Lecture 03

Optimización para redes neuronales - Backpropagation

Optimización

Objetivo: minimizar la función de pérdida $\mathcal{L}(\mathbf{w})$

Algoritmo:

1. Inicializar los pesos: $\mathbf{w}^{(0)} \sim$ aleatorio
2. Para cada iteración $t = 0, 1, 2, \ldots$:
   - Calcular el gradiente:

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}^{(t)})$$

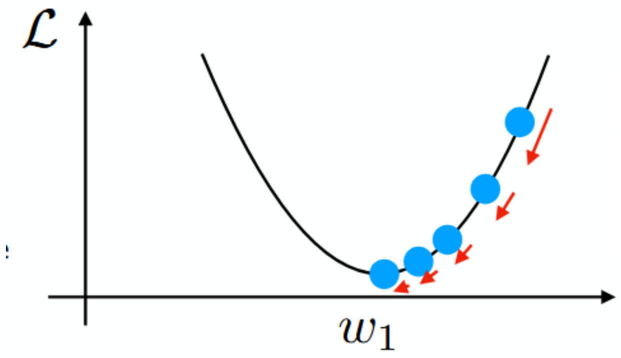
   - Actualizar los pesos:

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}^{(t)})$$

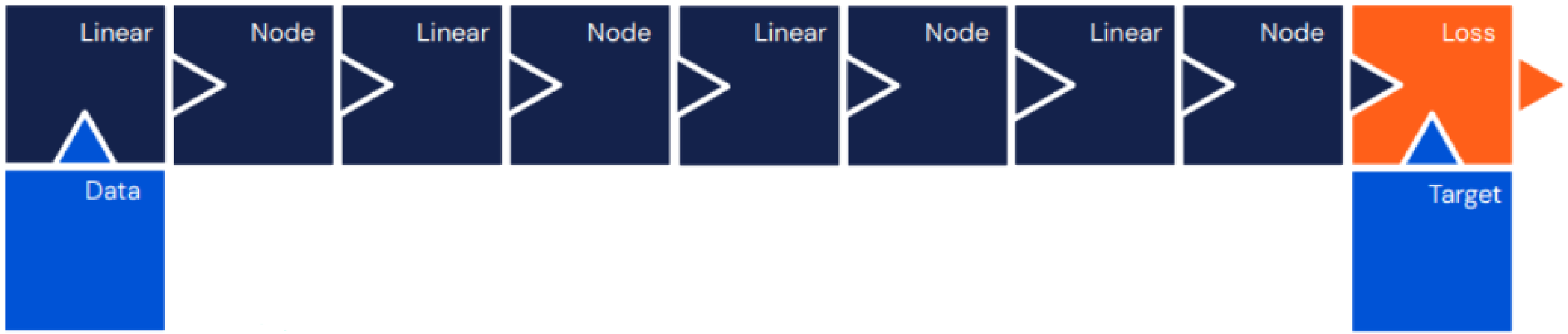
   donde $\eta > 0$ es la tasa de aprendizaje.
3. Repetir hasta convergencia

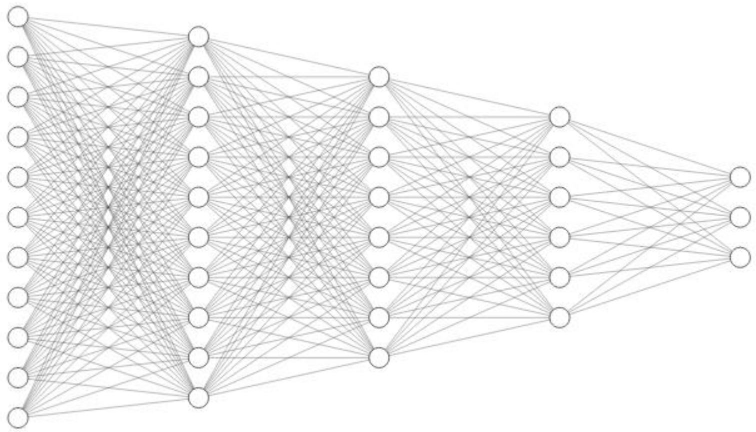
Input: Dataset $D = \{(\mathbf{x}^{(i)}, y^{(i)})\}$, learning rate η
Output: Parameters $\mathbf{W}, \mathbf{b}$
Initialize $\mathbf{W}, \mathbf{b}$;
for *epoch* $t = 1$ **to** T **do**
  foreach *mini-batch* $\mathcal{B} \subset D$ **do**
    Compute gradients $\nabla_{\mathbf{W}} \mathcal{L}_{\mathcal{B}},\ \nabla_b \mathcal{L}_{\mathcal{B}}$;
    Update parameters:;
    $\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} \mathcal{L}_{\mathcal{B}}$;
    $\mathbf{b} \leftarrow \mathbf{b} - \eta \nabla_{\mathbf{b}} \mathcal{L}_{\mathcal{B}}$;
  end
end

