APRENDIZAJE AUTOMÁTICO.

Fundamentos y Aplicaciones en Meteorología del Espacio

DCAO, UBA
08-12 AGOSTO 2022

Dra María Graciela Molina
FACET-UNT / CONICET
Tucumán Space Weather Center - TSWC

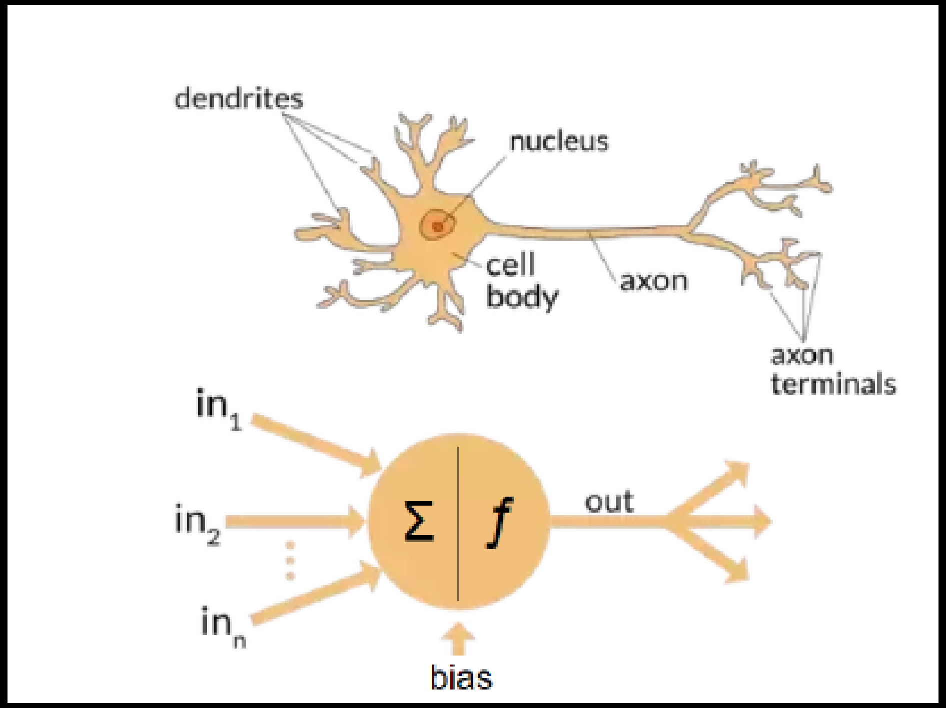
https://spaceweather.facet.unt.edu.ar/
IG -> @spaceweatherargentina

