Lecture 06
Deep Learning

Ph. D. Juan David Martínez Vargas
Ph. D. Raul Andrés Castañeda

Escuela de Ciencias Aplicadas e Ingeniería

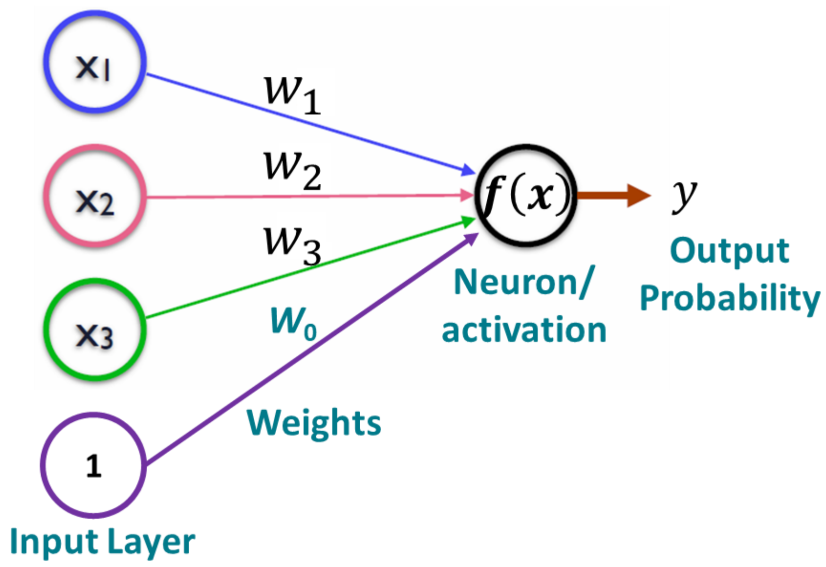
UNIVERSIDAD EAFIT®

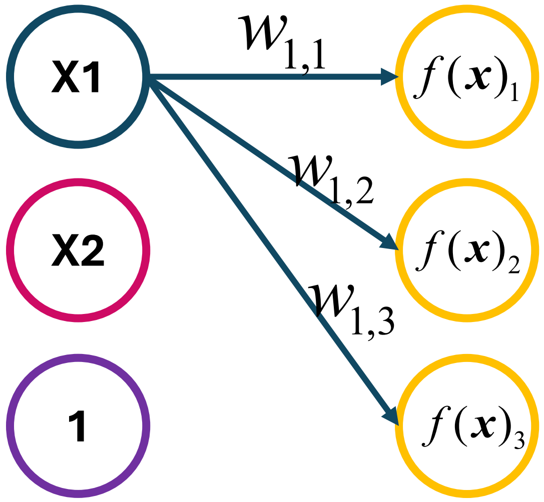
Vigilada Mineducación

Agenda

- Backpropagation
- Métricas de evaluación
- Regularización

Backpropagation



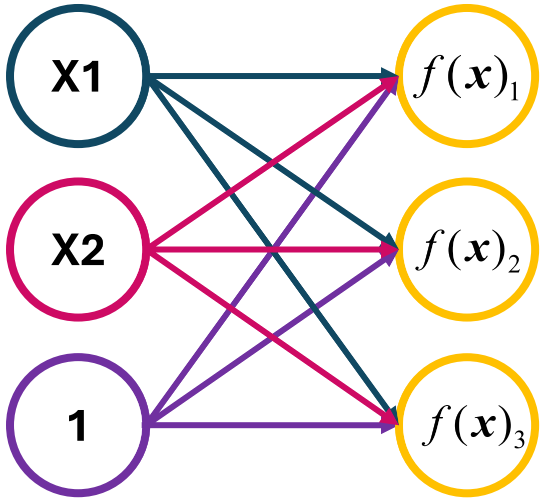


$$f(\boldsymbol{x})_1 = w_{1,1}x_1 + w_{2,1}x_2 + b_1$$

$$f(x_1, x_2, x_3) = x_1 w_1 + x_2 w_2 + x_3 w_3 + w_0$$

$$f(x) = X^T \cdot W$$

$$\hat{y} = acti\left[X^T \cdot W\right] \longrightarrow \hat{y} = \sigma\left[X^T \cdot W\right]$$



$$f(\boldsymbol{x})_2 = w_{1,2}x_1 + w_{2,2}x_2 + b_2$$

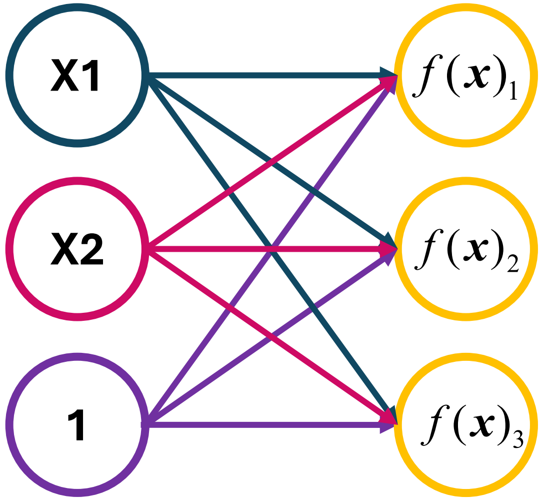
$$f(\boldsymbol{x})_3 = w_{1,3}x_1 + w_{2,3}x_2 + b_3$$

Backpropagation

$$f(x_1, x_2, x_3) = x_1 w_1 + x_2 w_2 + x_3 w_3 + w_0$$

$$f(x) = X^T \cdot W$$

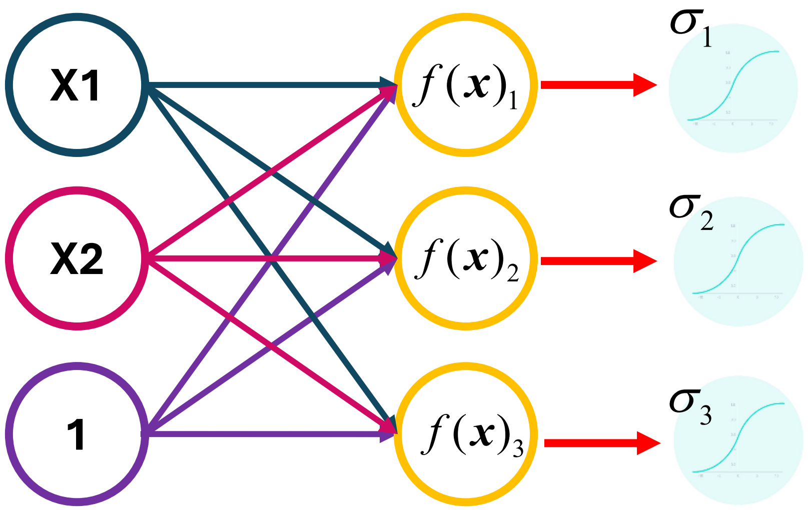
$$\hat{y} = acti\left[X^T \cdot W\right] \longrightarrow \hat{y} = \sigma\left[X^T \cdot W\right]$$



$$f(\boldsymbol{x})_1 = w_{1,1}x_1 + w_{2,1}x_2 + b_1$$

$$f(\boldsymbol{x})_2 = w_{1,2}x_1 + w_{2,2}x_2 + b_2$$

$$f(\boldsymbol{x})_3 = w_{1,3}x_1 + w_{2,3}x_2 + b_3$$

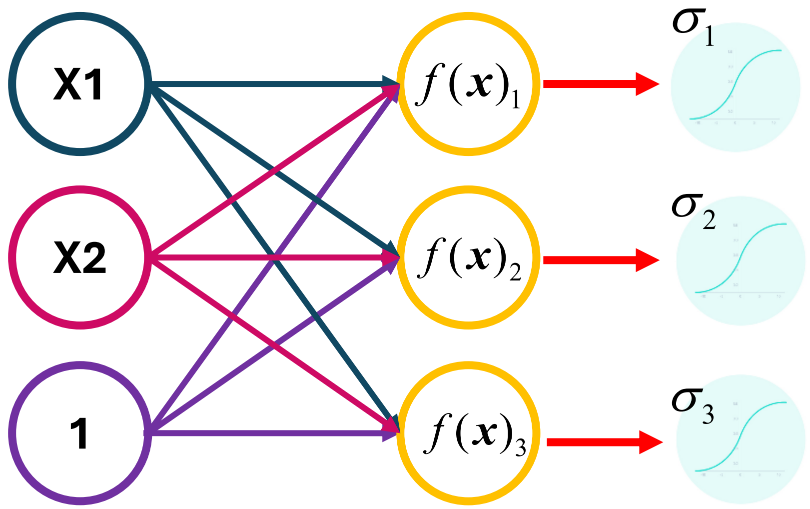


$$f(\boldsymbol{x})_1 = \sigma_1\left[w_{1,1}x_1 + w_{2,1}x_2 + b_1\right]$$

$$f(\boldsymbol{x})_2 = \sigma_2\left[w_{1,2}x_1 + w_{2,2}x_2 + b_2\right]$$

$$f(\boldsymbol{x})_3 = \sigma_3\left[w_{1,3}x_1 + w_{2,3}x_2 + b_3\right]$$

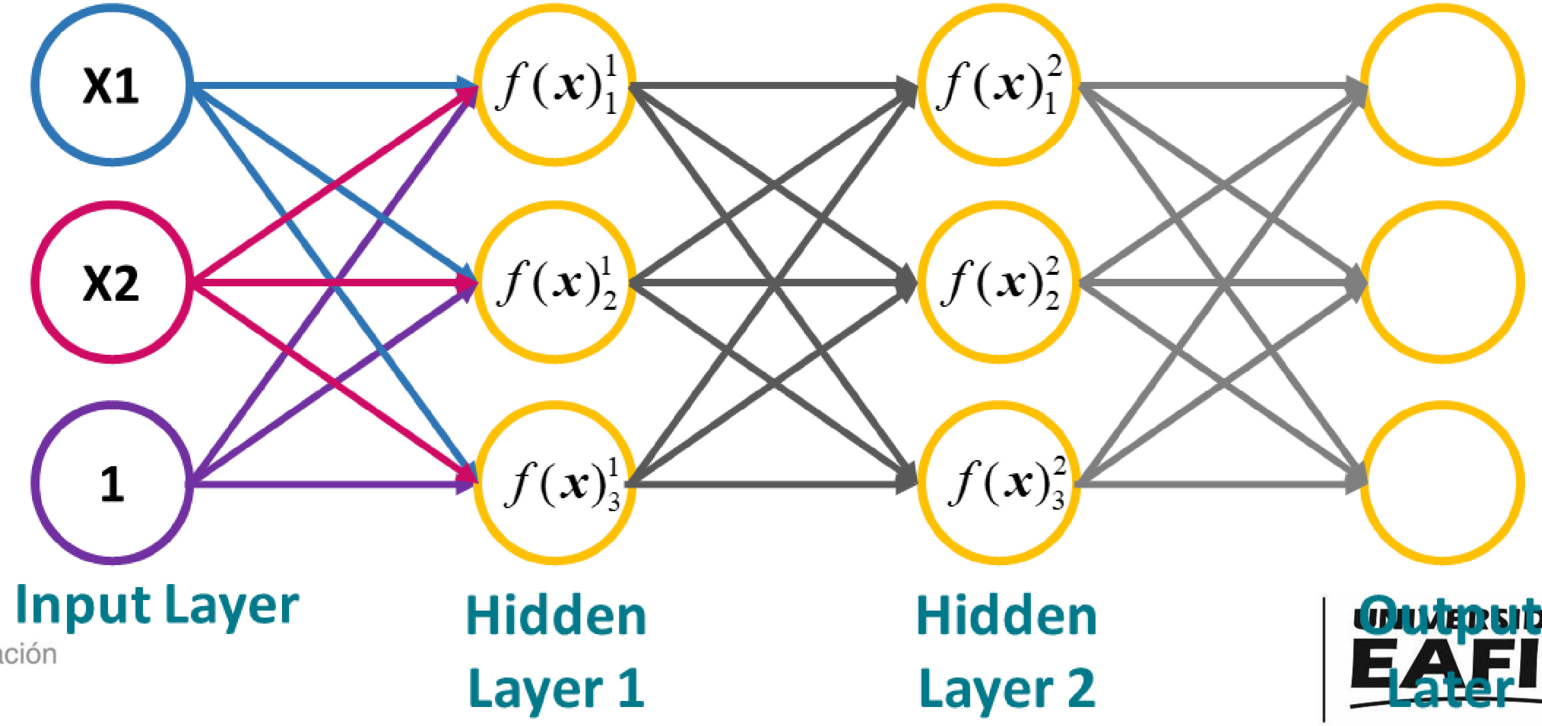
Backpropagation



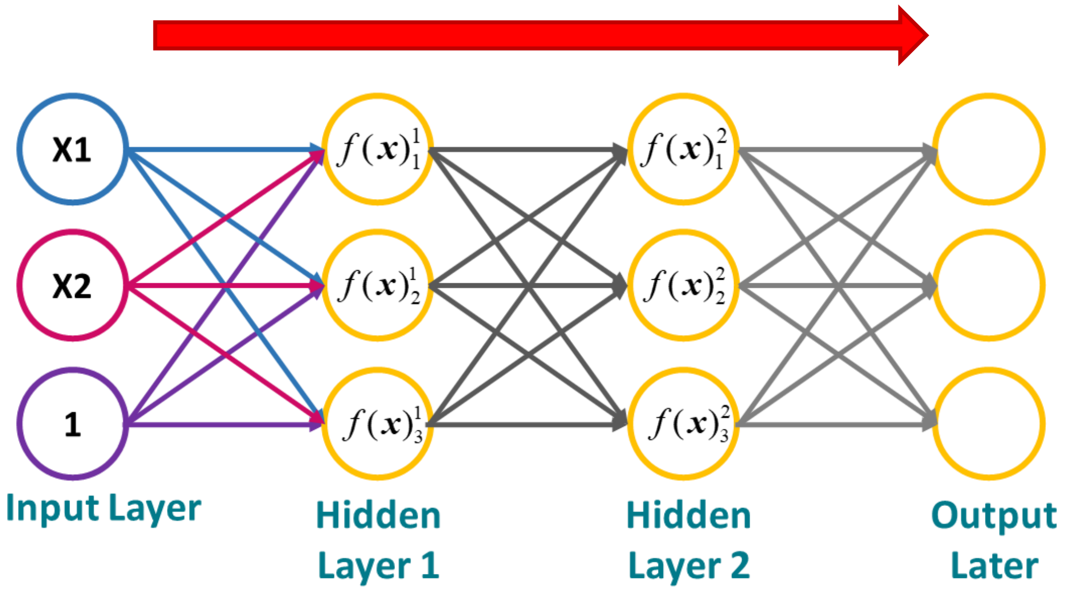
$$f(\boldsymbol{x})_1 = \sigma_1\left[w_{1,1}x_1 + w_{2,1}x_2 + b_1\right]$$

