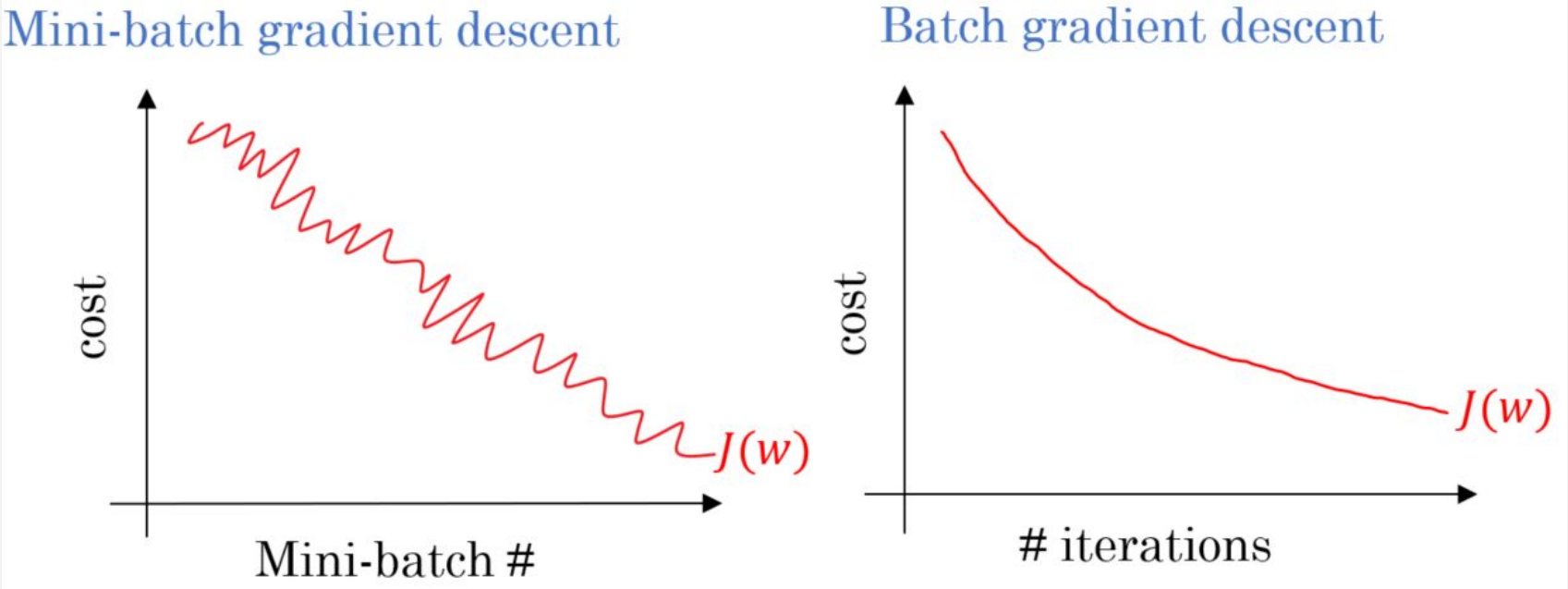
- Descenso clásico
  - Coste de iteración
    - **O(NP)**
- Descenso por lotes
  - Coste de iteración
    - **O(BP)**
    - **B** es un hiperparámetro
- Si **B = N**
  - Igual que clásico
- Si **B = 1**
  - Gradientes demasiado ruidoso
- Si B = intermedio
  - Gradientes ruidosos
    - Pero buenos
  - Más pasos/iteraciones
    - Pero más rápidas