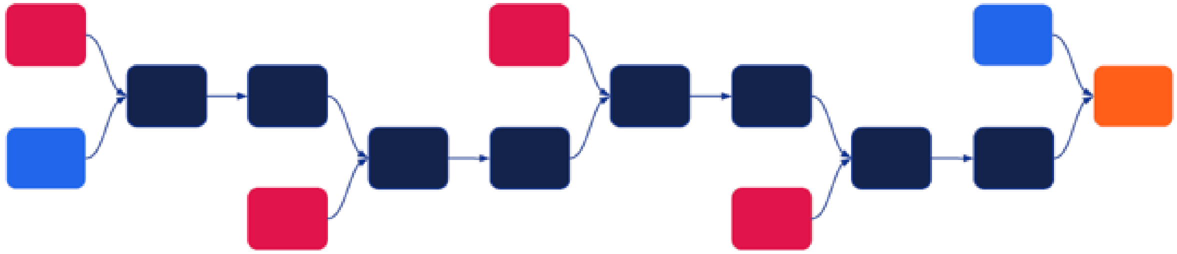
Mini-Batch Gradient Descent



Grafos computacionales

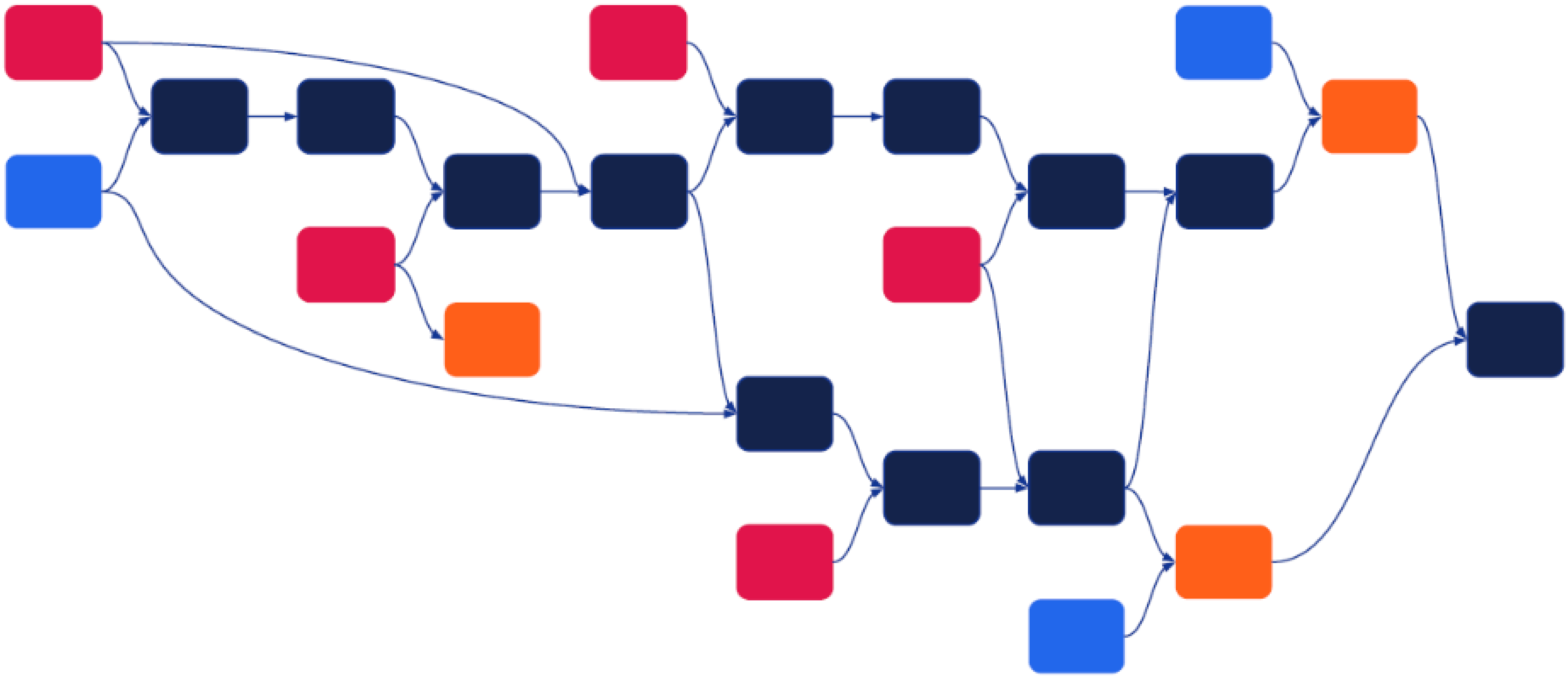






Grafos computacionales

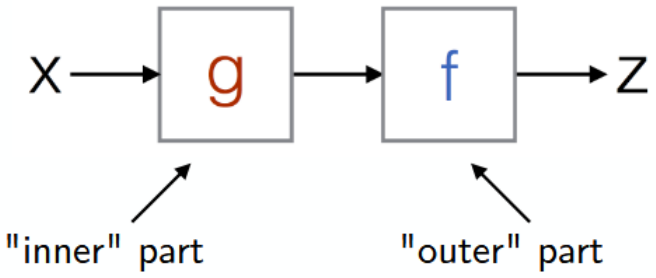
Escuela de Ciencias Aplicadas e Ingeniería



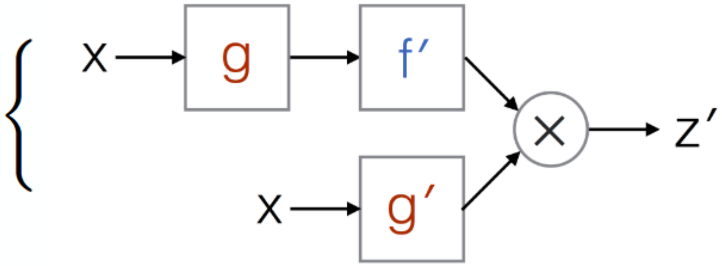
Regla de la cadena

Escuela de Ciencias Aplicadas e Ingeniería

$$F(x) = f(\,g(x)\,) = z$$



$$F'(x) = f'(\,g(x)\,)\,g'(x) = z'$$



$$\frac{d}{dx}[f(g(x))] = \frac{df}{dg} \cdot \frac{dg}{dx}$$

- Imagina que quieres convertir *pesos* a *dólares*, pero primero conviertes pesos a *euros*, y luego euros a dólares.
- Cada conversión es una función:

$$\text{Euros} = f(\text{Pesos}), \quad \text{Dólares} = g(\text{Euros})$$

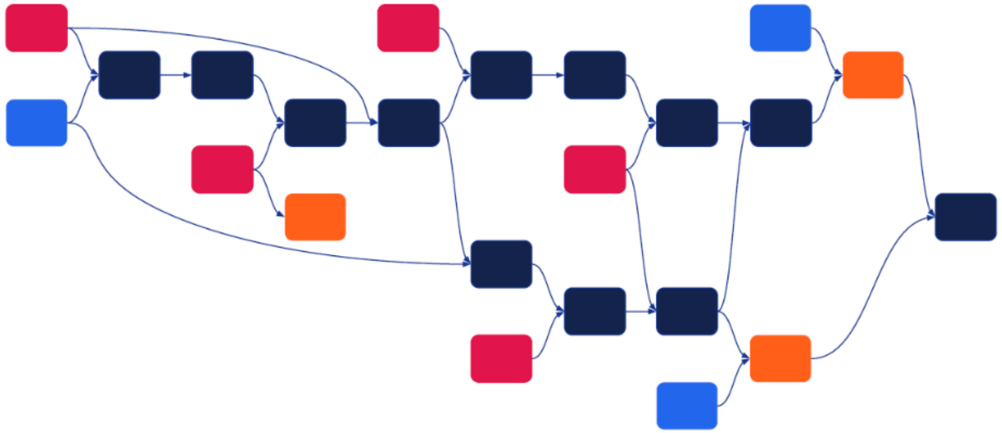
- El cambio total de pesos a dólares se obtiene multiplicando las tasas de cambio

$$\frac{d(\text{Dólares})}{d(\text{Pesos})} = \frac{d(\text{Dólares})}{d(\text{Euros})} \cdot \frac{d(\text{Euros})}{d(\text{Pesos})}$$

- **Interpretación:** la regla de la cadena multiplica los cambios intermedios para obtener el cambio total.