$$f(\boldsymbol{x})_2 = \sigma_2\left[w_{1,2}x_1 + w_{2,2}x_2 + b_2\right]$$

$$f(\boldsymbol{x})_3 = \sigma_3\left[w_{1,3}x_1 + w_{2,3}x_2 + b_3\right]$$

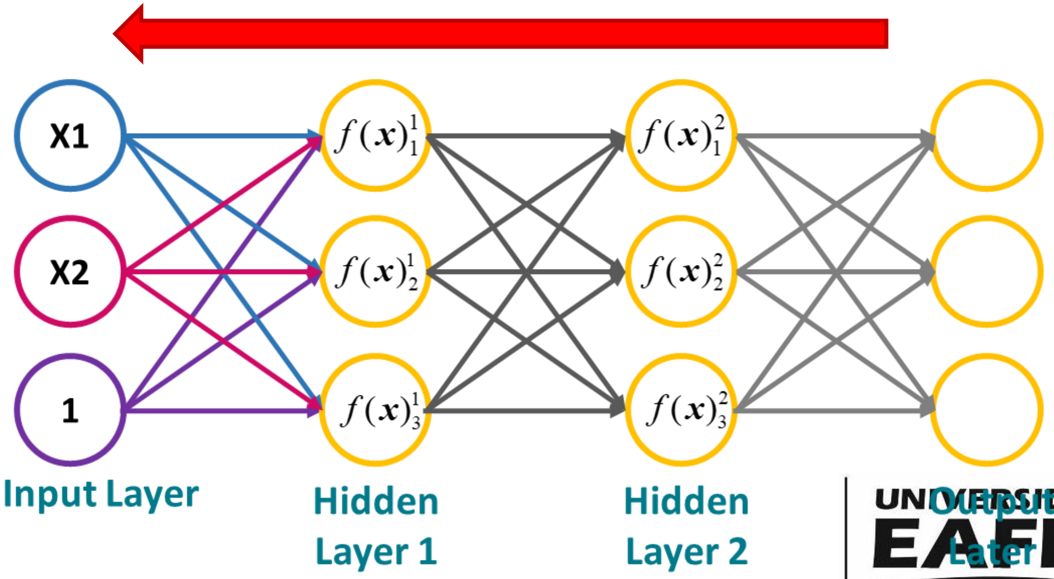


Backpropagation



Forward propagation es el proceso de pasar los datos de entrada a través de la red neuronal para calcular la salida prevista. Cada capa de la red realiza una suma ponderada de sus entradas, aplica una función de activación y pasa el resultado a la siguiente capa.

Backward propagation es el proceso de actualización de los pesos y sesgos, en función de la pérdida calculada durante la propagación hacia adelante. Implica calcular los gradientes de la pérdida con respecto a los parámetros del modelo. Estos gradientes se utilizan para actualizar los parámetros mediante un algoritmo de optimización, como el descenso de gradientes.

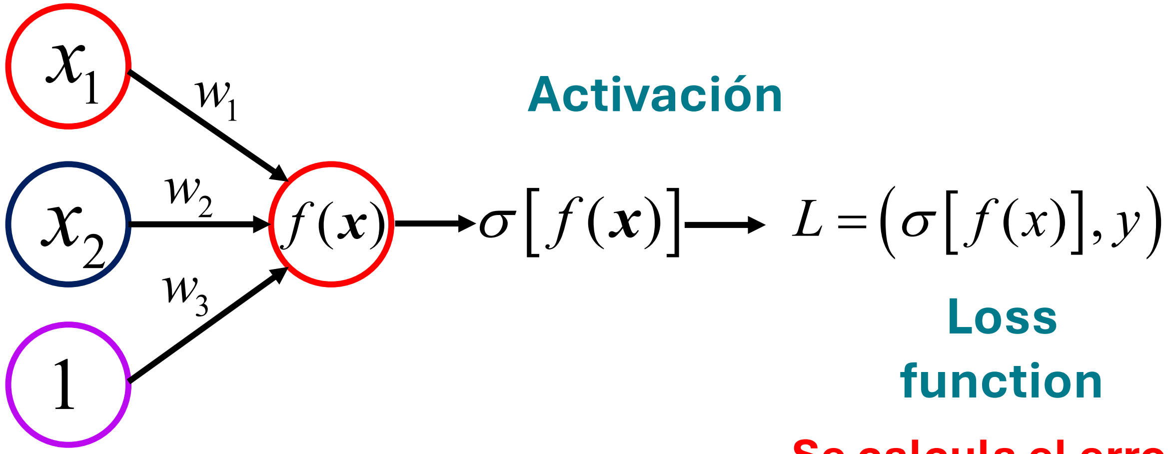


Backpropagation

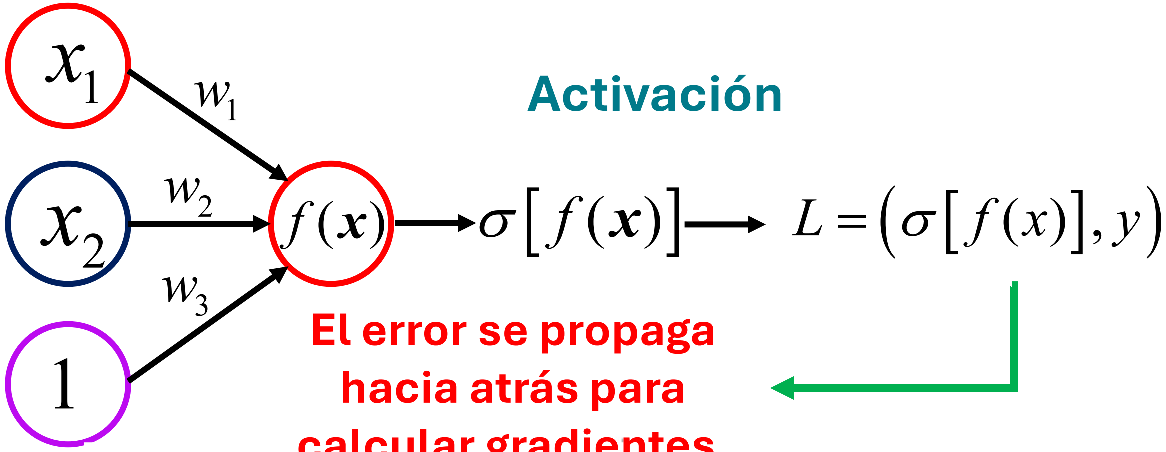
El entrenamiento de una NN se realiza en dos fases:

Forward propagation:

- ➢ La señal de entrada se propaga por toda la red, capa por capa.
- ➢ Se realizan transformaciones lineales y no lineales.
- ➢ Se calculan las activaciones neuronales.
- ➢ Se obtiene la predicción del modelo.
- ➢ Genera los valores de salida de la red.



Se calcula el error entre la salida del modelo y el valor real.



El error se propaga hacia atrás para calcular gradientes.

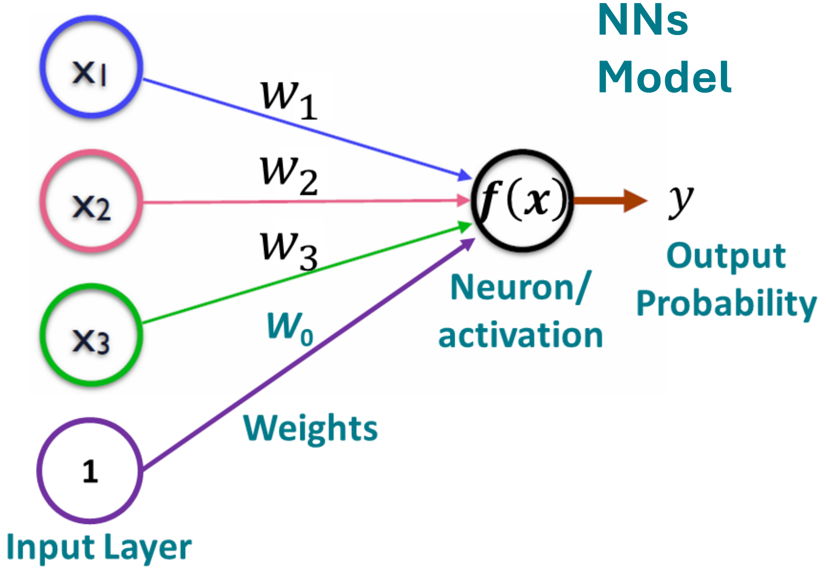
Backward propagation:

- ➢ El error se propaga desde la capa de salida hacia la capa de entrada mediante la regla de la cadena.
- ➢ Se calculan gradientes.
- ➢ Se obtiene información para ajustar los pesos.

UNIVERSIDAD EAFIT®

Backpropagation

NNs Model



Predictions

y → Real value

$\hat{y}$ → Model prediction

Loss function

Mean-square error (MSE) – Used for numerical predictions / continues data.

$$L = \frac{1}{2}\left(y-\hat{y}\right)^2$$

Logistic (Cross-entropy) – Used for probability / categorical data.

$$L = -\left(y\log(\hat{y}) + (1-y)(1-\hat{y})\right)$$

Cost function

Find the weight that minimizes the losses (Loss function)

$$J = \frac{1}{n}\sum_{i=1}^{n}\wp\left(\hat{y}_i, y_i\right) \rightarrow$$

$$W = \arg\min J = \frac{1}{n}\sum_{i=1}^{n}L\left(\hat{y}_i, y_i\right)$$

$$W = \arg\min J = \frac{1}{n}\sum_{i=1}^{n}L\left(f(x,W)_i, y_i\right)$$

Loss quantifies error; Cost defines the optimization problem

Backpropagation

Backpropagation

input:
example $(\mathbf{x}, \mathbf{y})$, weight vector $\mathbf{w}$, layered graph (V, E),
activation function $\sigma : \mathbb{R} \to \mathbb{R}$

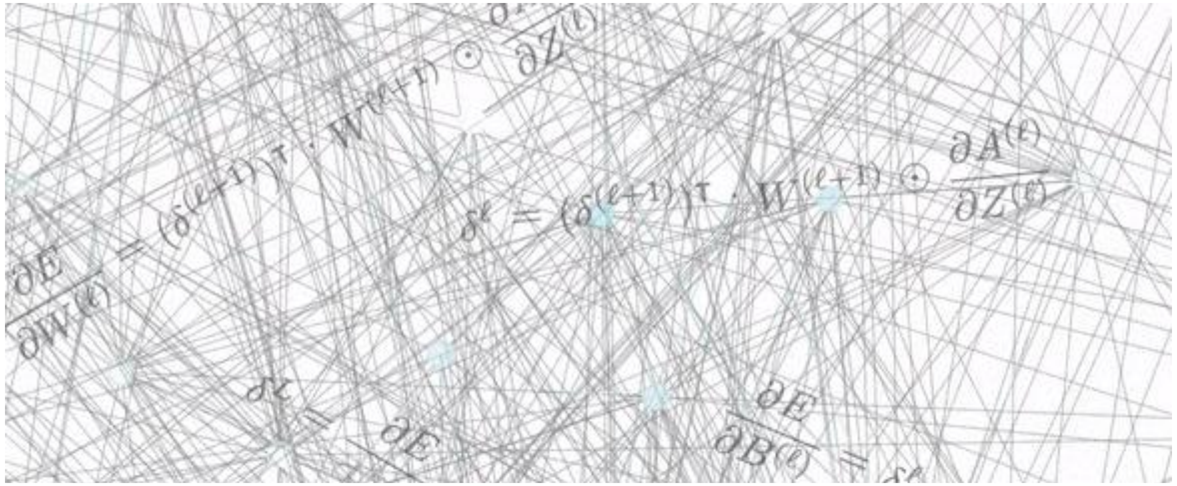
initialize:
denote layers of the graph $V_0, \ldots, V_T$ where $V_t = \{v_{t,1}, \ldots, v_{t,k_t}\}$
define $W_{t,i,j}$ as the weight of $(v_{t,j}, v_{t+1,i})$
(where we set $W_{t,i,j} = 0$ if $(v_{t,j}, v_{t+1,i}) \notin E$)

forward:
set $\mathbf{o}_0 = \mathbf{x}$
for $t = 1, \ldots, T$
for $i = 1, \ldots, k_t$
set $a_{t,i} = \sum_{j=1}^{k_{t-1}} W_{t-1,i,j}\, o_{t-1,j}$
set $o_{t,i} = \sigma(a_{t,i})$

backward:
set $\boldsymbol{\delta}_T = \mathbf{o}_T - \mathbf{y}$
for $t = T-1, T-2, \ldots, 1$
for $i = 1, \ldots, k_t$
$\delta_{t,i} = \sum_{j=1}^{k_{t+1}} W_{t,j,i}\, \delta_{t+1,j}\, \sigma'(a_{t+1,j})$

output:
foreach edge $(v_{t-1,j}, v_{t,i}) \in E$
set the partial derivative to $\delta_{t,i}\, \sigma'(a_{t,i})\, o_{t-1,j}$