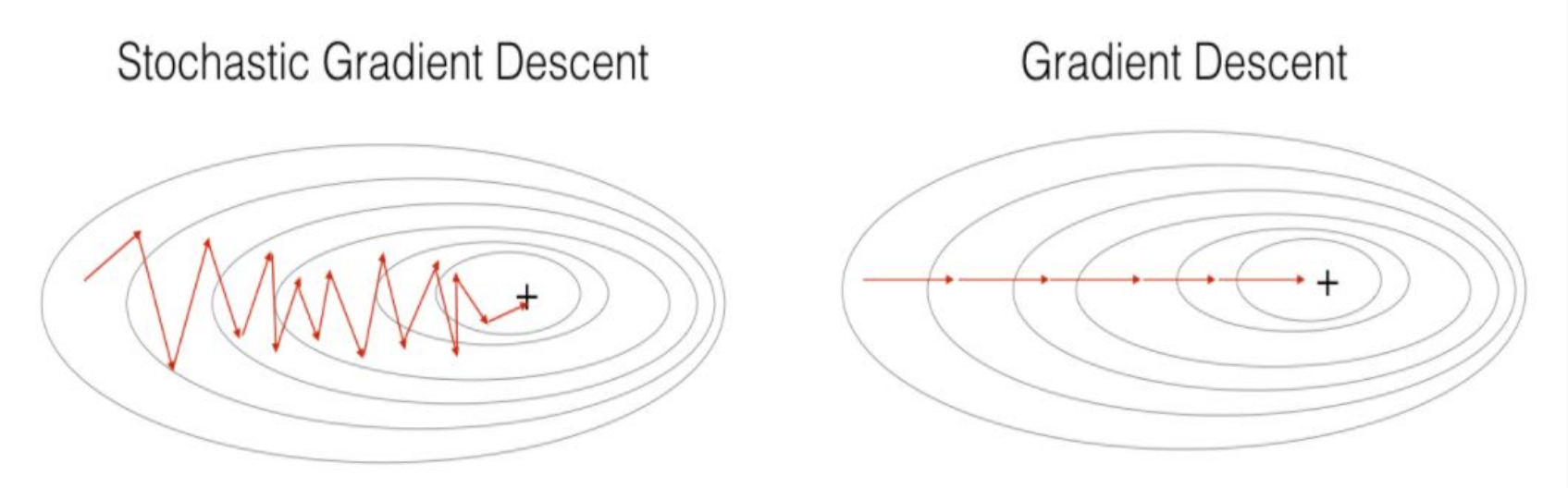


Por lotes = ruidoso = **estocástico**

Descenso de Gradiente Estocástico - Efecto

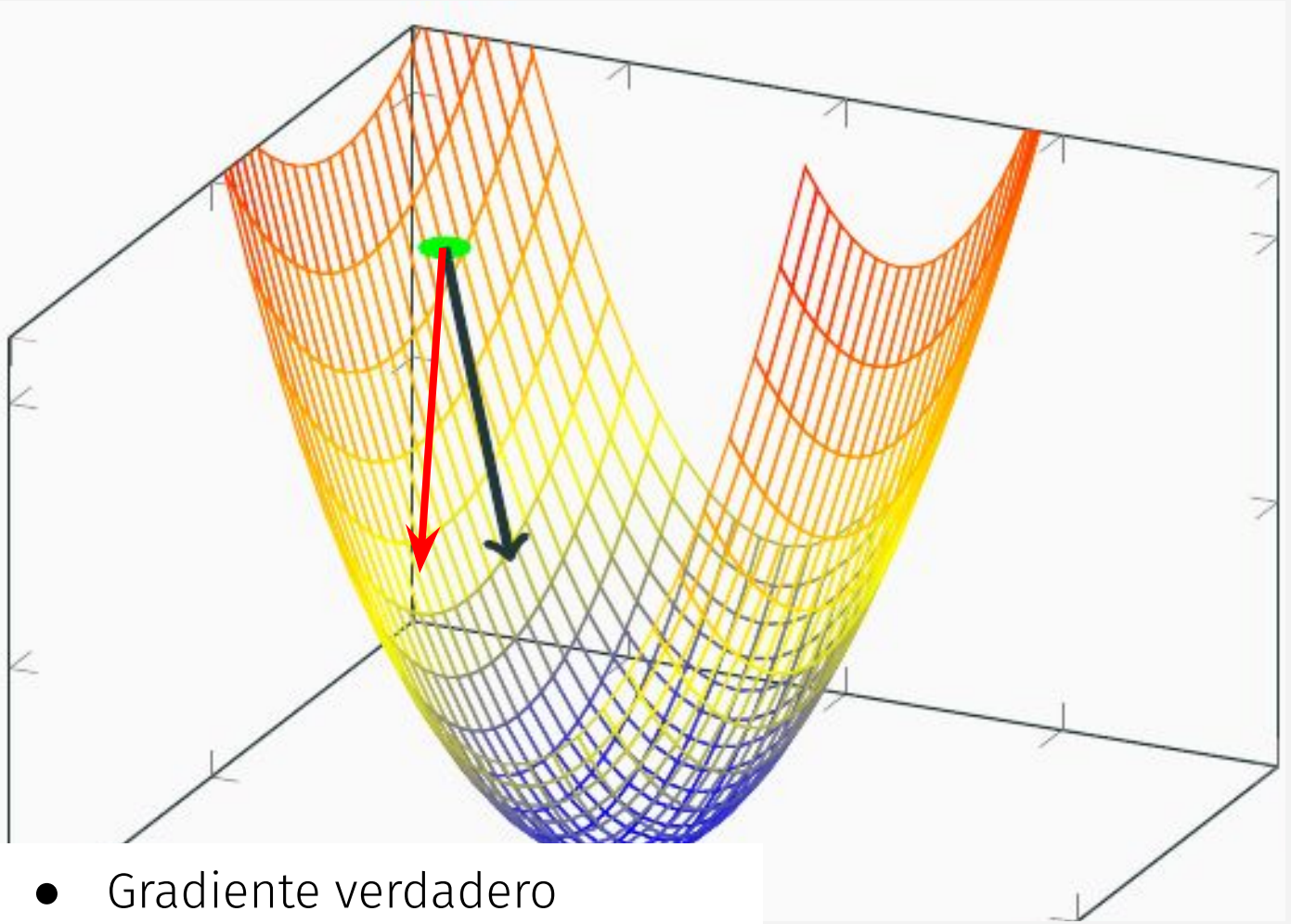
- Error (cost)
  - Cambia de forma “ruidosa”
  - Algunas iteraciones aumentan el error
    - Pero decrece en promedio
- (minibatch = estocástico)
- (batch = clásico)





¿Por qué funciona?