Regla de la cadena

$$\frac{dF}{dx} = \frac{d}{dx}F(x) = \frac{d}{dx}f(g(h(u(v(x)))))$$
$$= \frac{df}{dg} \cdot \frac{dg}{dh} \cdot \frac{dh}{du} \cdot \frac{du}{dv} \cdot \frac{dv}{dx}$$



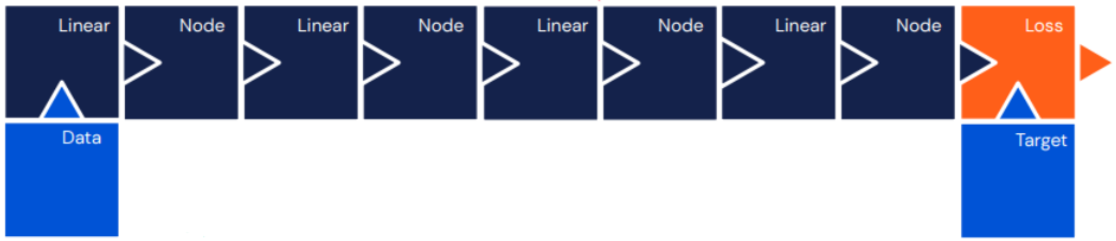
- Para la capa ℓ:

  $$z^{(\ell)} = W^{(\ell)}a^{(\ell-1)} + b^{(\ell)}, \quad a^{(\ell)} = \sigma^{(\ell)}(z^{(\ell)})$$

- Con $a^{(0)} = x$, el modelo completo es:

  $$f(x) = a^{(L)} = \sigma^{(L)}(W^{(L)}\,\sigma^{(L-1)}(\cdots\sigma^{(1)}(W^{(1)}x+b^{(1)})\cdots)+b^{(L)})$$

- **Clave:** $f(x)$ es una función compuesta — usamos la *regla de la cadena* para retropropagar.