[Link]



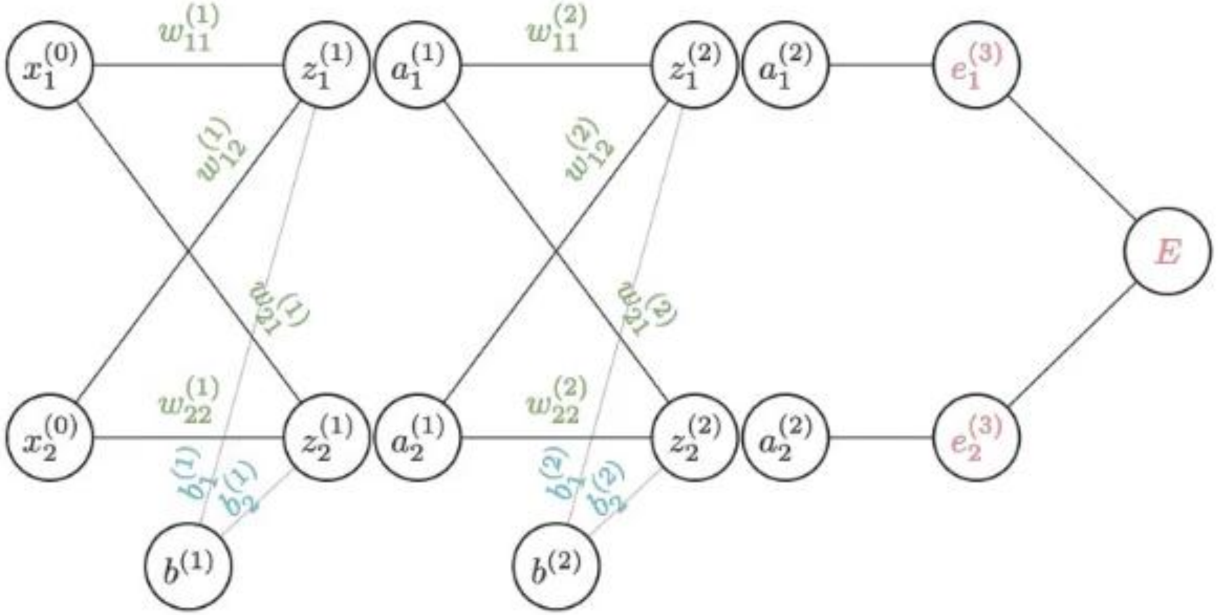
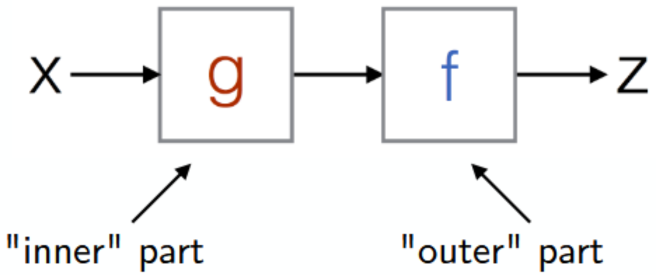


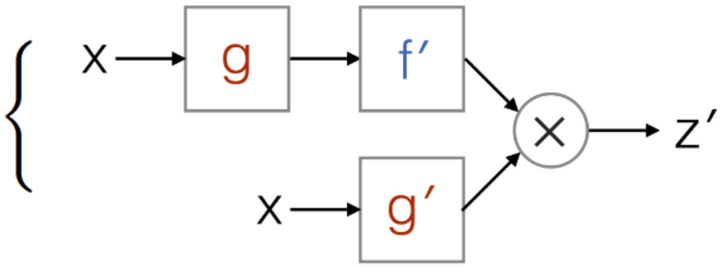
Figure 1: The example neural network setup (Image by Author)

Regla de la cadena

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



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



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

- 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

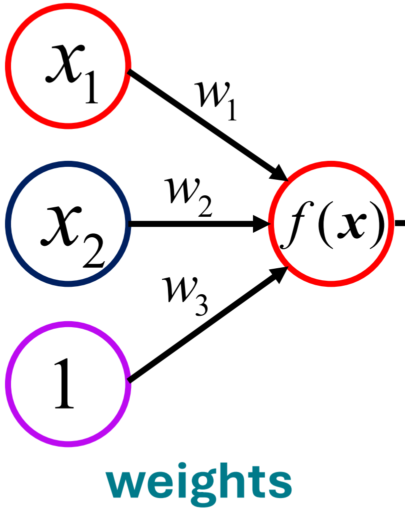
$$\frac{d(\text{Dólares})}{d(\text{Pesos})} = \frac{d(\text{Dólares})}{d(\text{Euros})}\cdot\frac{d(\text{Euros})}{d(\text{Pesos})}$$

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

Inspira Crea Transforma | Vigilada Mineducación

UNIVERSIDAD EAFIT

Backpropagation



$$\rightarrow \sigma[f(\boldsymbol{x})] \rightarrow L = (\sigma[f(x)], y)$$

Activation

Loss function

$$w_{n+1} = w - \alpha \frac{\partial L}{\partial w}$$

$$\mathrm{w} \rightarrow \mathrm{z} \rightarrow \sigma \rightarrow \mathrm{L}$$

$$\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_\mathrm{n} \end{bmatrix} \rightarrow z = w_1 x_1 + w_1 x_1 + \ldots + b \rightarrow \hat{y} = \sigma(z) \rightarrow L = (\hat{y}, y)$$

$$L = (\sigma[f(x)], \mathrm{y})$$

Backpropagation

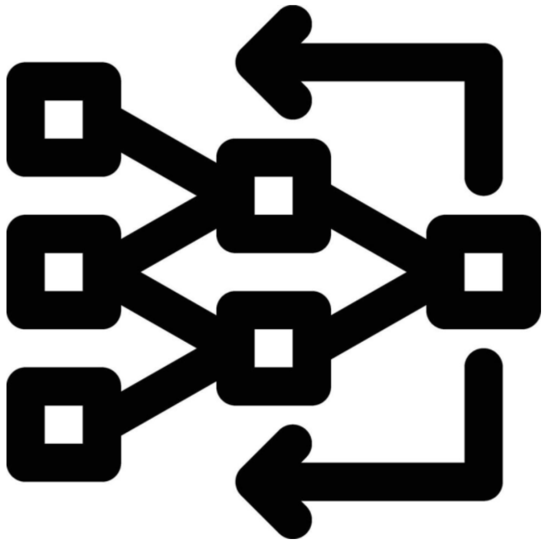
"Si cambio un poquito la predicción... ¿cuánto cambia el error?"

"Sensibilidad de la pérdida respecto a la preactivación z"

$$\frac{\partial L}{\partial \hat{y}} \rightarrow \frac{\partial L}{\partial \sigma} \rightarrow \frac{\partial L}{\partial z} \rightarrow \frac{\partial L}{\partial w}$$

"Sensibilidad de la pérdida respecto a la activación"

"Sensibilidad de la pérdida respecto al parámetro w"

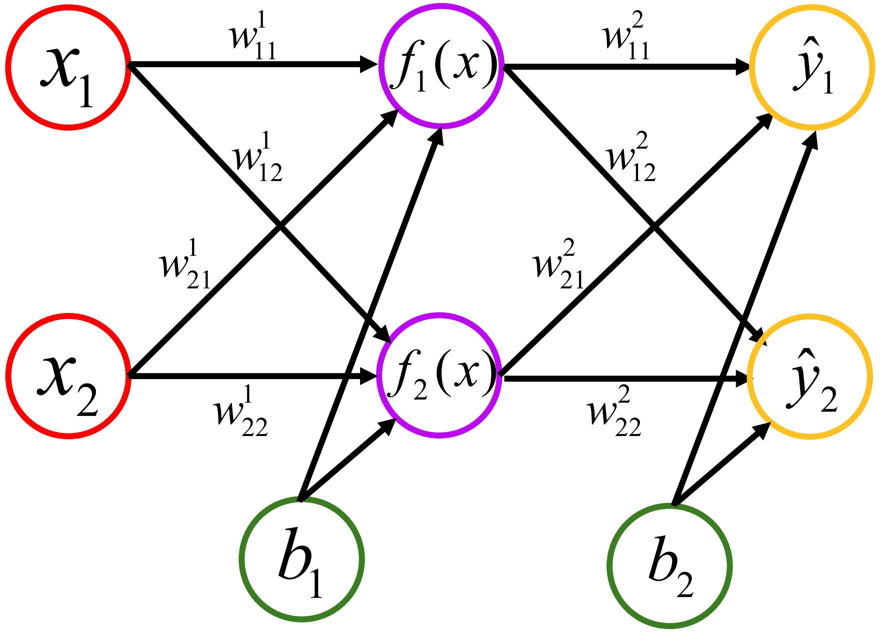


$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w}$$

Inspira Crea Transforma | Vigilada Mineducación

UNIVERSIDAD EAFIT

Backpropagation



Supongamos sigmoidea como función de activación y una función de pérdida MSE.

$$f(\boldsymbol{x}) = \frac{1}{1+e^{-x}} \qquad L = \frac{1}{2}(y-\hat{y})^2$$

$$\frac{df(\boldsymbol{x})}{dx} = \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] \rightarrow \frac{df(\boldsymbol{x})}{dx} = \frac{\frac{d(1)}{dx}\left(1+e^{-x}\right) - \frac{d\left(1+e^{-x}\right)}{dx}(1)}{\left(1+e^{-x}\right)^2} \rightarrow \frac{df(\boldsymbol{x})}{dx} = \frac{e^{-x}}{\left(1+e^{-x}\right)^2}$$

Expresémoslo alternativamente por razones de esfuerzo computacional

$$\frac{df(\boldsymbol{x})}{dx} = \frac{1-1+e^{-x}}{\left(1+e^{-x}\right)^2} \rightarrow \frac{df(\boldsymbol{x})}{dx} = \frac{1+e^{-x}}{\left(1+e^{-x}\right)^2} - \frac{1}{\left(1+e^{-x}\right)^2} \rightarrow \frac{df(\boldsymbol{x})}{dx} = \frac{1}{\left(1+e^{-x}\right)} - \frac{1}{\left(1+e^{-x}\right)^2}$$

$$\frac{d}{dx}\left[\frac{g(x)}{f(x)}\right] = \frac{g'(x)f(x) - g(x)f'(x)}{f(x)^2}$$

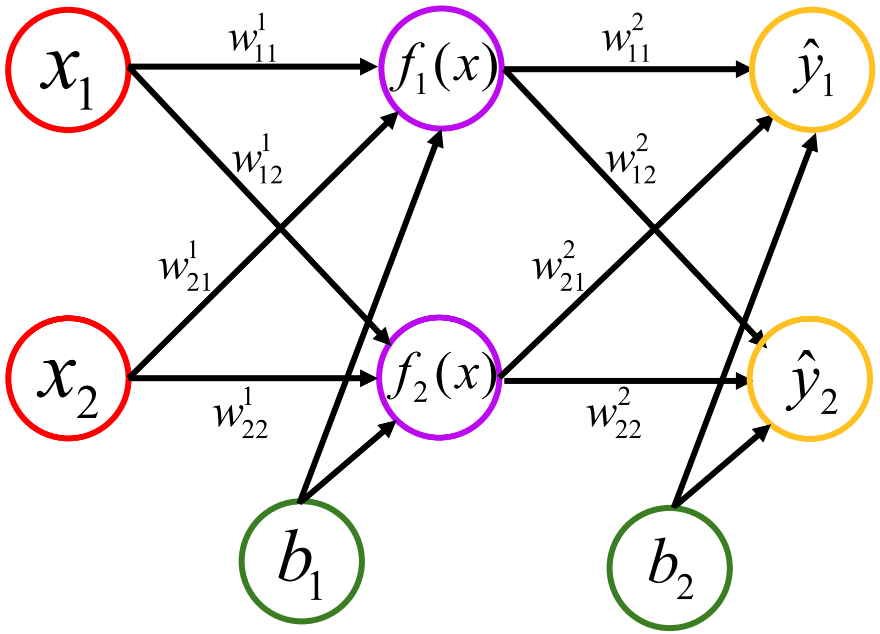
$$\frac{df(\boldsymbol{x})}{dx} = \frac{1}{\left(1+e^{-x}\right)}\left[1 - \frac{1}{\left(1+e^{-x}\right)}\right] \longrightarrow \boxed{f(\boldsymbol{x})\left[1-f(\boldsymbol{x})\right]}$$