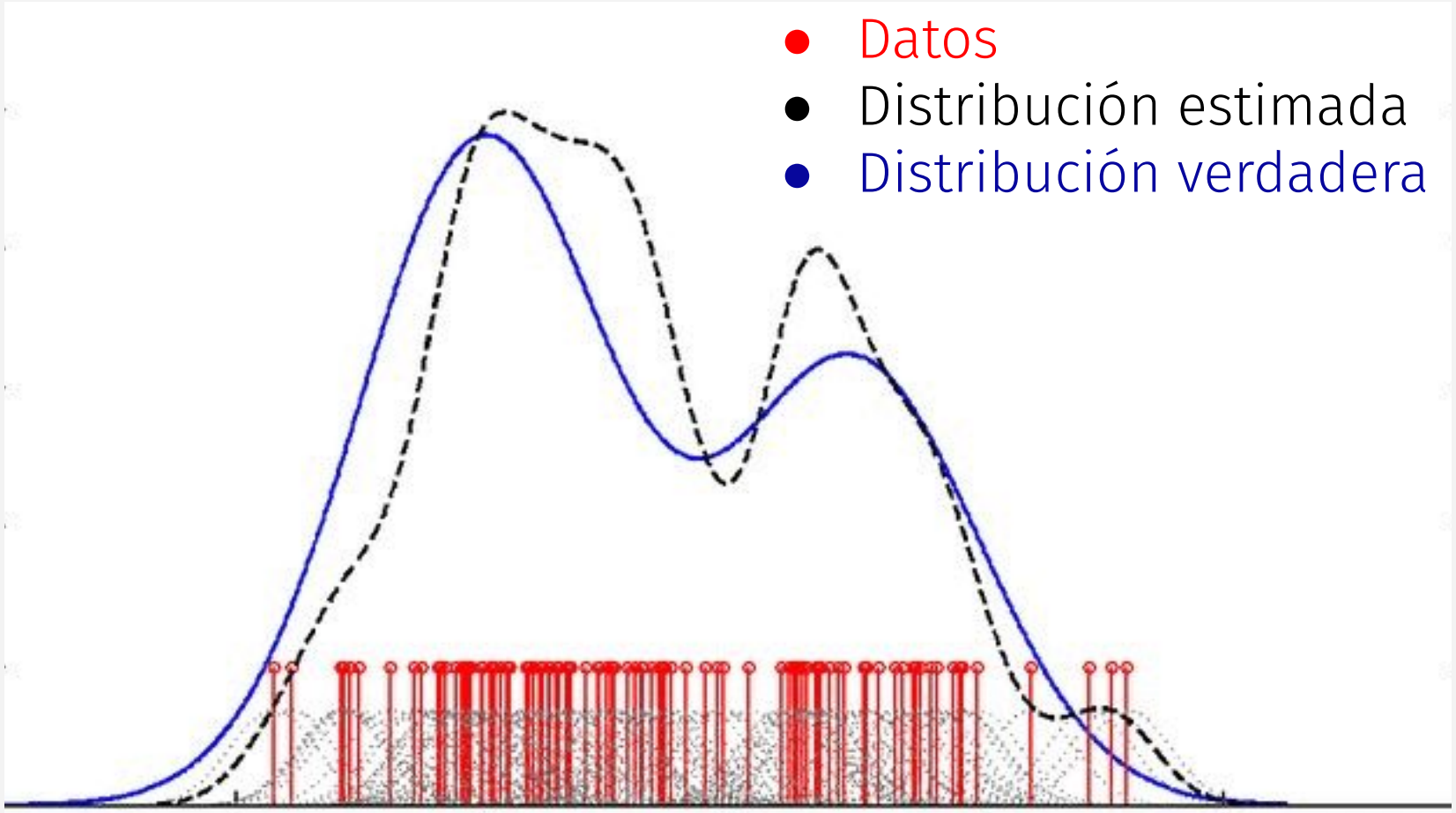
- **Descenso de Gradiente Clásico**
  - Utiliza todas las **muestras**
  - Gradiente es exacto
    - Respecto de las **muestras**
  - Siguen siendo **muestras**
    - **No es exacto (es ruidoso)**
      - Respecto de la distribución
- **Descenso de Gradiente Estocástico**
  - Utiliza una **muestra** de la **muestra**
    - Gradiente **no** es **exacto**
      - **Significativo estadísticamente**
      - **Útil para entrenar**

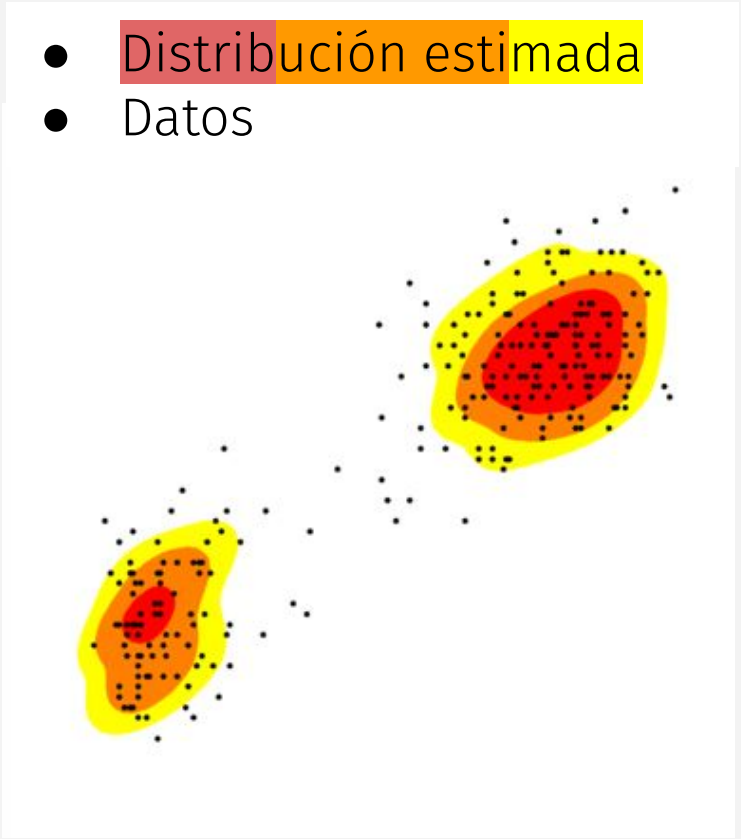


- Gradiente verdadero
- Gradiente estimado

¿Por qué funciona?