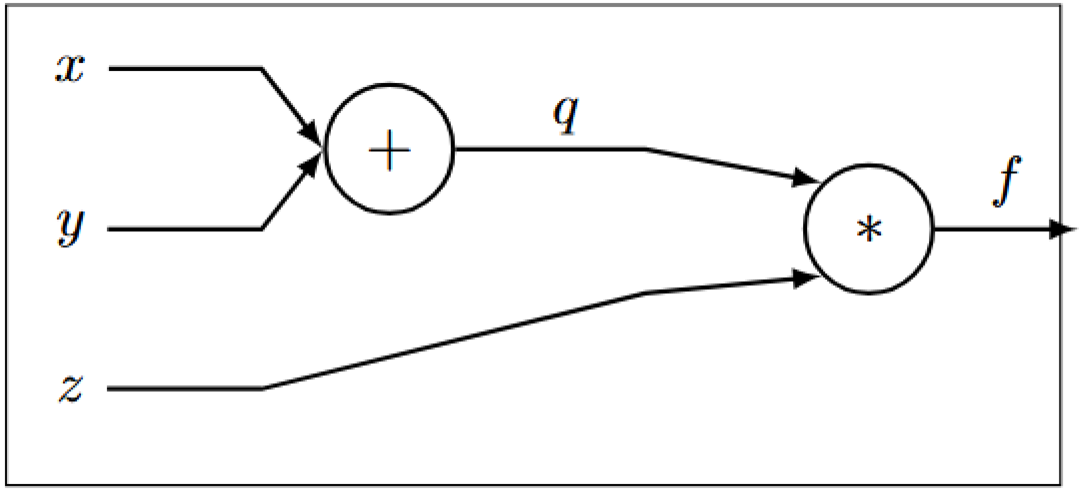


Ejemplo simple

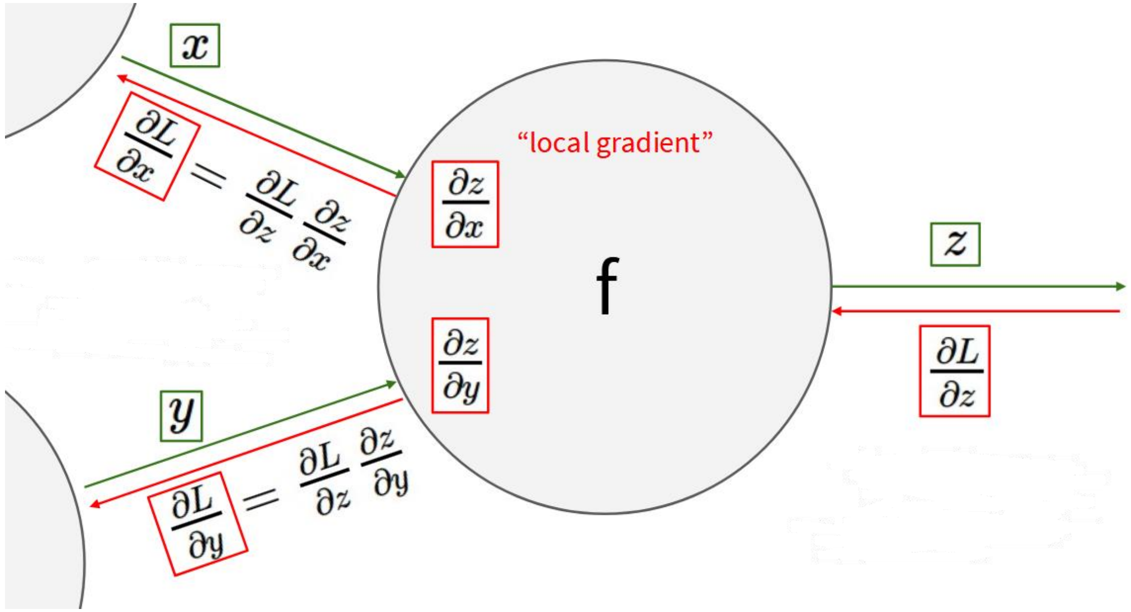
e.g. x = -2, y = 5, z = -4

$$f(x, y, z) = (x + y)\, z$$

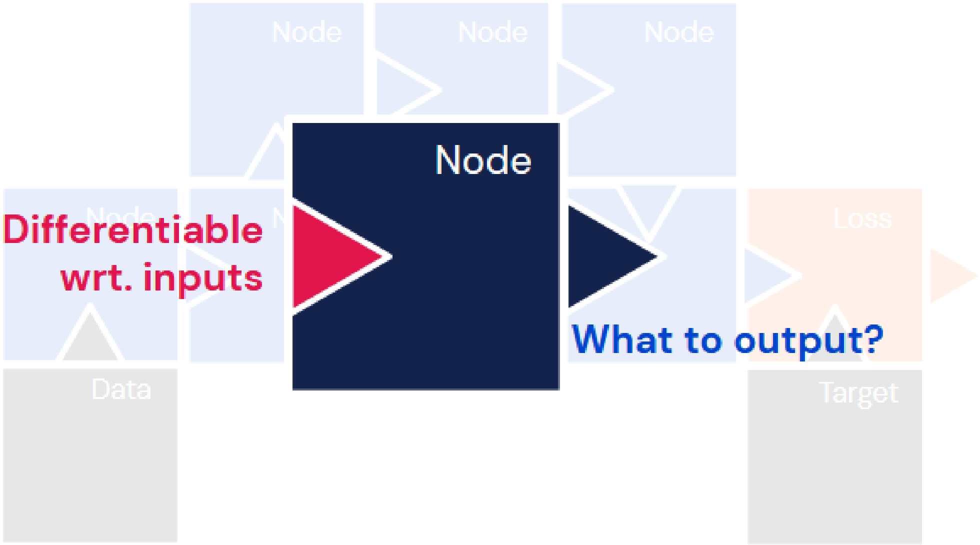
- $q = x + y$
- $f = q \cdot z$



Lego block



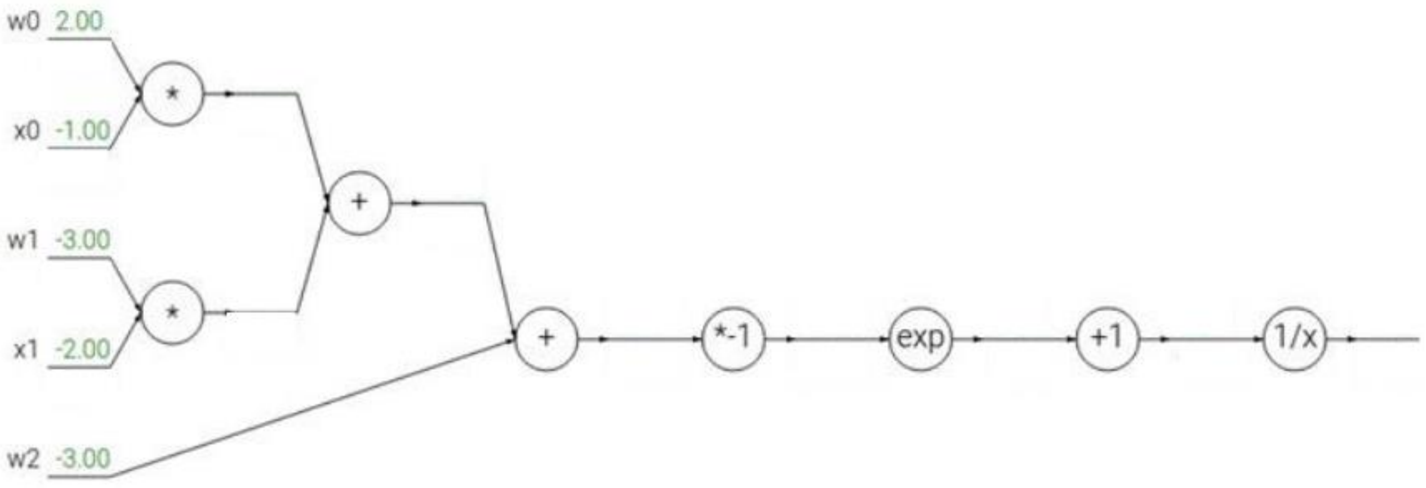
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$



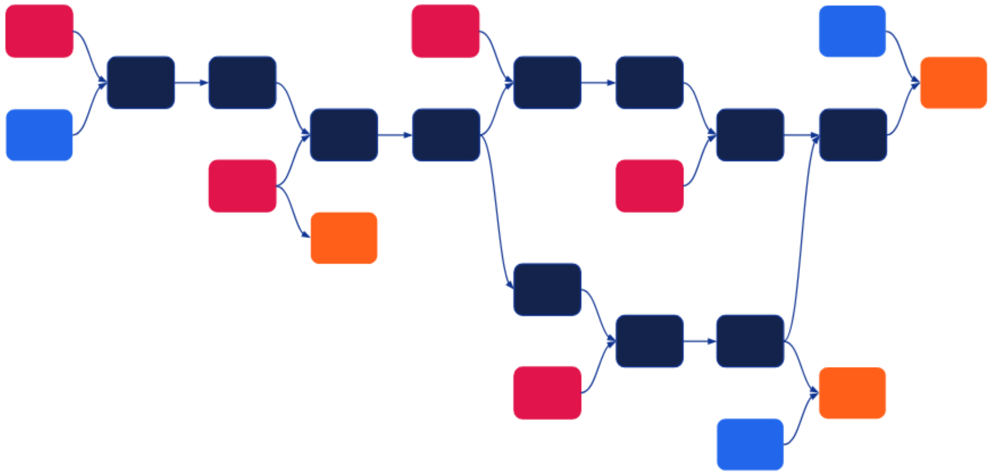
Node
Node
Node
Node
Differentiable
wrt. inputs
What to output?
Loss
Data
Target