IMPORTANTE

$$w_{n+1} = w - \alpha\frac{\partial L}{\partial w}$$

UNIVERSIDAD EAFIT

Backpropagation



$$L = \frac{1}{2}(y-\hat{y})^2 \rightarrow \frac{\partial L}{\partial \hat{y}} = \frac{1}{2}\frac{d(y-\hat{y})^2}{d\hat{y}}$$

$$\frac{1}{2}\frac{d(y-\hat{y})^2}{d\hat{y}} = \frac{1}{2}(-2(y-\hat{y}))$$

$$\frac{\partial L}{\partial \hat{y}} = \boxed{(\hat{y}-y)}$$

Inputs / Outputs / Learning Rate

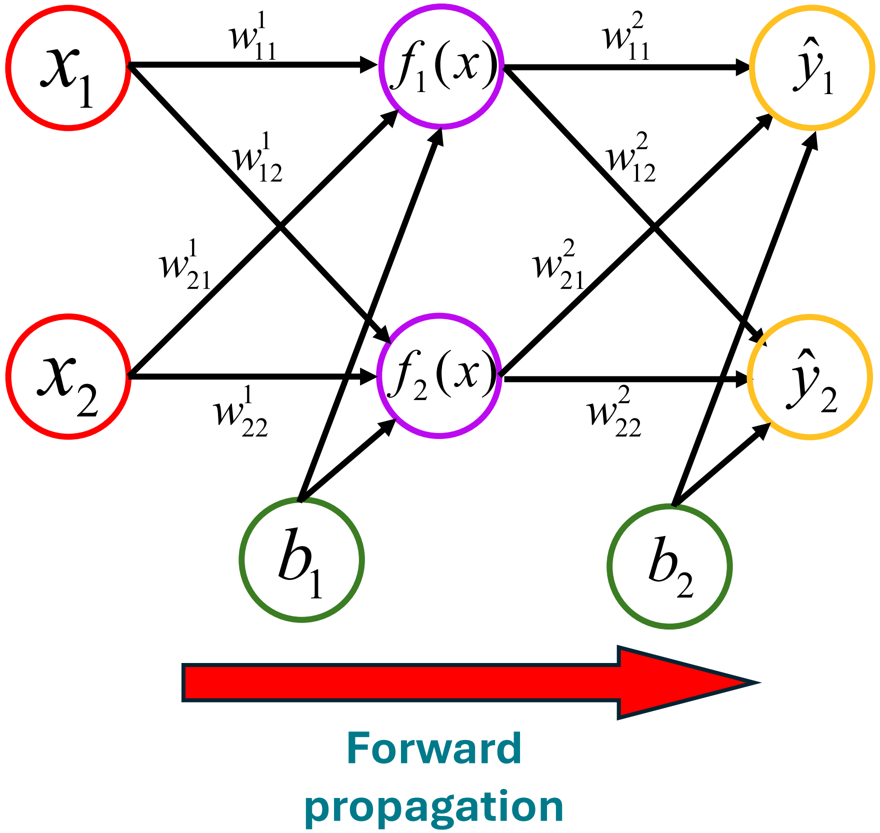
$x_1 = 0.05$	$y_1 = 0.01$	$\alpha = 0.5$
$x_2 = 0.10$	$y_2 = 0.99$	

Initialize values

$w^1_{11} = 0.15$	$w^2_{11} = 0.40$	
$w^1_{12} = 0.25$	$w^2_{12} = 0.50$	$b_1 = 0.35$
$w^1_{21} = 0.20$	$w^2_{21} = 0.45$	$b_2 = 0.60$
$w^1_{22} = 0.30$	$w^2_{22} = 0.55$	

UNIVERSIDAD EAFIT

Backpropagation



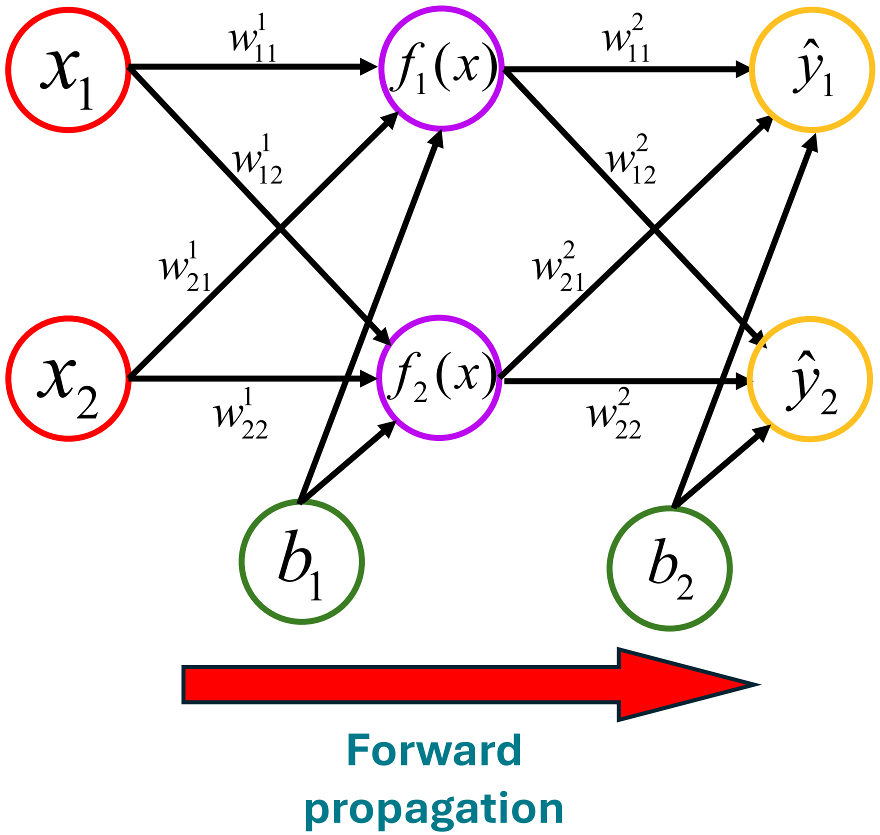
$$f_1(x) = x_1 w_{11}^1 + x_2 w_{21}^1 + b_1$$

$$f_1(x) = (0.05)(0.15) + (0.10)(0.20) + 0.35 \rightarrow f_1(x) = 0.3775$$

$$\sigma[f_1(x)] = \frac{1}{1+e^{-f_1(x)}} \rightarrow \sigma[f_1(x)] = \frac{1}{1+e^{-0.3775}}$$

$$\sigma[f_1(x)] = 0.59326999211$$

Backpropagation



$$f_1(x) = x_1 w_{11}^1 + x_2 w_{21}^1 + b_1$$

$$f_1(x) = (0.05)(0.15) + (0.10)(0.20) + 0.35 \rightarrow f_1(x) = 0.3775$$

$$\sigma[f_1(x)] = \frac{1}{1+e^{-f_1(x)}} \rightarrow \sigma[f_1(x)] = \frac{1}{1+e^{-0.3775}}$$

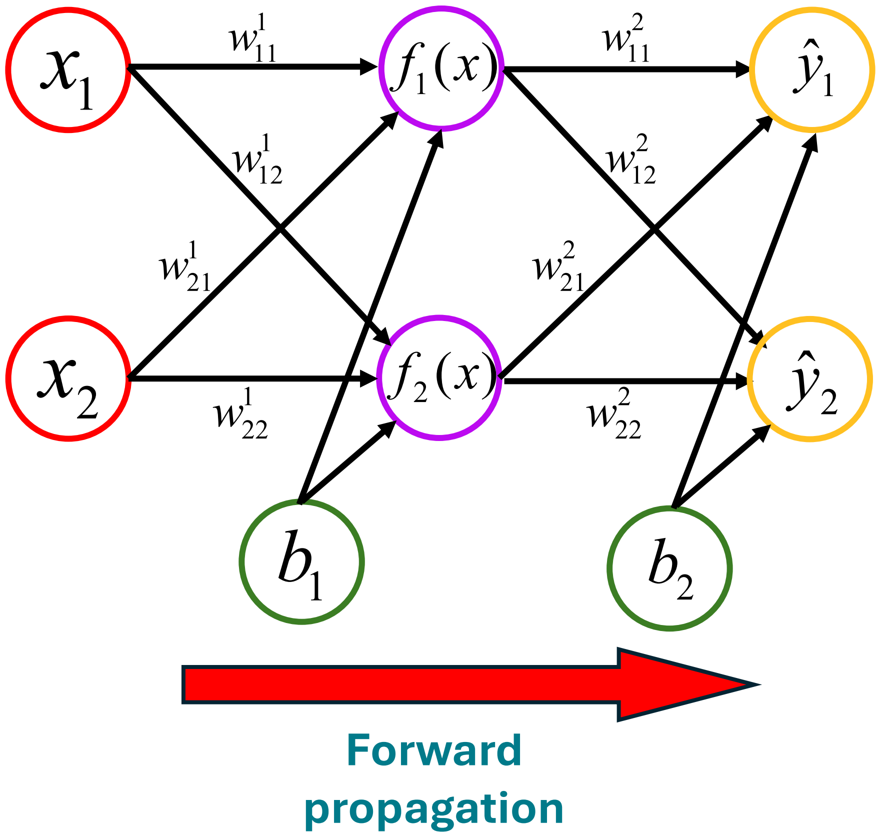
$$\sigma[f_1(x)] = 0.59326999211$$

$$f_2(x) = x_1 w_{12}^1 + x_2 w_{22}^1 + b_1$$

$$f_2(x) = (0.05)(0.25) + (0.10)(0.30) + 0.35 \rightarrow f_2(x) = 0.3925$$

$$\sigma[f_2(x)] = \frac{1}{1+e^{-f_2(x)}} \rightarrow \sigma[f_2(x)] = \frac{1}{1+e^{-0.3925}}$$

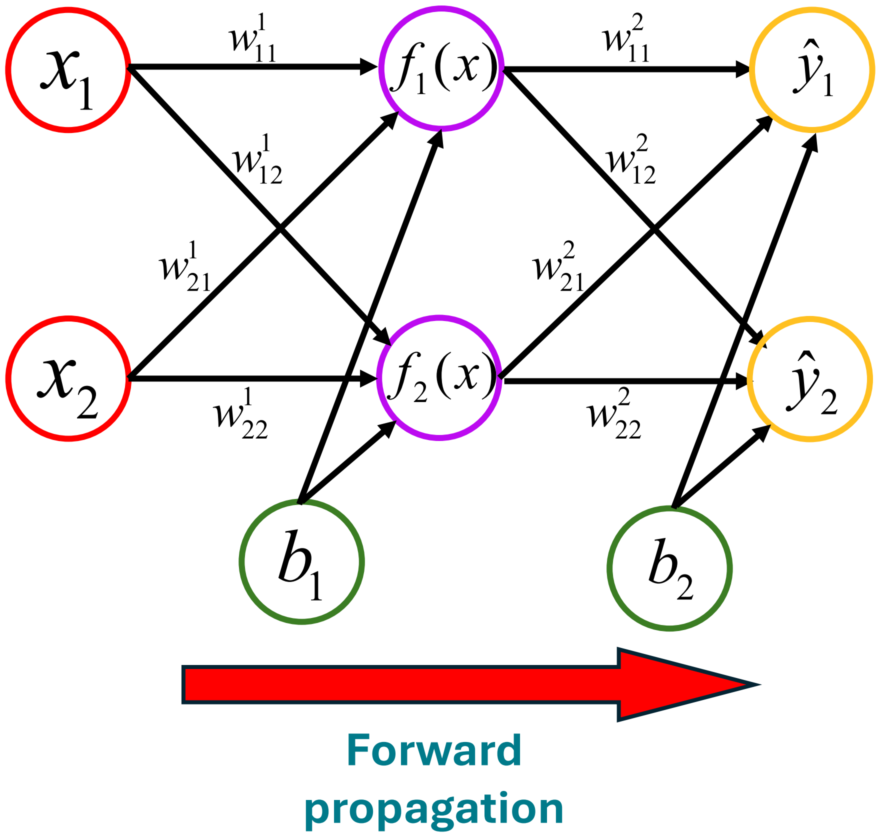
$$\sigma[f_2(x)] = 0.59688437826$$



$$\sigma[f_1(x)] = 0.59326999211$$

$$\sigma[f_2(x)] = 0.59688437826$$

Backpropagation



$$\sigma[f_1(x)] = 0.59326999211$$

$$\sigma[f_2(x)] = 0.59688437826$$

$$\hat{y}_1 = \sigma[f_1(x)]w_{11}^2 + \sigma[f_2(x)]w_{21}^2 + b_2$$

$$\hat{y}_1 = (0.5932)(0.40) + (0.5968)(0.45) + 0.60$$

$$\hat{y}_1 = 1.10590596706$$

$$\sigma[\hat{y}_1] = \frac{1}{1+e^{-\hat{y}_1}} \rightarrow \sigma[\hat{y}_1] = \frac{1}{1+e^{-1.1059}} \rightarrow \sigma[\hat{y}_1] = 0.75136506955$$

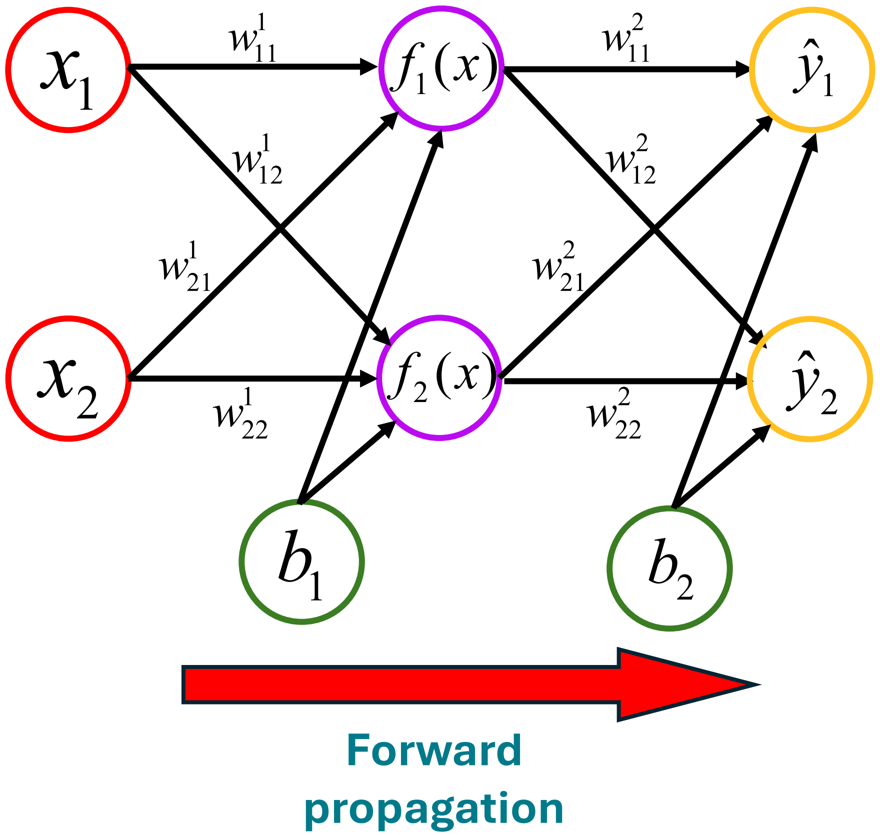
$$\hat{y}_2 = \sigma[f_1(x)]w_{12}^2 + \sigma[f_2(x)]w_{22}^2 + b_2$$

$$\hat{y}_2 = (0.5932)(0.50) + (0.5968)(0.55) + 0.60$$

$$\hat{y}_2 = 1.22492140407$$

$$\sigma[\hat{y}_2] = \frac{1}{1+e^{-\hat{y}_2}} \rightarrow \sigma[\hat{y}_2] = \frac{1}{1+e^{-1.2249}} \rightarrow \sigma[\hat{y}_2] = 0.77292846532$$