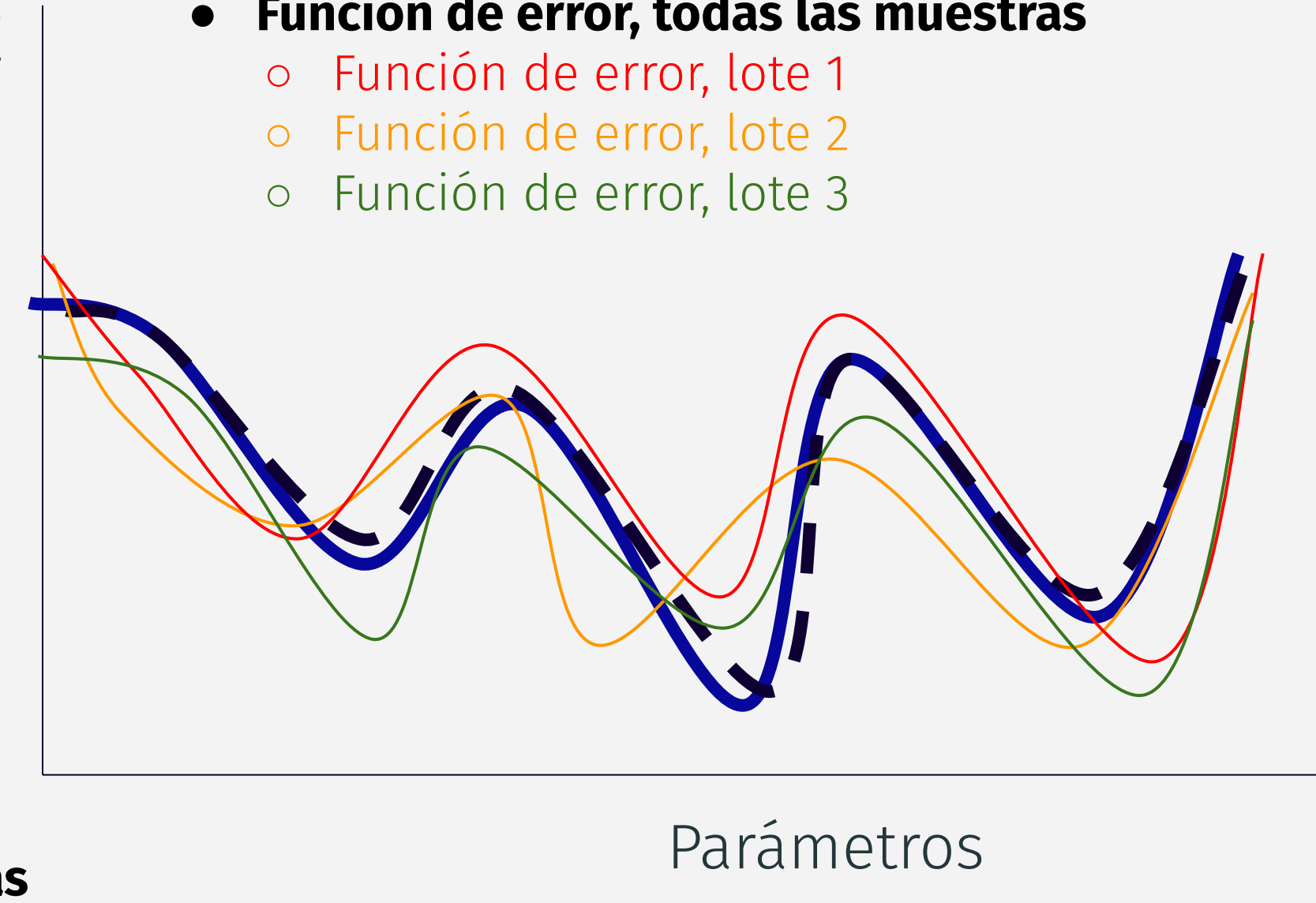
- Distribuciones vs muestras
  - La **distribución** nunca se completamente
    - La estimamos a través de **muestras**





¿Por qué funciona?

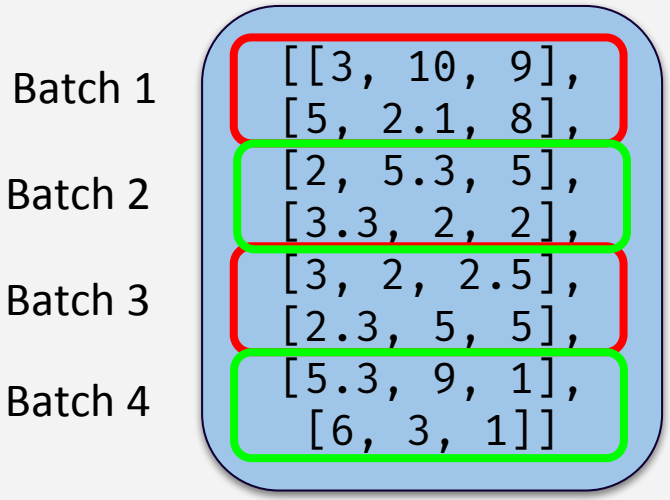
- Función de error de la distribución
  - Imposible de conocer
- Función de error de la muestra completa
  - Utilizada por Descenso Clásico
- Función de error de cada lote
  - Utilizada por Descenso Estocástico
  - **Cada lote redefine la función de error**
    - Se **redefinen** las **derivadas**
    - **Equivalente: ruidosas**