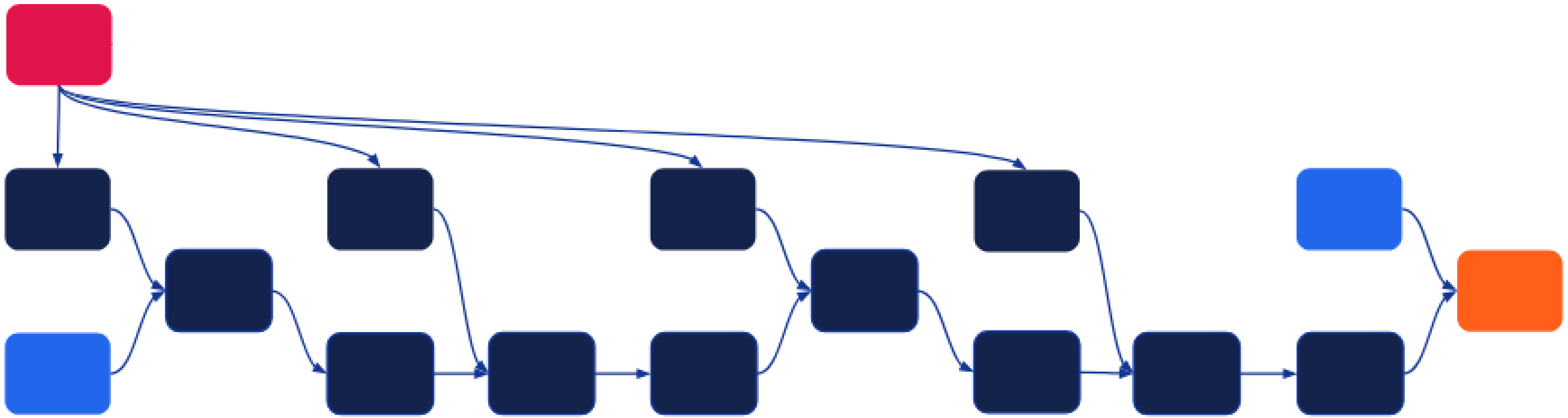
Ejemplo

$$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

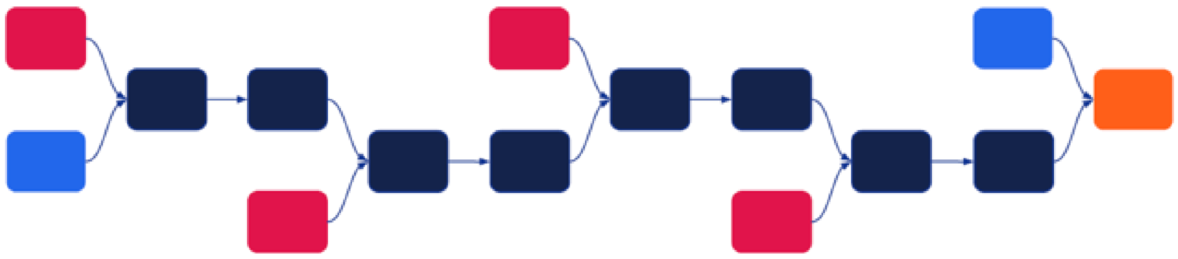


Grafos computacionales





Lego block - backprop



$$f(\mathbf{x})$$

Forward pass

$$\frac{\partial L}{\partial \mathbf{y}} \mathbf{J}_{\mathbf{x}} f(\mathbf{x})$$

Backward pass

Lego block – Linear Layer

$$\mathbf{Z} = \mathbf{X}\mathbf{W}^\top + \mathbf{1}_B \mathbf{b}^\top$$

$\mathbf{X} \in \mathbb{R}^{B \times D_{\text{in}}}$ — input batch

$\mathbf{W} \in \mathbb{R}^{D_{\text{out}} \times D_{\text{in}}}$ — weights

$\mathbf{b} \in \mathbb{R}^{D_{\text{out}}}$ — bias (broadcasted)

$\mathbf{Z} \in \mathbb{R}^{B \times D_{\text{out}}}$ — output

$$\nabla \mathcal{L}_{\mathbf{W}} = (\nabla \mathcal{L}_{\mathbf{Z}})^\top \mathbf{X}$$

$$\nabla \mathcal{L}_{\mathbf{b}} = \sum_{i=1}^{B} \nabla \mathcal{L}_{\mathbf{Z}}[i, :]$$

$$\nabla \mathcal{L}_{\mathbf{X}} = \nabla \mathcal{L}_{\mathbf{Z}}\, \mathbf{W}$$

$\nabla \mathcal{L}_{\mathbf{Z}} \in \mathbb{R}^{B \times D_{\text{out}}}$

Lego block – ReLU

$\mathbf{Z} \in \mathbb{R}^{B \times D}$

$$\mathbf{A} = \mathrm{ReLU}(\mathbf{Z}), \quad A_{ij} = \max(0, Z_{ij})$$

$$\nabla \mathcal{L}_{\mathbf{Z}} = \nabla \mathcal{L}_{\mathbf{A}} \odot \mathbf{1}_{\mathbf{Z}>0}$$

$\nabla \mathcal{L}_{\mathbf{A}} \in \mathbb{R}^{B \times D}$

$\odot$ = multiplicación elemento a elemento

$\mathbf{1}_{\mathbf{Z}>0}$ es la máscara binaria (1 donde $Z_{ij} > 0$, 0 en otro caso)

Lego block – MSE

$$\mathbf{A} \in \mathbb{R}^{B\times 1}$$

$$\mathbf{Y} \in \mathbb{R}^{B\times 1}$$

$$\mathcal{L} = \frac{1}{B}\sum_{i=1}^{B}\left(A_i - Y_i\right)^2$$

$$\nabla\mathcal{L}_{\mathbf{A}} = \frac{2}{B}\left(\mathbf{A} - \mathbf{Y}\right)$$

$$\nabla\mathcal{L}_{\mathbf{A}} \in \mathbb{R}^{B\times K}$$

SoftMax

$\mathbf{Z} \in \mathbb{R}^{B\times K}$ (logits)

$\mathbf{A} \in \mathbb{R}^{B\times K}$ (probabilidades)

$\mathbf{Z} \in \mathbb{R}^{B\times K}$

$$A_{ik} = \frac{e^{Z_{ik}}}{\sum_{j=1}^{K} e^{Z_{ij}}} \iff \mathbf{A}_i = \mathrm{softmax}(\mathbf{Z}_i)$$

$$\nabla\mathcal{L}_{\mathbf{Z}} = \mathbf{A} \odot \left(\nabla\mathcal{L}_{\mathbf{A}} - ((\nabla\mathcal{L}_{\mathbf{A}} \odot \mathbf{A})\,\mathbf{1}_K)\right)$$