Backpropagation



$$E_T = \sum_{i=1}^{n} E_i = \frac{1}{2}\left(y_1 - \hat{y}_1\right)^2 + \frac{1}{2}\left(y_2 - \hat{y}_2\right)^2$$

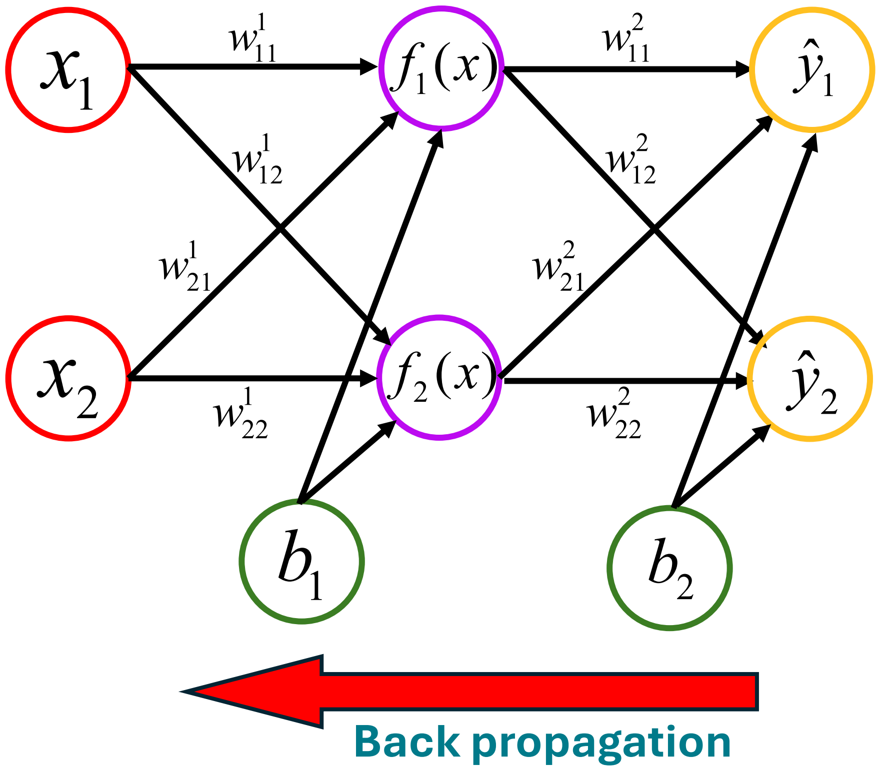
$$E_T = \frac{1}{2}(0.01 - 0.7513)^2 + \frac{1}{2}(0.99 - 0.7729)^2 = 0.29837110876$$

Aunque la salida para el segundo dato predicha es "similar", para el primer dato no lo es.. El modelo presenta un error significativo respecto a los valores objetivo.

$$\sigma[\hat{y}_1] = 0.75136506955 \qquad \hat{y}_1$$

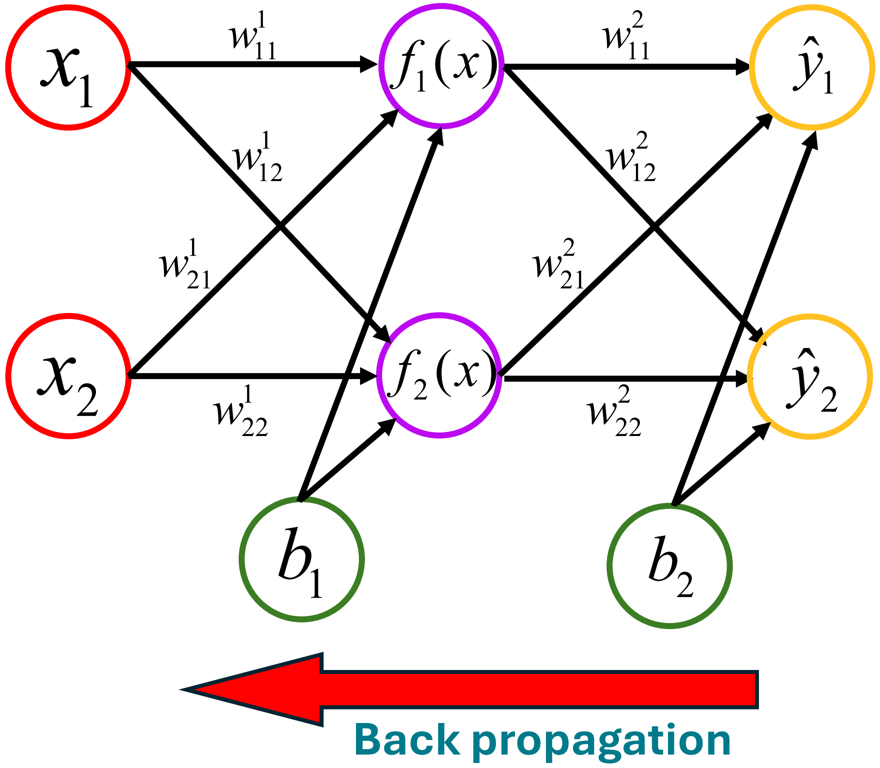
$$\sigma[\hat{y}_2] = 0.77292846532 \qquad \hat{y}_2$$

Backpropagation



$$E_T = \frac{1}{2}(0.01 - 0.7573)^2 + \frac{1}{2}(0.99 - 0.7729)^2 = 0.29837110876$$

Backpropagation



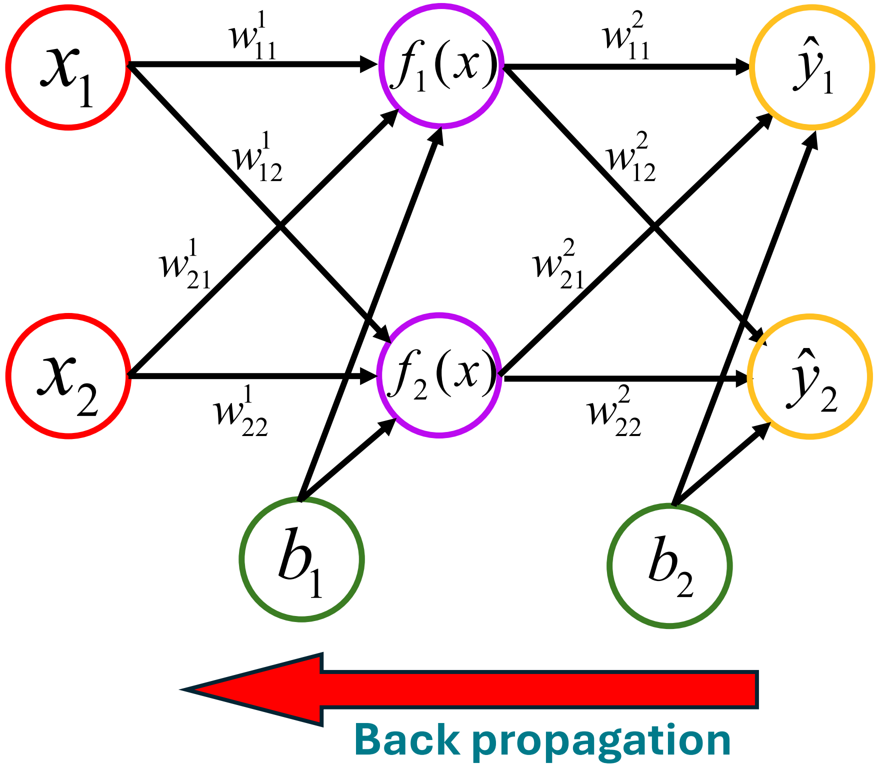
$$E_T = \frac{1}{2}(0.01 - 0.7573)^2 + \frac{1}{2}(0.99 - 0.7729)^2 = 0.29837110876$$

Backpropagation calcula gradientes para cada parámetro

$$\left[w^2_{11}, w^2_{12}, w^2_{21}, w^2_{22}\right] \qquad \left[w^1_{11}, w^1_{12}, w^1_{21}, w^1_{22}\right]$$

$$\boxed{w_{n+1} \leftarrow w - \alpha \frac{\partial E}{\partial w} \qquad \frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w}}$$

Backpropagation



$$\frac{\partial L}{\partial \mathrm{w}} = \frac{\partial L}{\partial \hat{\mathrm{y}}}\frac{\partial \hat{\mathrm{y}}}{\partial \mathrm{z}}\frac{\partial \mathrm{z}}{\partial \mathrm{w}}$$

$$\left[w^2_{11}, w^2_{12}, w^2_{21}, w^2_{22}\right]$$

$$w^{2'}_{11} = w^2_{11} - (0.5)\frac{\partial L}{\partial w^2_{11}}$$

$$\frac{\partial L}{\partial \mathrm{w}^2_{11}} = \frac{\partial L}{\partial \hat{\mathrm{y}}_1}\frac{\partial \hat{\mathrm{y}}_1}{\partial \mathrm{z}_1}\frac{\partial \mathrm{z}_1}{\partial \mathrm{w}^2_{11}} \rightarrow \frac{\partial L}{\partial \mathrm{w}^2_{11}} = (\hat{\mathrm{y}}_1 - y)(\sigma(\mathrm{z}_1)[1 - \sigma(\mathrm{z}_1)])p$$

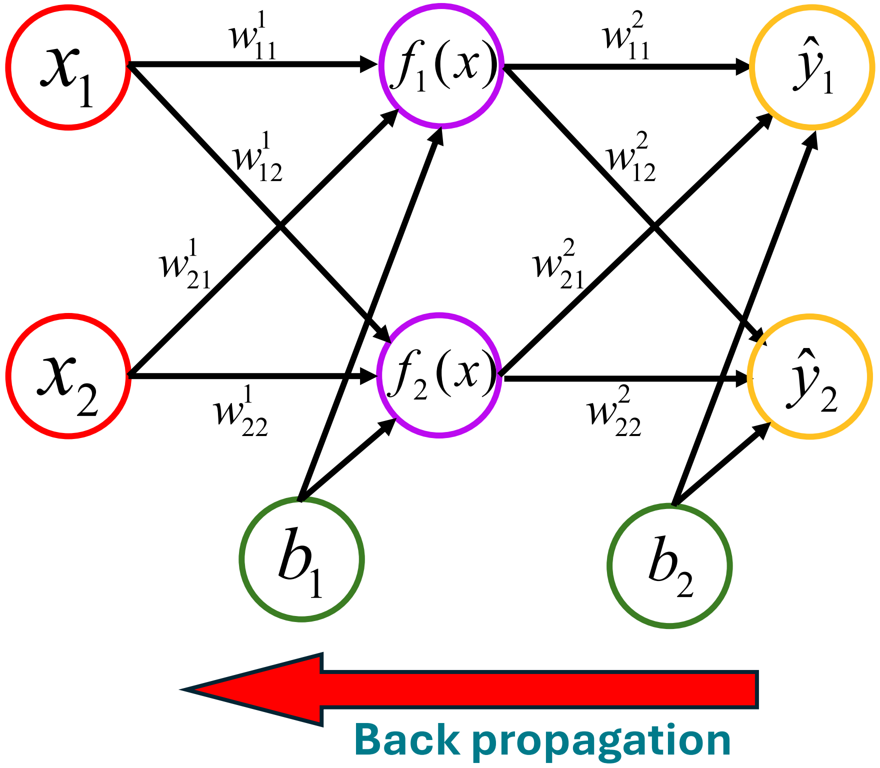
$$\frac{\partial L}{\partial \hat{\mathrm{y}}_1} = (\hat{\mathrm{y}}_1 - y)$$

$$\frac{\partial \hat{\mathrm{y}}_1}{\partial \mathrm{z}_1} = \sigma(\mathrm{z}_1)[1 - \sigma(\mathrm{z}_1)]$$

$$\frac{\partial \mathrm{z}_1}{\partial \mathrm{w}^2_{11}} = \frac{\partial}{\partial \mathrm{w}^2_{11}}[\sigma[f_1(x)]\mathrm{w}^2_{11} + \sigma[f_2(x)]\mathrm{w}^2_{21} + \mathrm{b}_2]$$

$$[\sigma[f_1(x)]$$

Backprogation



$$\frac{\partial L}{\partial \mathrm{w}} = \frac{\partial L}{\partial \hat{\mathrm{y}}}\frac{\partial \hat{\mathrm{y}}}{\partial \mathrm{z}}\frac{\partial \mathrm{z}}{\partial \mathrm{w}}$$

$$\left[w_{11}^2, w_{12}^2, w_{21}^2, w_{22}^2\right] \qquad w_{11}^{2'} = w_{11}^2 - (0.5)\frac{\partial L}{\partial w_{11}^2}$$

$$\frac{\partial L}{\partial \mathrm{w}_{11}^2} = \frac{\partial L}{\partial \hat{\mathrm{y}}_1}\frac{\partial \hat{\mathrm{y}}_1}{\partial \mathrm{z}_1}\frac{\partial \mathrm{z}_1}{\partial \mathrm{w}_{11}^2} \rightarrow \frac{\partial L}{\partial \mathrm{w}_{11}^2} = (\hat{\mathrm{y}}_1 - y)(\sigma(\mathrm{z}_1)[1 - \sigma(\mathrm{z}_1)])p$$

$$\frac{\partial L}{\partial \mathrm{w}_{11}^2} = (0.7513 - 0.01)(0.7513)[1 - 0.7513])(0.5932)$$

$$\frac{\partial L}{\partial \mathrm{w}_{11}^2} = (0.7513 - 0.01)(0.7513)[1 - 0.7513])(0.5932)$$

$$\frac{\partial L}{\partial \mathrm{w}_{11}^2} = 0.082167041$$

$$w_{11}^{2'} = 0.40 - (0.5)(0.082167041)$$

$$w_{11}^{2'} = 0.35891$$

$w_{11}^{2'} = 0.35891$ $w_{21}^{2'} = 0.511301270$

$w_{12}^{2'} = 0.408666186$ $w_{22}^{2'} = 0.561370121$

Actividad

Implemente el notebook backpropagation_example.ipynb para realizar las siguientes actividades:

1. Sin modificar el código, determine los valores de los pesos tras una segunda iteración.
Importante: realice este cálculo **sin alterar el código**.