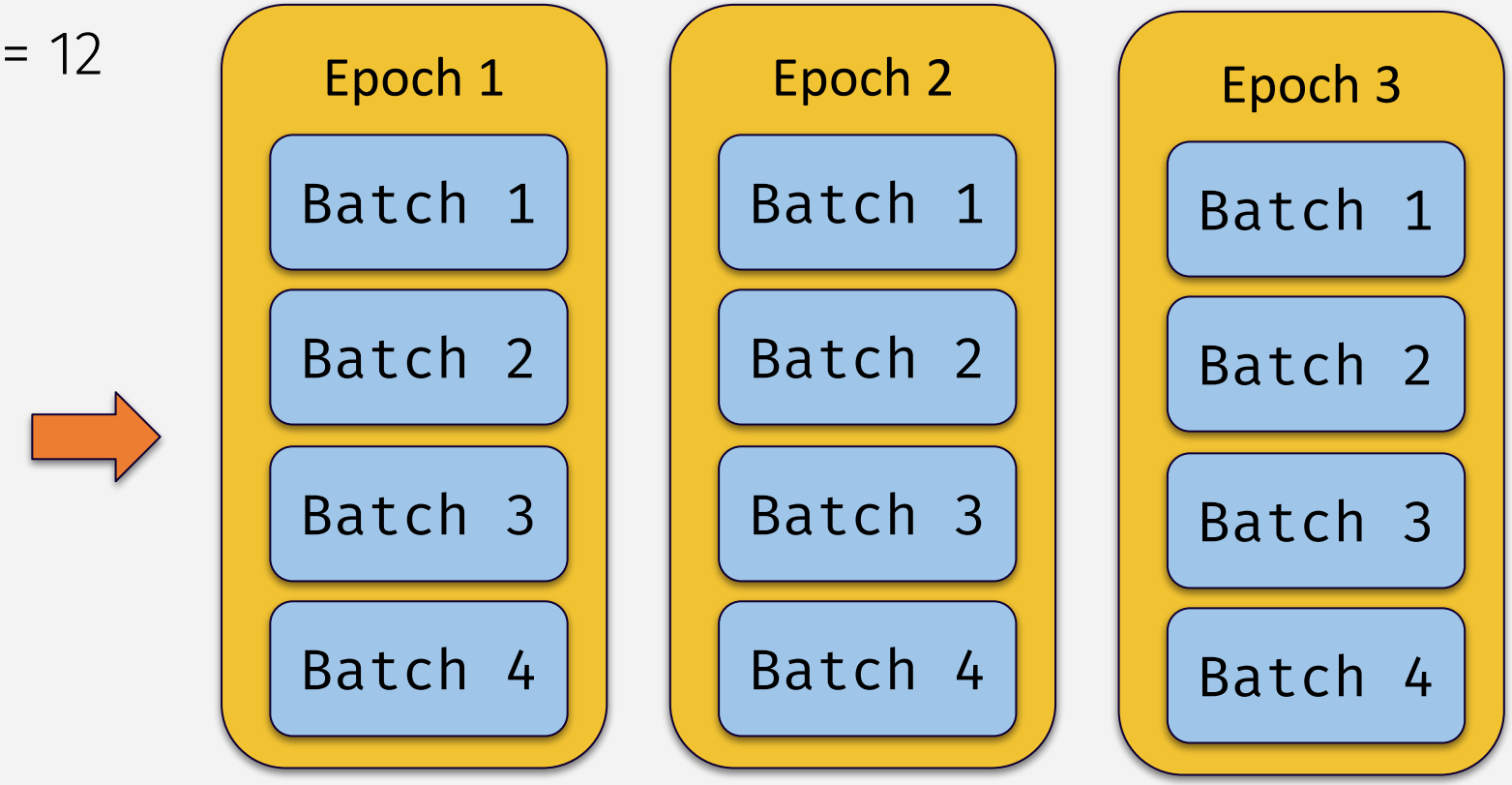


- **Función de error de la distribución**
- **Función de error, todas las muestras**
  - Función de error, lote 1
  - Función de error, lote 2
  - Función de error, lote 3

Épocas, tamaños de lote e iteraciones

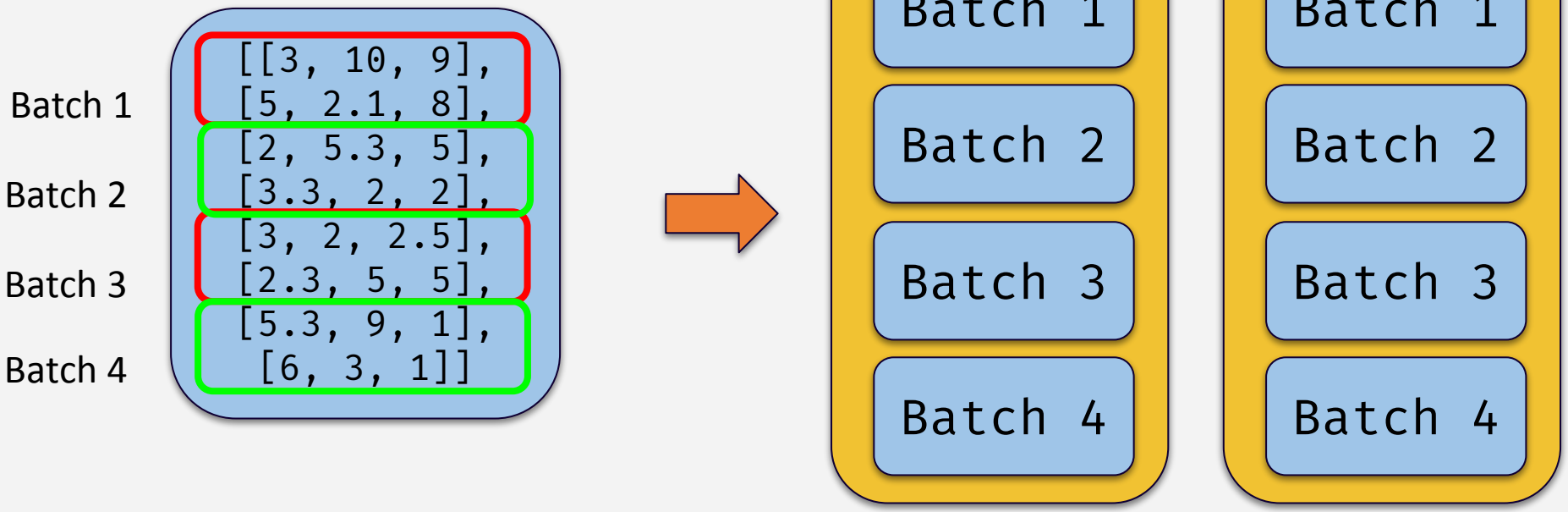
- Épocas = epochs
  - Cantidad de iteraciones de entrenamiento
    - En relación al tamaño del dataset
- Ejemplo con **N=8, batch_size=2, y epochs = 3**
  - Iteraciones reales por epoch = 4
    - N/batch_size
  - Iteraciones reales totales = 12
    - N/batch_size*epochs