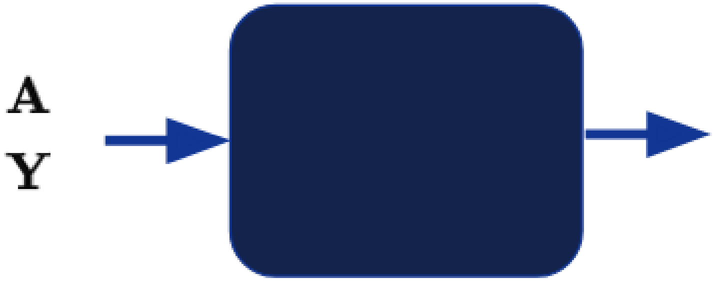
$$(\nabla\mathcal{L}_{\mathbf{Z}})_{ij} = A_{ij}\left((\nabla\mathcal{L}_{\mathbf{A}})_{ij} - \sum_{k=1}^{K}(\nabla\mathcal{L}_{\mathbf{A}})_{ik}\,A_{ik}\right)$$

$\nabla\mathcal{L}_{\mathbf{A}} \in \mathbb{R}^{B\times K}$

Cross-Entropy

$\mathbf{A} \in \mathbb{R}^{B\times K}$: probabilidades por fila (cada fila suma 1)

$\mathbf{Y} \in \mathbb{R}^{B\times K}$: one-hot (o distribuciones objetivo)



$$\mathcal{L} = -\frac{1}{B}\sum_{i=1}^{B}\sum_{k=1}^{K} Y_{ik}\,\log A_{ik}$$

$$(\nabla\mathcal{L}_{\mathbf{A}})_{i,k} = \begin{cases} -\frac{1}{B}\frac{1}{A_{i,y_i}}, & k = y_i \\ 0, & k \neq y_i \end{cases}$$

$$\nabla\mathcal{L}_{\mathbf{A}} = -\frac{1}{B}\,\frac{\mathbf{Y}}{\mathbf{A}}$$

(división elemento a eleme

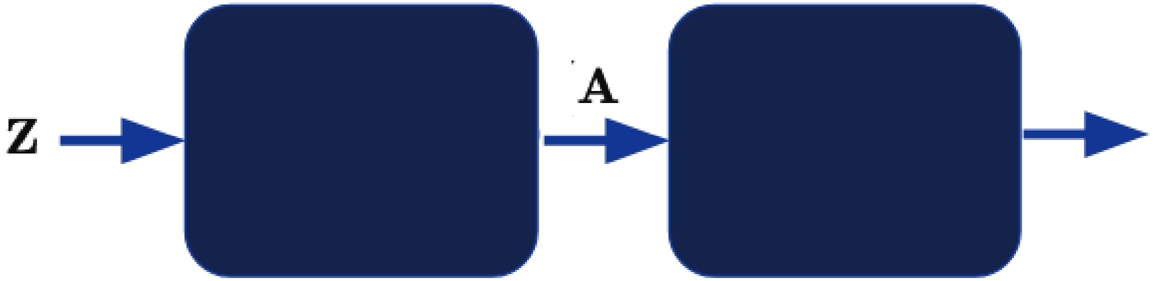
$$(\nabla\mathcal{L}_{\mathbf{A}})_{i,k} = \begin{cases} -\frac{1}{B}\frac{1}{A_{i,y_i}}, & k = y_i \\ 0, & k \neq y_i \end{cases}$$

SoftMax + Cross-Entropy

$\mathbf{Z} \in \mathbb{R}^{B\times K}$: logits

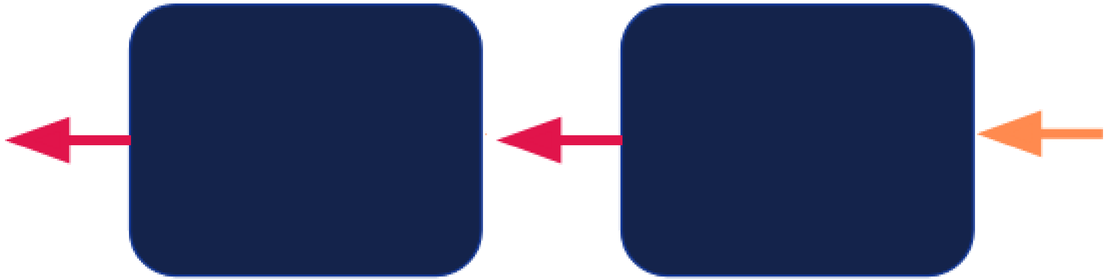
$\mathbf{A} = \mathrm{softmax}(\mathbf{Z})$, $A_{ik} = \dfrac{e^{Z_{ik}}}{\sum_j e^{Z_{ij}}}$

$\mathbf{Y} \in \mathbb{R}^{B\times K}$: one-hot (o distribuciones objetivo)



$$\mathcal{L} = -\frac{1}{B}\sum_{i=1}^{B}\sum_{k=1}^{K} Y_{ik}\,\log A_{ik}$$

$$\nabla\mathcal{L}_{\mathbf{Z}} = \frac{1}{B}\left(\mathbf{A} - \mathbf{Y}\right)$$