2. Modifique el código para calcular el error después de dos iteraciones.

Sobre y sub entrenamiento

Sub-entrenamiento (Underfitting): El modelo no es lo suficientemente bueno para describir las relaciones entre los datos de entrada **X** y la variable de salida **y.**

- El modelo es muy simple para capturar patrones importantes en los datos.

- El modelo tendrá un rendimiento bajo en los conjuntos de entrenamiento y validación.

Sobreentrenamiento (Overfitting): El modelo memoriza o imita los datos de entrenamiento, y falla generalizando para datos desconocidos (datos de test).

- El modelo es demasiado complejo.
- El modelo aprende patrones de ruido y no de relaciones entre variables.
- Tendrá un rendimiento muy bueno en los datos de entrenamiento pero muy malo en los datos de validación o prueba.

UNIVERSIDAD EAFIT

Buen ajuste: El modelo captura las relaciones generales entre los datos de entrada **X** y la varianble de salida **y**.

- El modelo no es ni muy simple ni muy complejo.
- El modelo descifra las relaciones subyacentes de los datos de entrenamiento y no el ruido.
- El modelo tendrá un buen rendimiento en los conjuntos de entrenamiento, validación y prueba.

Sobre y sub entrenamiento

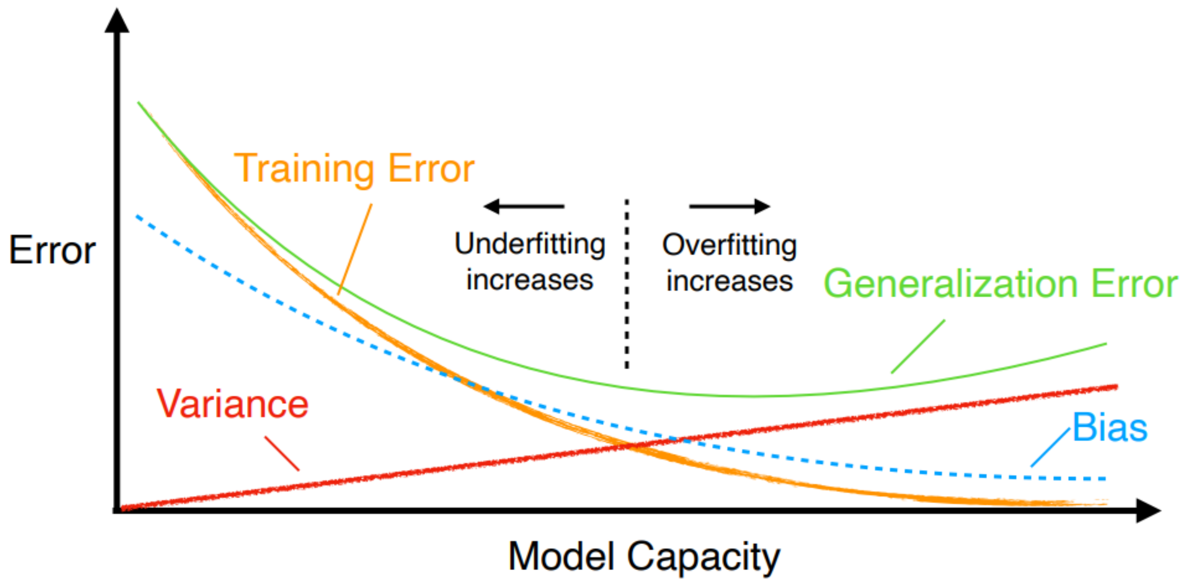
- **Model Capacity: que tan complejos es el modelo**.
  A la izquierda → modelos simples.
  A la derecha → modelos complejos.

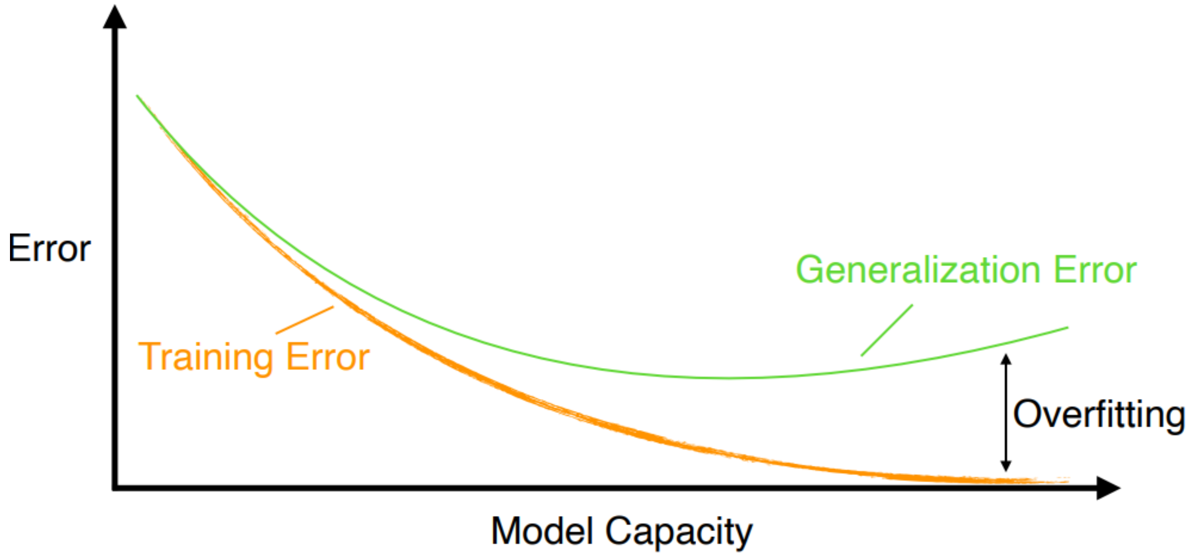
- El error de entrenamiento **siempre disminuye** cuando aumentas la capacidad.

- El Generalization error este es **el error que realmente importa** (test / datos nuevos)..

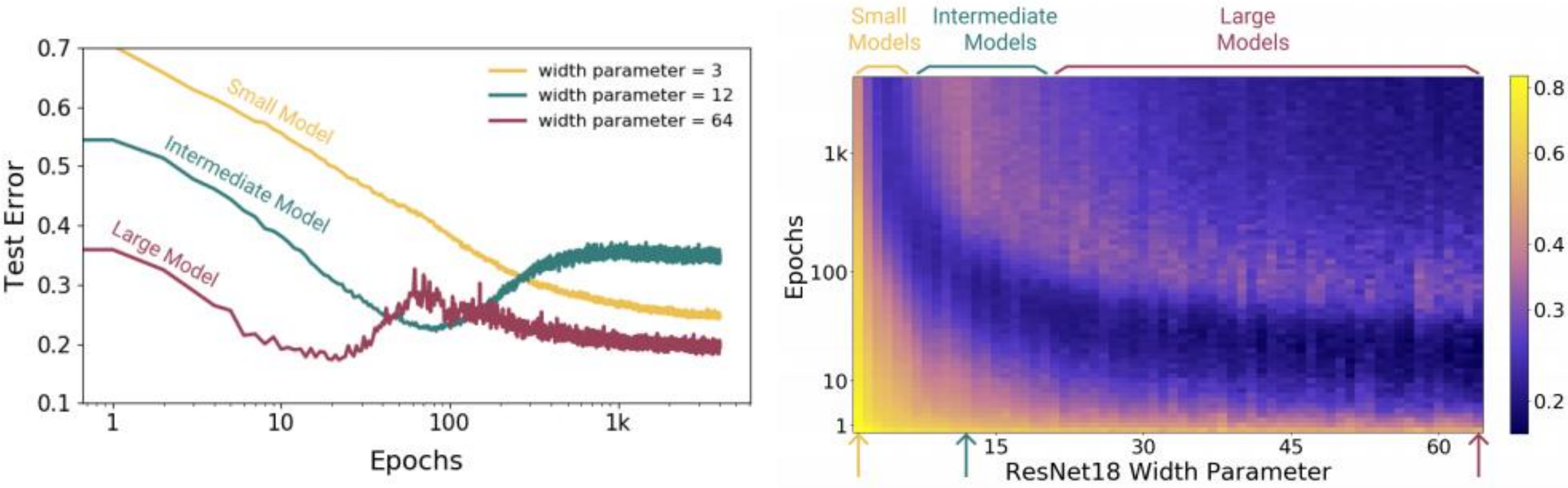
- **Variance** mide sensibilidad al ruido.
  Variance alta = modelo demasiado inestable.
  Modelos simples → variance baja
  Modelos complejos → variance alta
  Modelo complejo = pequeñas variaciones en datos, grandes cambios en predicción.

- Usualmente se usa el error del conjunto de Test como estimador del error de generalización.