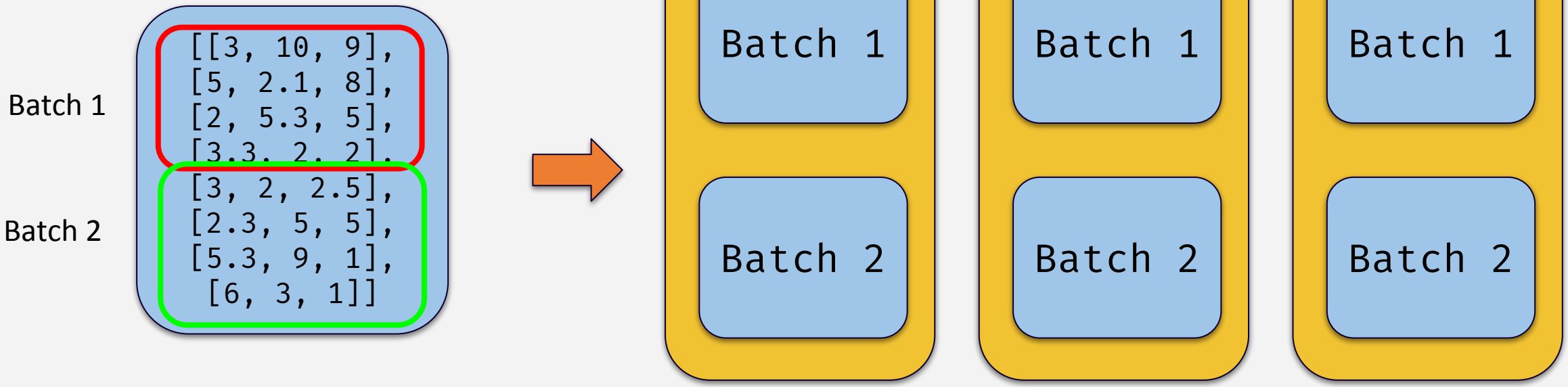
Épocas, tamaños de lote e iteraciones

- Épocas = epochs
  - Cantidad de iteraciones de entrenamiento
    - En relación al tamaño del dataset
- Ejemplo con **N=8, batch_size=2, y epochs = 2**
  - Iteraciones reales por epoch = 4
    - N/batch_size
  - Iteraciones reales totales = 8
    - N/batch_size*epochs