Early Stopping

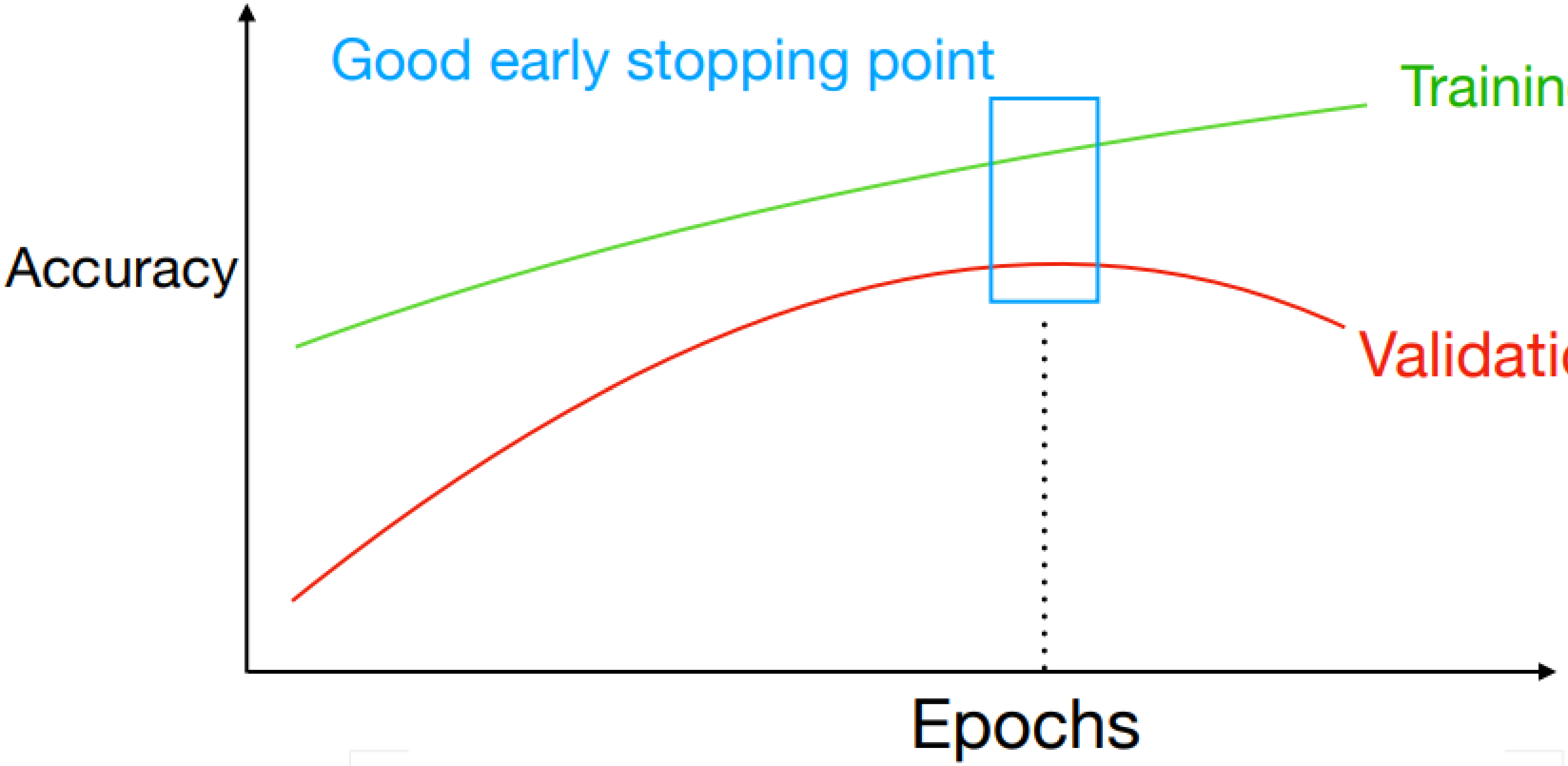
Paso 1: separar el conjunto de datos en 3 partes (siempre recomendado)

- Usar los datos de test solo una vez al final (para una estimación no sesgada del rendimiento de generalización)
- Usar la precisión de validación para el ajuste

Early Stopping

Paso 2: parada anticipada (no es muy común actualmente)

- Reducir *overfitting* mediante la observación de la brecha de precisión de entrenamiento/validación y luego parar en el punto "correcto"



Ridge – Weight Decay


```
#########################################################
## Apply L2 regularization
optimizer = torch.optim.SGD(model.parameters(),
                            lr=0.1,
                            weight_decay=LAMBDA)
#-------------------------------------------------------


for epoch in range(num_epochs):

    #### Compute outputs ####
    out = model(X_train_tensor)

    #### Compute gradients ####
    cost = F.binary_cross_entropy(out, y_train_tensor)
    optimizer.zero_grad()
    cost.backward()
```

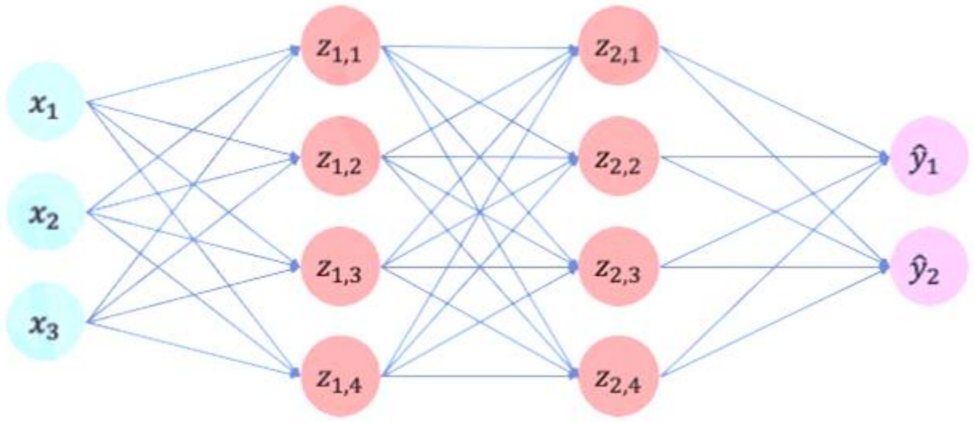
Dropout

Artículos de investigaciones originales:

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.

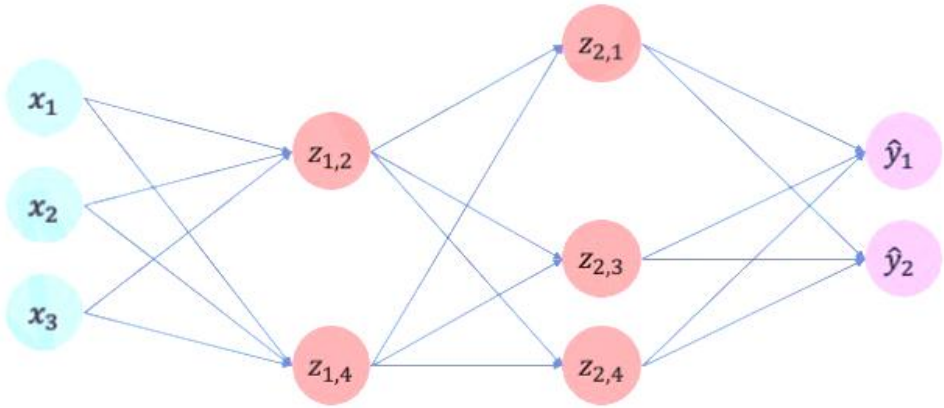
Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.