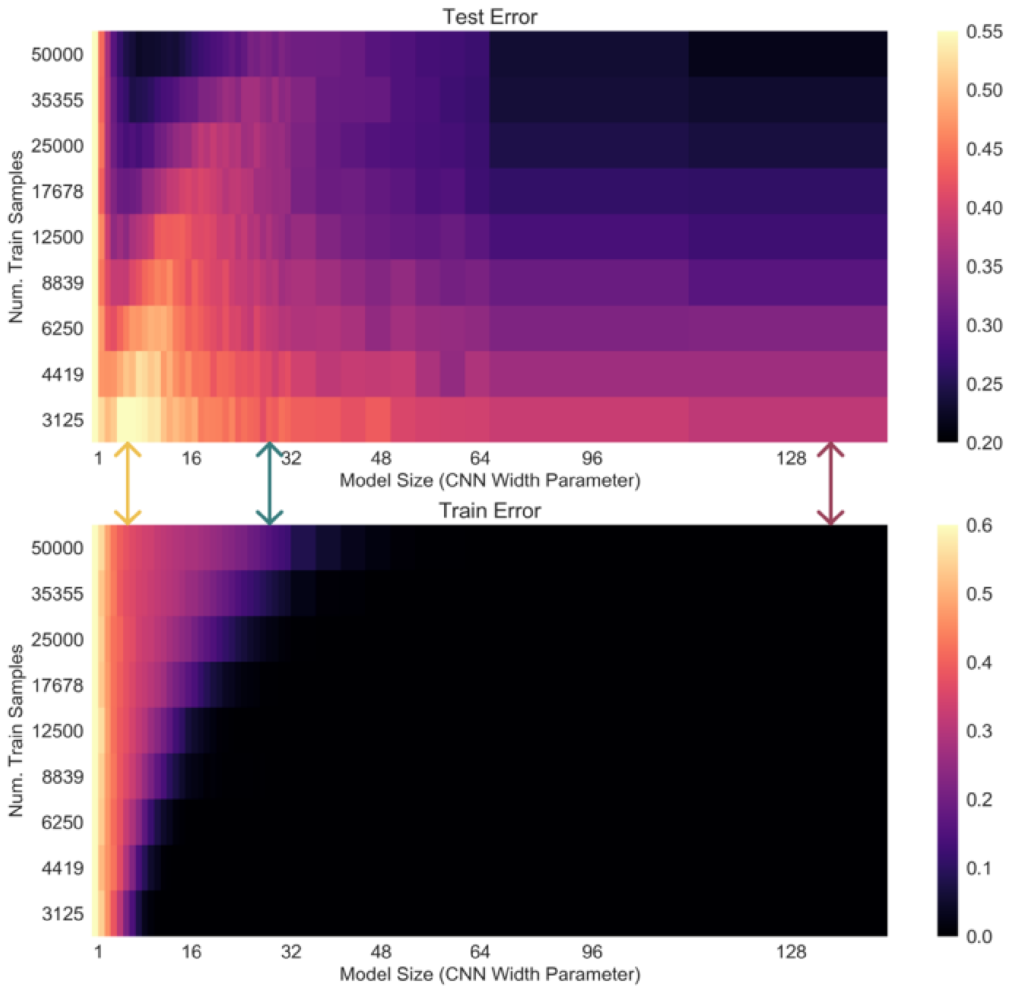
Sobre y sub entrenamiento

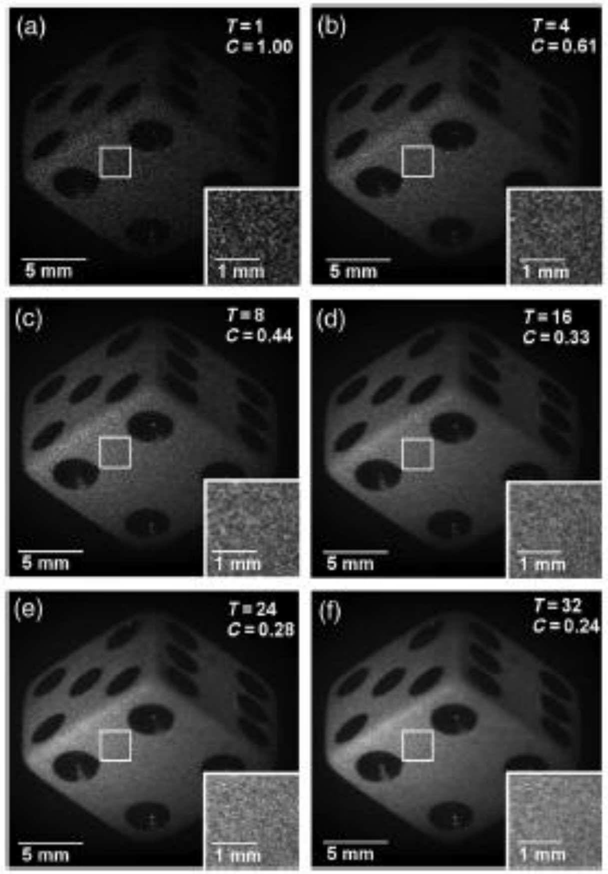


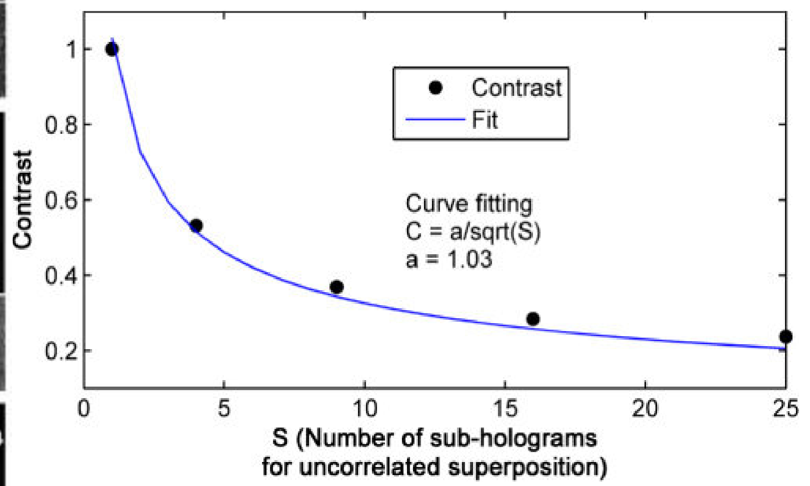
- En la región crítica, solo un modelo se ajusta bien a los datos y es muy sensible al ruido

- Modelos sobre parametrizados: muchos modelos se ajustan bien a los datos, SGD encuentra uno que memoriza los datos de entrenamiento pero que también se desempeña bien en los datos de prueba.

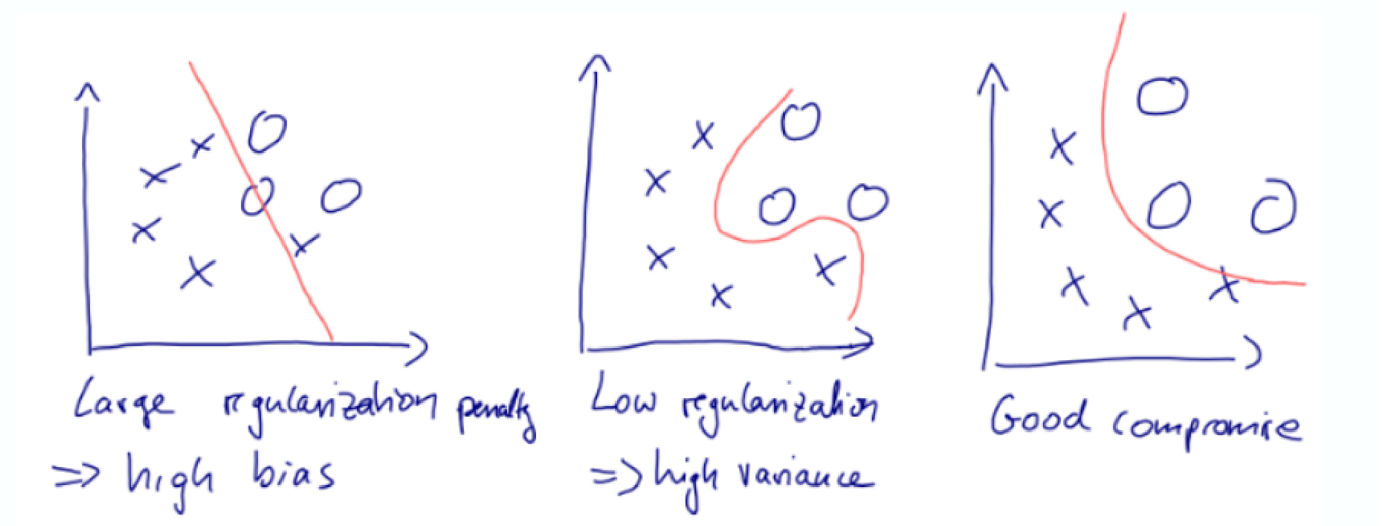
Sobre y sub entrenamiento

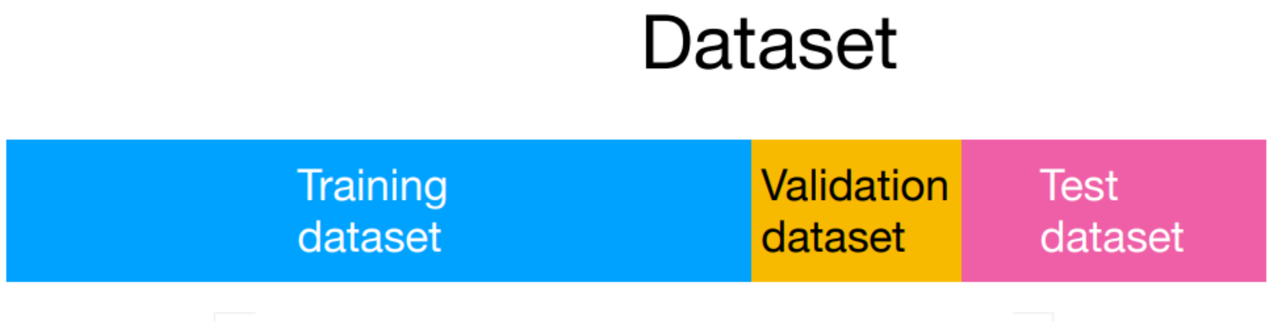






Conjunto de datos



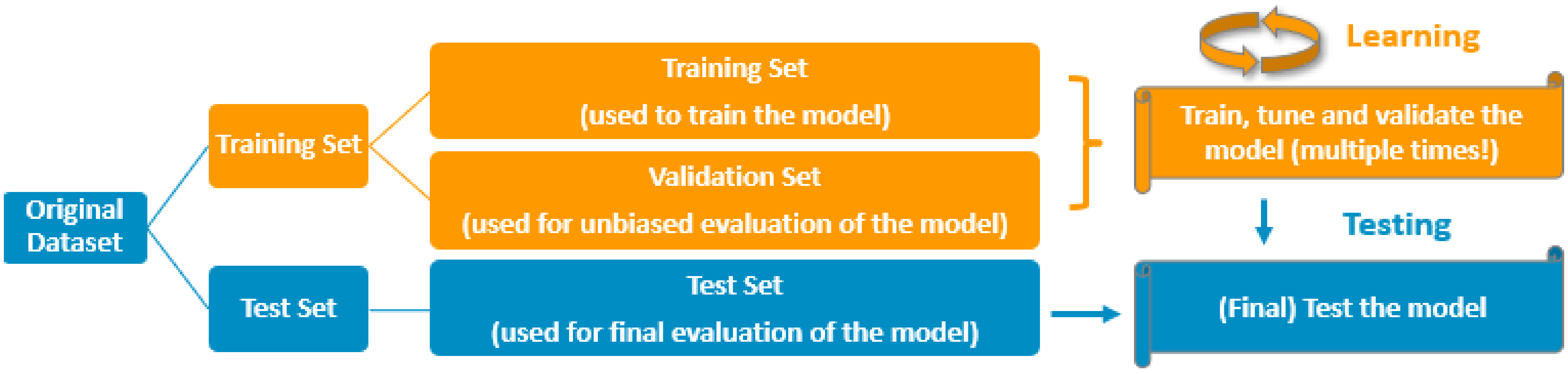


Regla común: 80 / 15 / 5

- Training set → aprendizaje del modelo
- Validation set → estimación del desempeño
- Test set → evaluación final

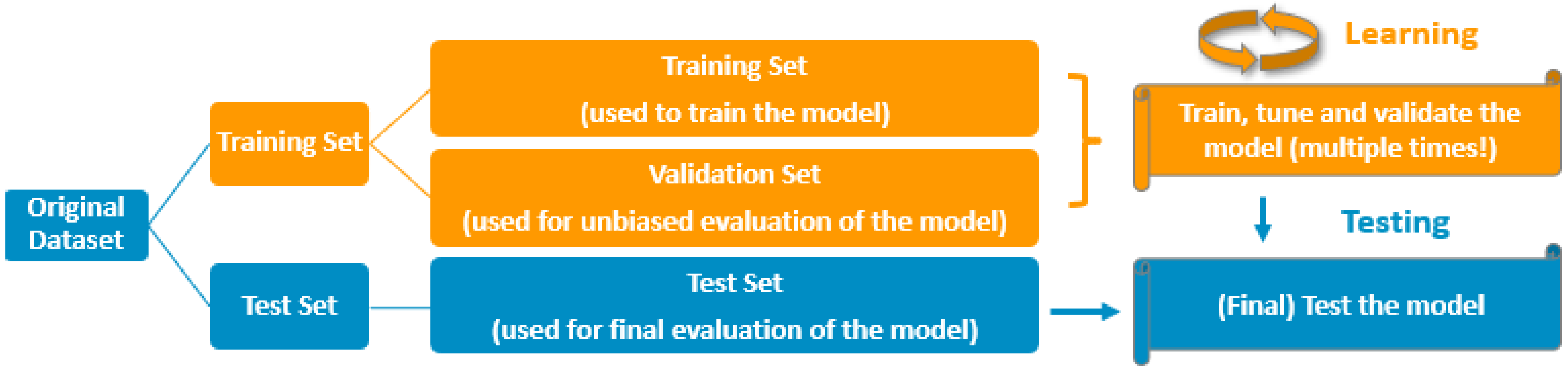
🎯 Test = estimador del error de generalización

Conjunto de datos

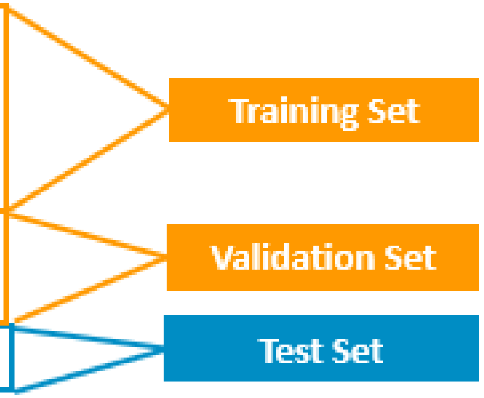


El **conjunto de test no se utiliza para el aprendizaje (entrenamiento),** solo se usa para asegurar que el modelo **generaliza** bien en **nuevos datos "desconocidos".**

Conjunto de datos

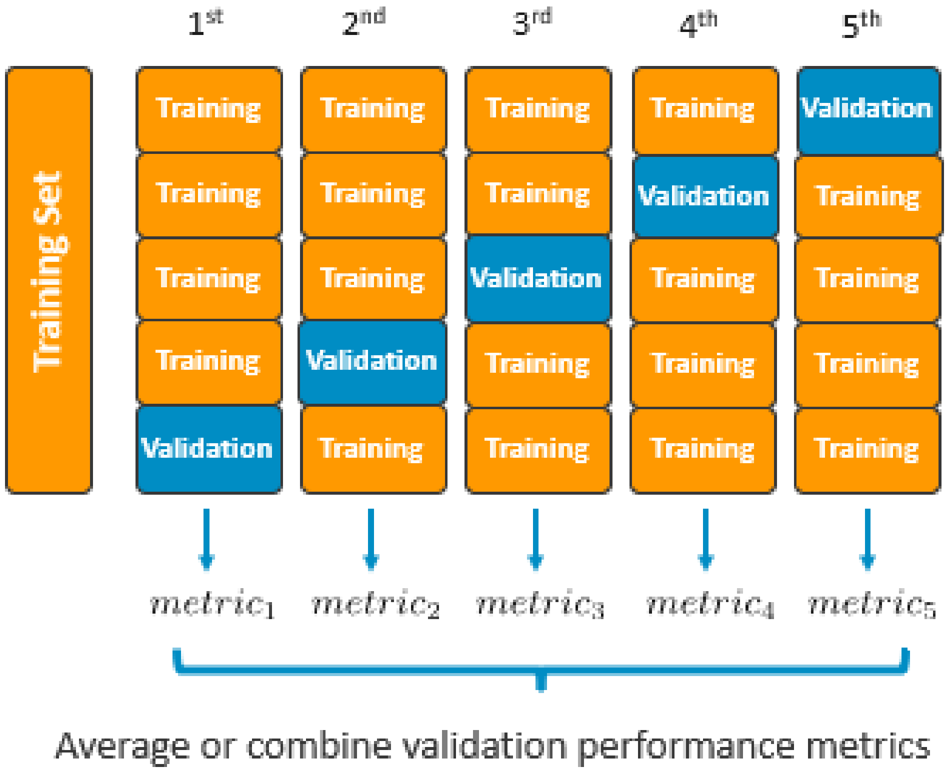


	bad_weather	is_rush_hour	mile_distance	urban_address	late
0	0.0	1.0	5.00	1.0	0.0
1	1.0	0.0	7.00	0.0	1.0
2	0.0	1.0	2.00	1.0	0.0
3	1.0	1.0	4.20	1.0	0.0
4	0.0	0.0	7.80	0.0	1.0
5	1.0	0.0	3.90	1.0	0.0
6	0.0	1.0	4.00	1.0	0.0
7	1.0	1.0	2.00	0.0	0.0
8	0.0	0.0	3.50	0.0	1.0
9	1.0	0.0	2.60	1.0	0.0
10	0.0	0.0	4.10	0.0	1.0