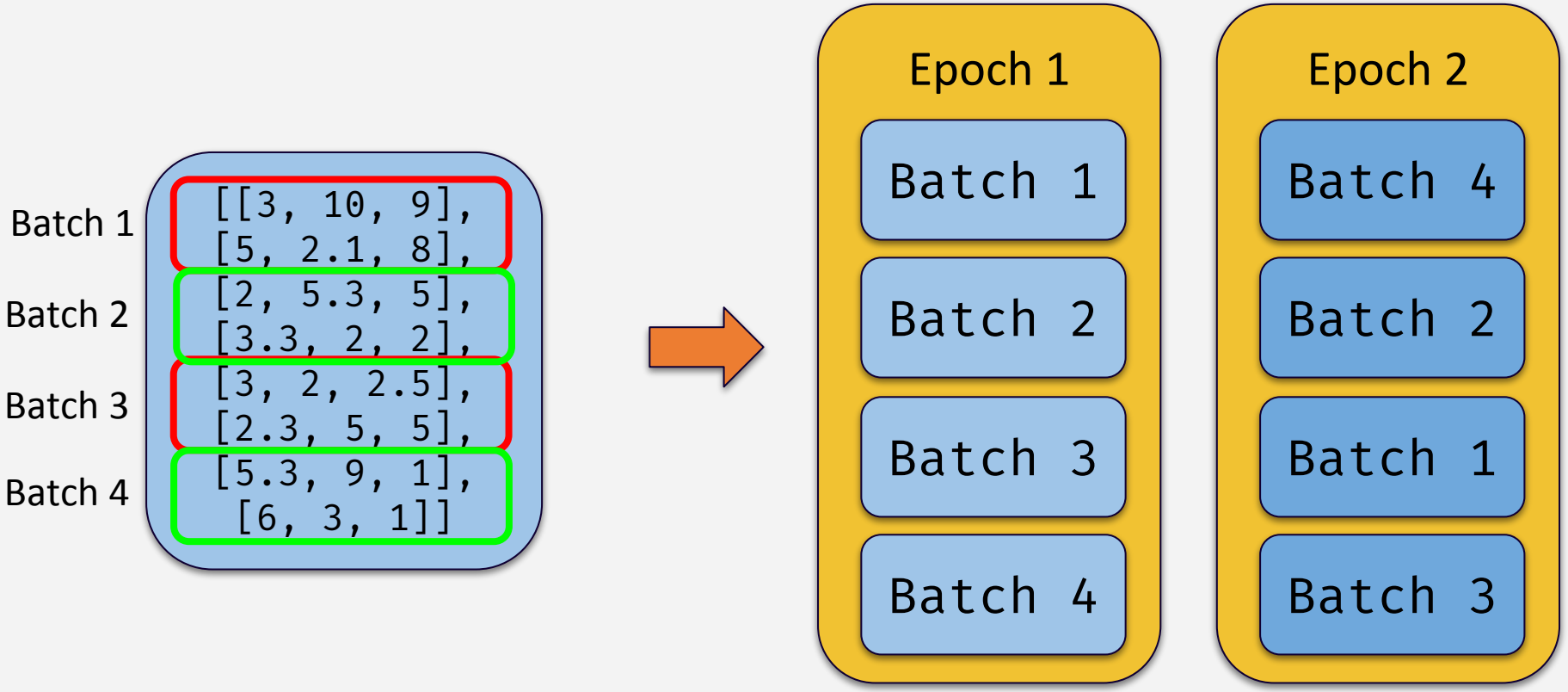
Épocas, tamaños de lote e iteraciones

- Épocas = epochs
  - Cantidad de iteraciones de entrenamiento
    - En relación al tamaño del dataset
- Ejemplo con **N=8, batch_size=4, y epochs = 3**
  - Iteraciones reales por epoch = 2
    - N/batch_size
  - Iteraciones reales totales = 6
    - N/batch_size*epochs



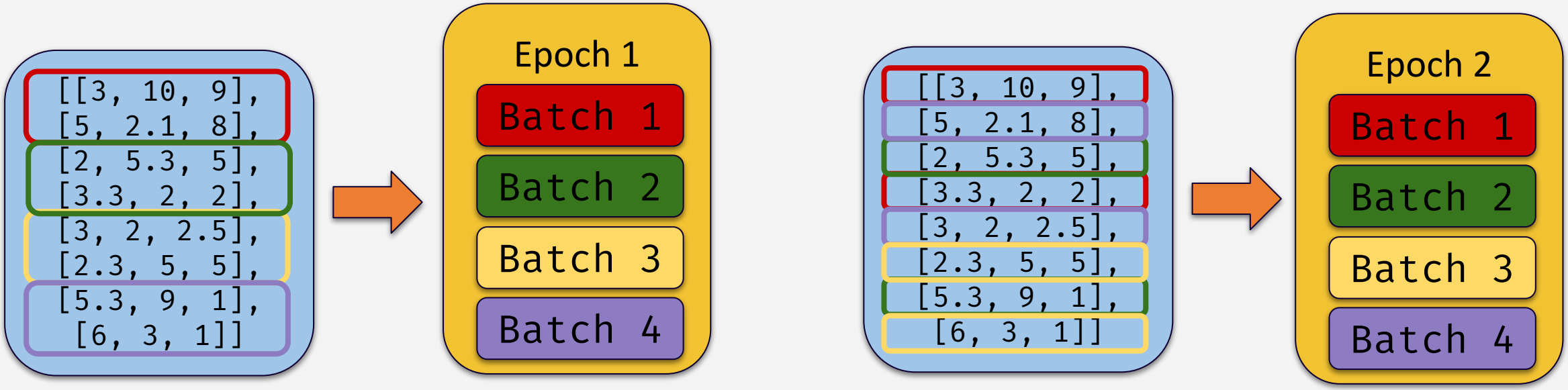
Desordenar lotes entre épocas

- Ayuda al entrenamiento
  - Genera ruido
  - Remueve efectos de orden



Desordenar ejemplos entre épocas

- Mismo objetivo que desordenar lotes
- Mismo efecto
  - Distinto orden de ejemplos
    - Distintos lotes
  - Aún más desorden



Paréntesis: nomenclatura

- Lote = Batch
  - Tamaño de lote = Batch Size = B
- Épocas = Epochs
  - Algunos las llaman **iteraciones** 🤦‍♀️
- Descenso de Gradiente **Tradicional**
  - *También conocido como*
    - Descenso de Gradiente **Batch**
      - (Batch Gradient Descent)
    - Pero **NO** utiliza lotes 🤦‍♀️
      - Proceso **Batch** vs **Online** (fábricas)