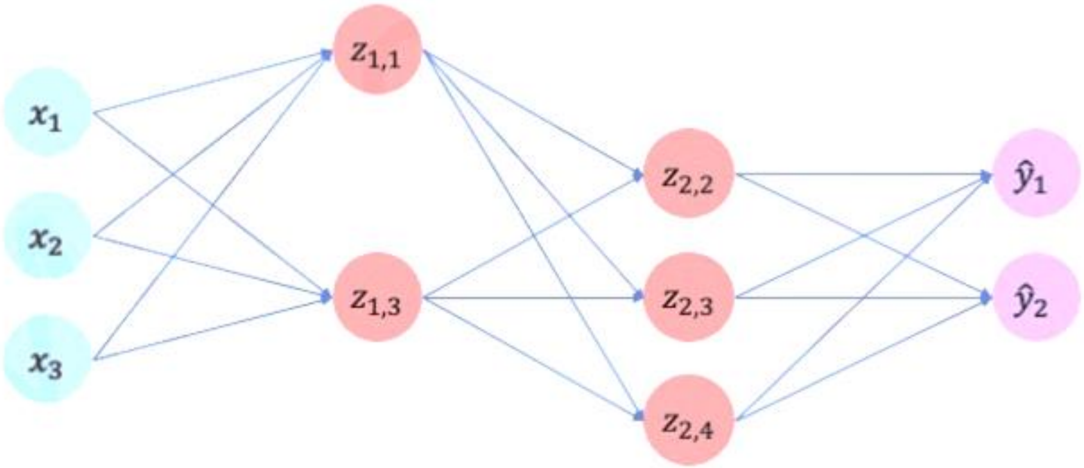
Dropout



Durante el entrenamiento, poner aleatoriamente algunas activaciones en 0

- Normalmente se "eliminan" el 50% de las activaciones en la capa
- Obliga a la red a no depender de un solo nodo





Dropout

¿Comó se eliminan nodos eficientemente?

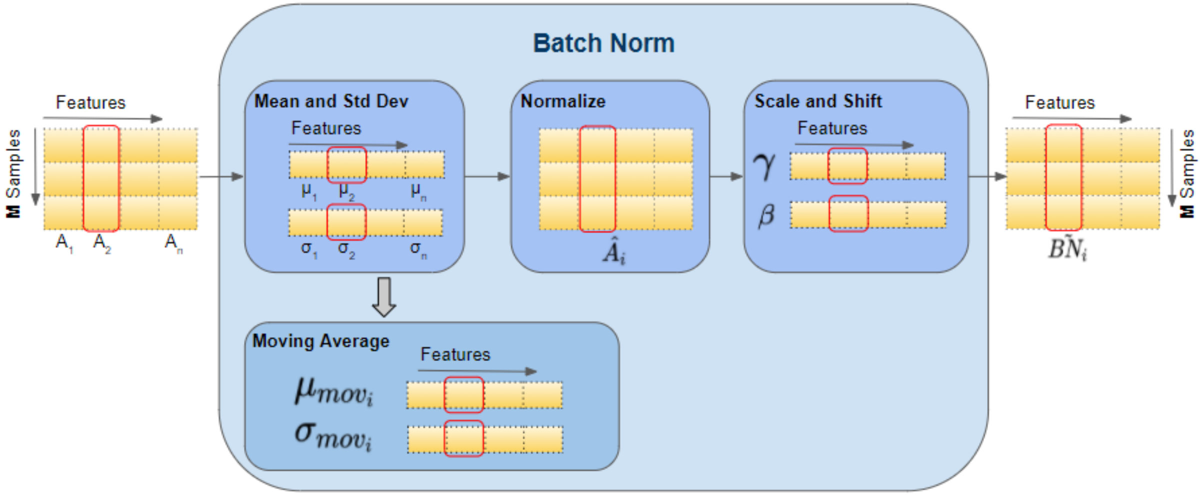
Muestreo de Bernoulli (durante el entrenamiento):

- $p :=$ probabilidad de eliminación
- $v :=$ muestra aleatoria de una distribución uniforme en el rango $[0, 1]$
- $\forall i \in v : u_i := 0$ si $u_i < p$ si no 1
- $a := a \bigodot v$ ($p \times 100\%$ de las activaciones a será 0)

Después del entrenamiento, durante la "inferencia", se deben escalar las activaciones a través de: $a := a \bigodot (1 - p)$

¿Por qué es necesario?

BatchNorm



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

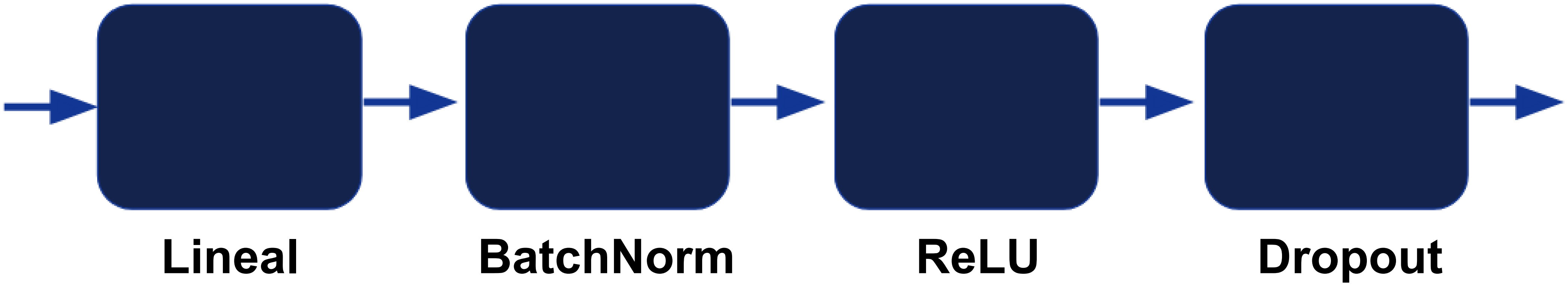
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

https://medium.com/deeplearningmadeeasy/everything-you-wish-to-know-about-batchnorm-6055e07fdce2

Capa completa



Diagnosing and debugging

- **Initialisation** matters
- **Overfit** small sample
- **Monitor** training **loss**
- **Monitor** weights **norms** and **NaNs**
- Add **shape asserts**
- Start with **Adam**
- **Change one thing** at the time

Want to learn more?

Karpathy A. **A Recipe for Training Neural Networks** http://karpathy.github.io/2019/04/25/recipe/ (2019)

- It is always worth spending time on verifying correctness.
- Be suspicious of good results more than bad ones.
- Experience is key, just keep trying!