Generalmente se “barajan” los datos antes de hacer las particiones para evitar sesgos en los datos resultantes

Conjunto de datos

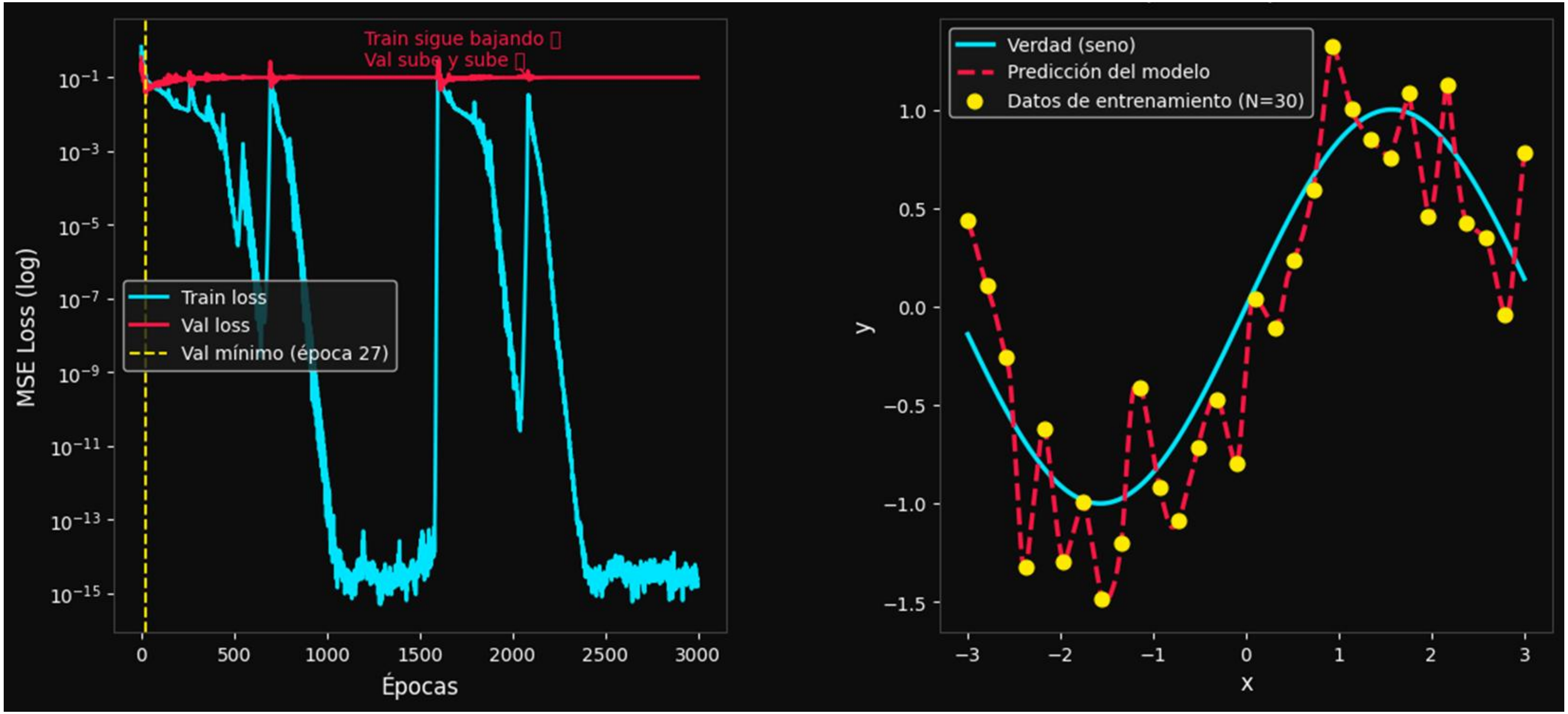


K-fold cross-validation: Es una técnica de validación para ver que tan bien generaliza un modelo a un conjunto de validación independiente.

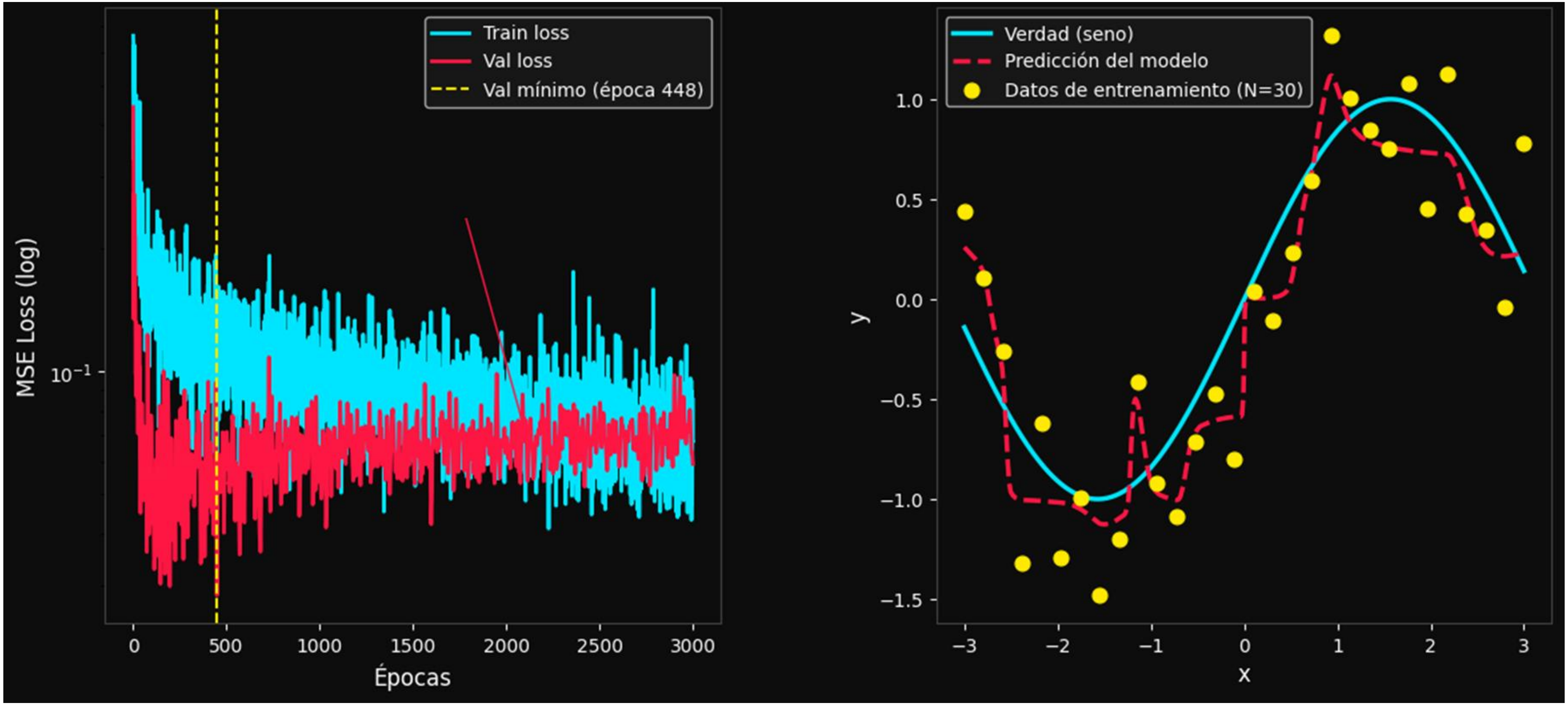
Se utilizan **K muestras reservadas** para validar el modelo, cada vez entrenando con las muestras restantes:

- Dividir el conjunto de entrenamiento en K grupos (folds).
- Repetir K veces el siguiente procedimiento:
  - Reservar el K^{th} fold para **validación**
  - Entrenar el modelo en los folds restantes
  - Calcular el desempeño en el fold de validación
- Combinar la métrica de rendimiento calculada

Sobre entrenamiento / Dropout



Sobre entrenamiento / Dropout

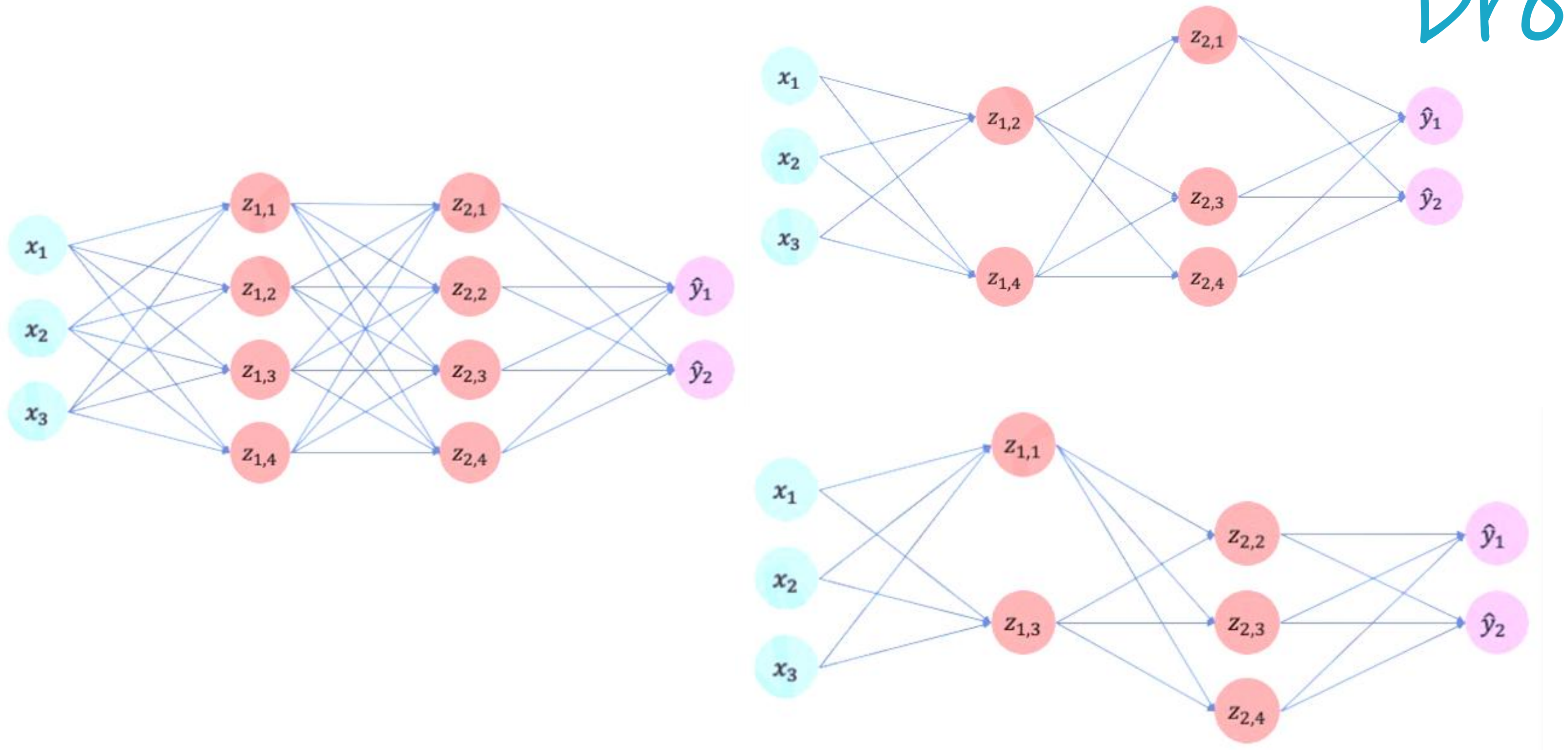


Sobre entrenamiento / Dropout

```
class RedConDropout(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(1, 512),  nn.ReLU(), nn.Dropout(0.4),
            nn.Linear(512, 512), nn.ReLU(), nn.Dropout(0.4),
            nn.Linear(512, 512), nn.ReLU(), nn.Dropout(0.4),
            nn.Linear(512, 512), nn.ReLU(), nn.Dropout(0.4),
            nn.Linear(512, 512), nn.ReLU(), nn.Dropout(0.4),
            nn.Linear(512, 1),
                )

    def forward(self, x):
        return self.net(x)
```

Sobre entrenamiento / 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

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.

UNIVERSIDAD EAFIT

Sobre entrenamiento / 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?

Parámetros vs Hiperparámetros

Parámetros

- *weights (weight parameters)*
- *biases (bias units)*

Hiperparámetros

- *minibatch size*
- *data normalization schemes*
- *number of epochs*
- *number of hidden layers*
- *number of hidden units*
- *learning rates*
- *(random seed, **why?**)*
- *loss function*
- *various weights (weighting terms)*
- *activation function types*
- *regularization schemes*
- *weight initialization schemes*
- *optimization algorithm type*

(En su mayoría sin explicación científica, principalmente ingeniería; se necesita probar muchas cosas → *graduate student descent*)