Paréntesis: nomenclatura

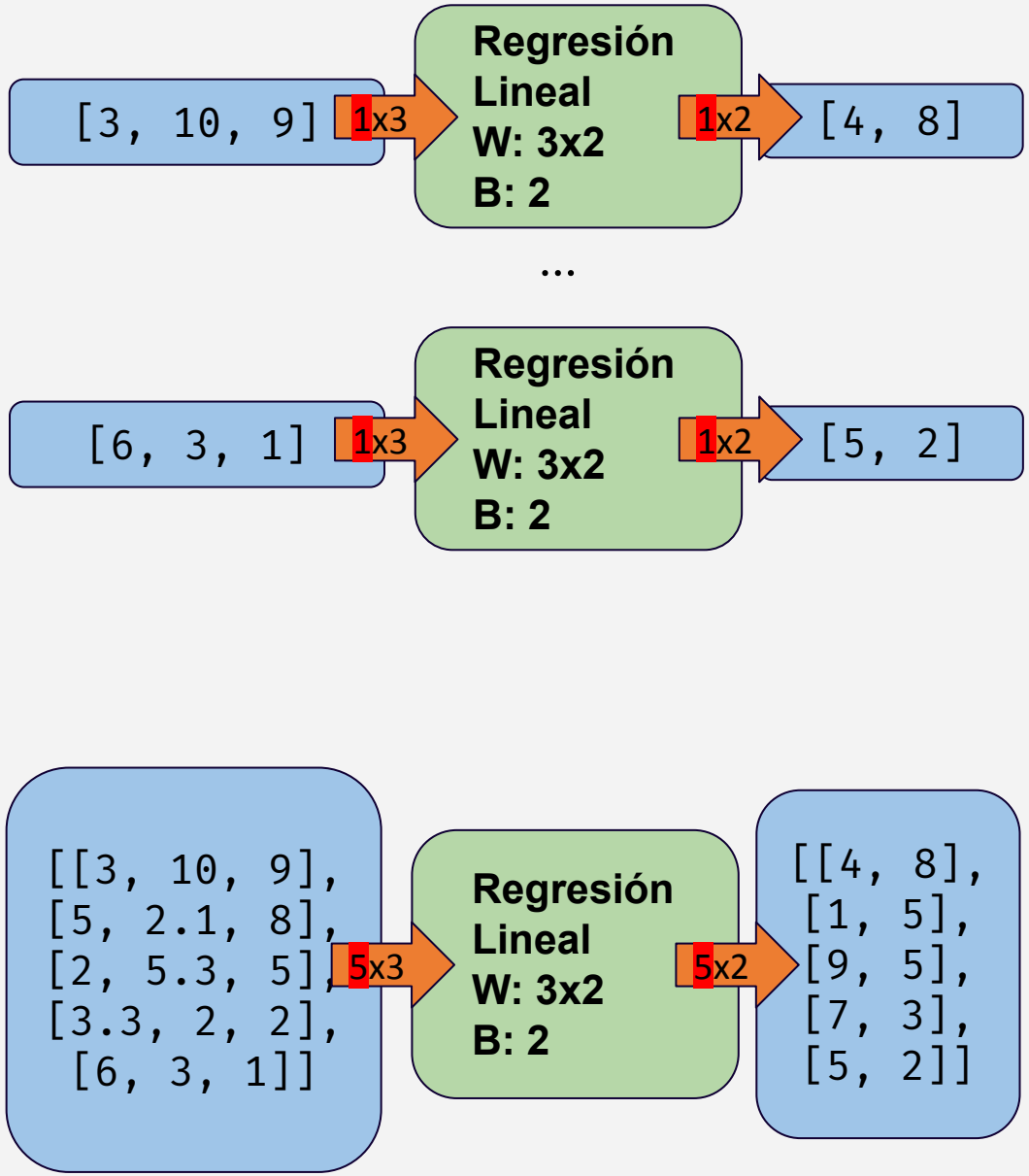
- Descenso de Gradiente **Estocástico**
  - Utiliza **lotes/batchs**
    - *También conocido como*
      - Descenso de Gradiente **mini-batch** 🤦‍♀️
- Descenso de Gradiente **Online**
  - Utiliza lotes de tamaño 1
    - B = 1
  - Simula reentrenar el modelo con nuevos ejemplos continuamente
  - *También conocido como*
    - Descenso de Gradiente **Estocástico** 🤦‍♀️

Tamaño de lote

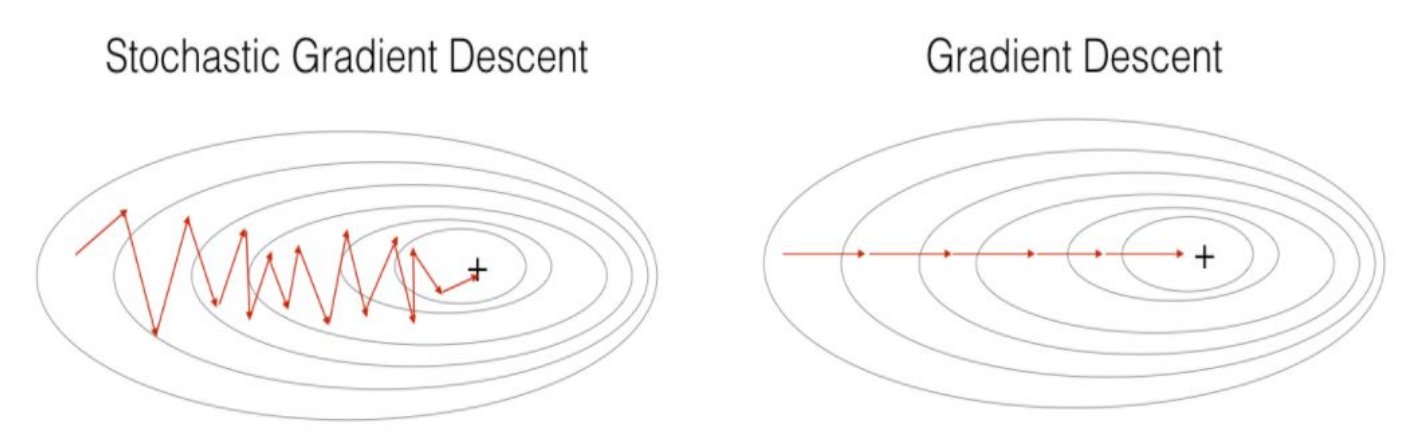
- **Predicción** por lotes
  - Más eficiente
    - Multiplicaciones de matrices
    - pre-alocación de memoria
  - Con GPUs:
    - Maximiza el uso de la memoria
    - Minimiza overhead copias GPU←→ CPU
- Tamaño de lote
  - No se puede usar todo el dataset al mismo tiempo!
    - N = O(millones)
    - No entra en memoria
- Generalmente **batch_size** = potencia de 2
  - 8, 16, 32, 64, 128, 256, 512
  - Más de 512 es redundante para entrenar

Predicción por lotes

- **Predicción de a un ejemplo**
  - Multiplicación de matrices muy pequeñas
  - Overhead de copias
  - Mala localidad de memoria
  - Con GPUs:
    - overhead copias GPU←→ CPU
- **Predicción por lotes**
  - Resuelve estos problemas
  - Tamaño de lote
    - Tan grande como sea posible
      - Tamaño RAM



Resumen



Descenso de gradiente tradicional	Descenso de gradiente Estocástico
Gradiente con **todos** los ejemplos	Gradiente con **lotes** de ejemplos
Gradientes más exactos	Gradientes ruidosos
No puede escapar de mínimos locales	Puede que escape de mínimos locales
Si f convexa, garantiza mínimo global	Mínimo global no garantizado
Lento, poco escalable	Rápido, escalable