gmolina@herrera.unt.edu.ar

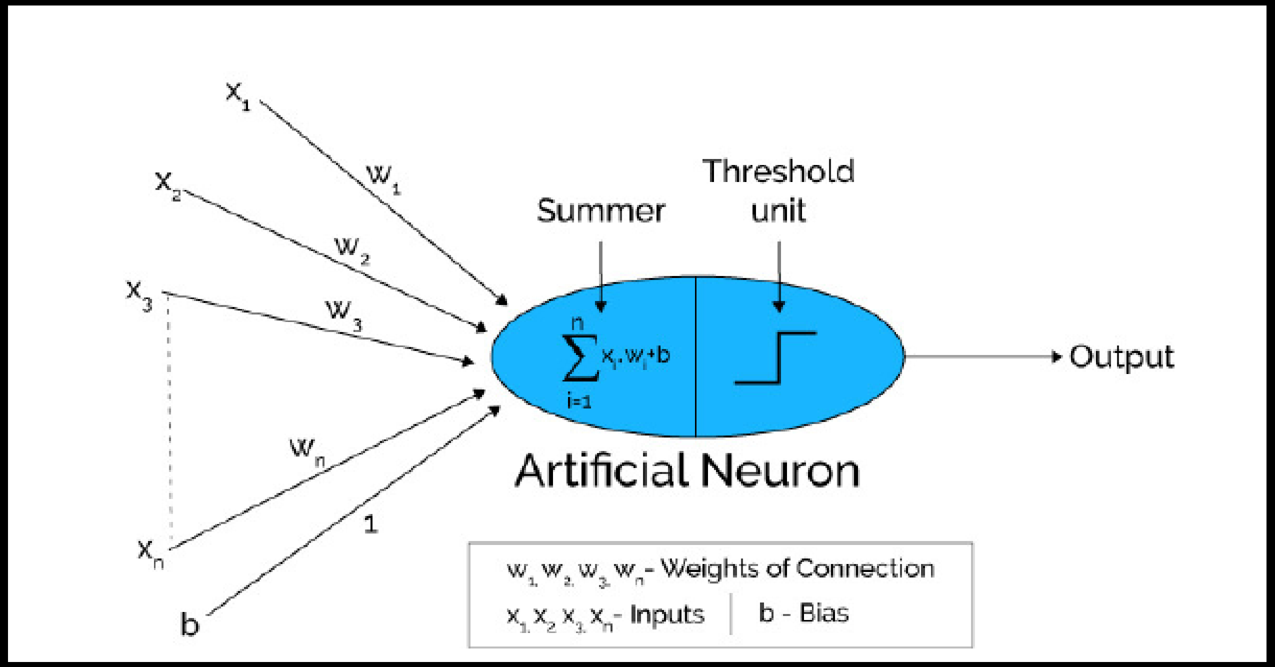
TSWC, 2022

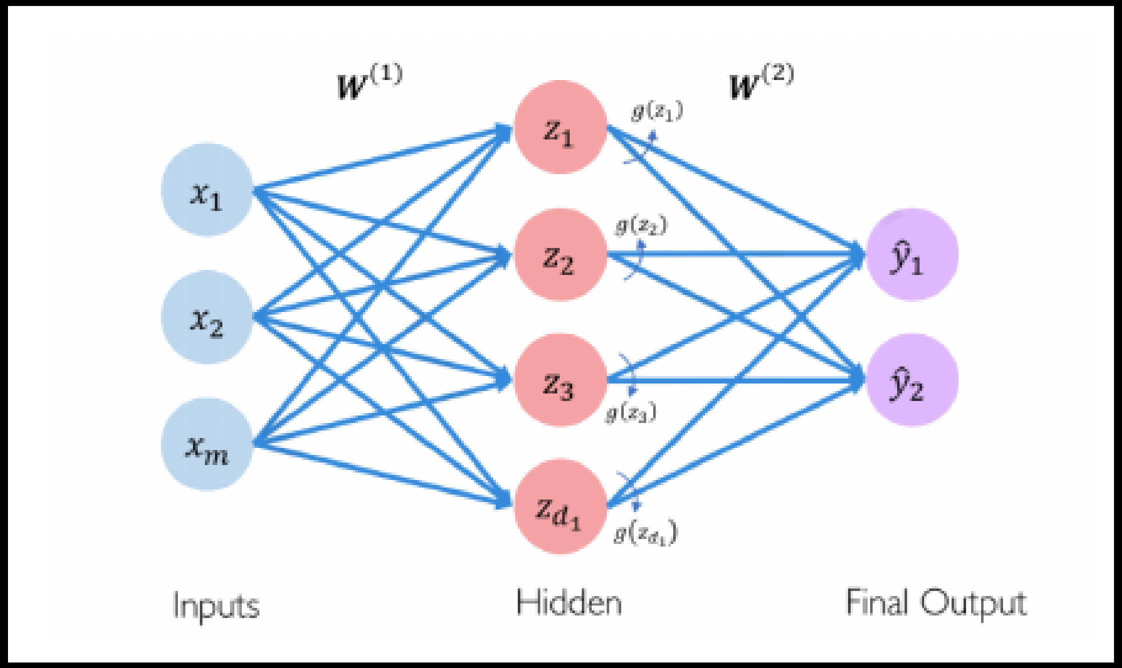
Artificial Neural Networks

ANN



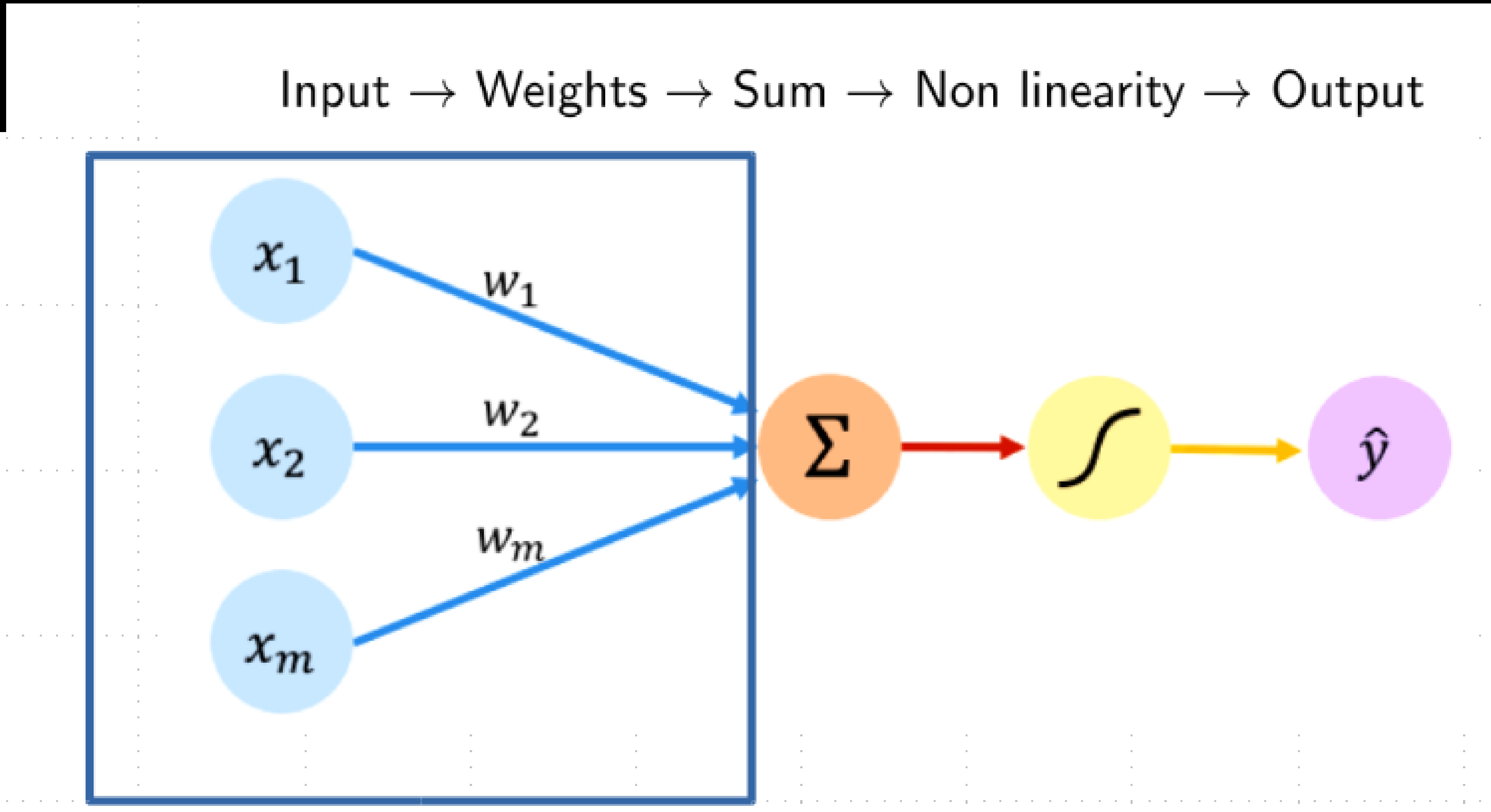
- Data-driven modeling
- Inspired in brain neural networks
- Solving wide number of complex problems: facial recognition, handwrite recognition, weather forecasting, etc.
- Black-box modelling (?) based in the composition of interconnected neurons (neural network)





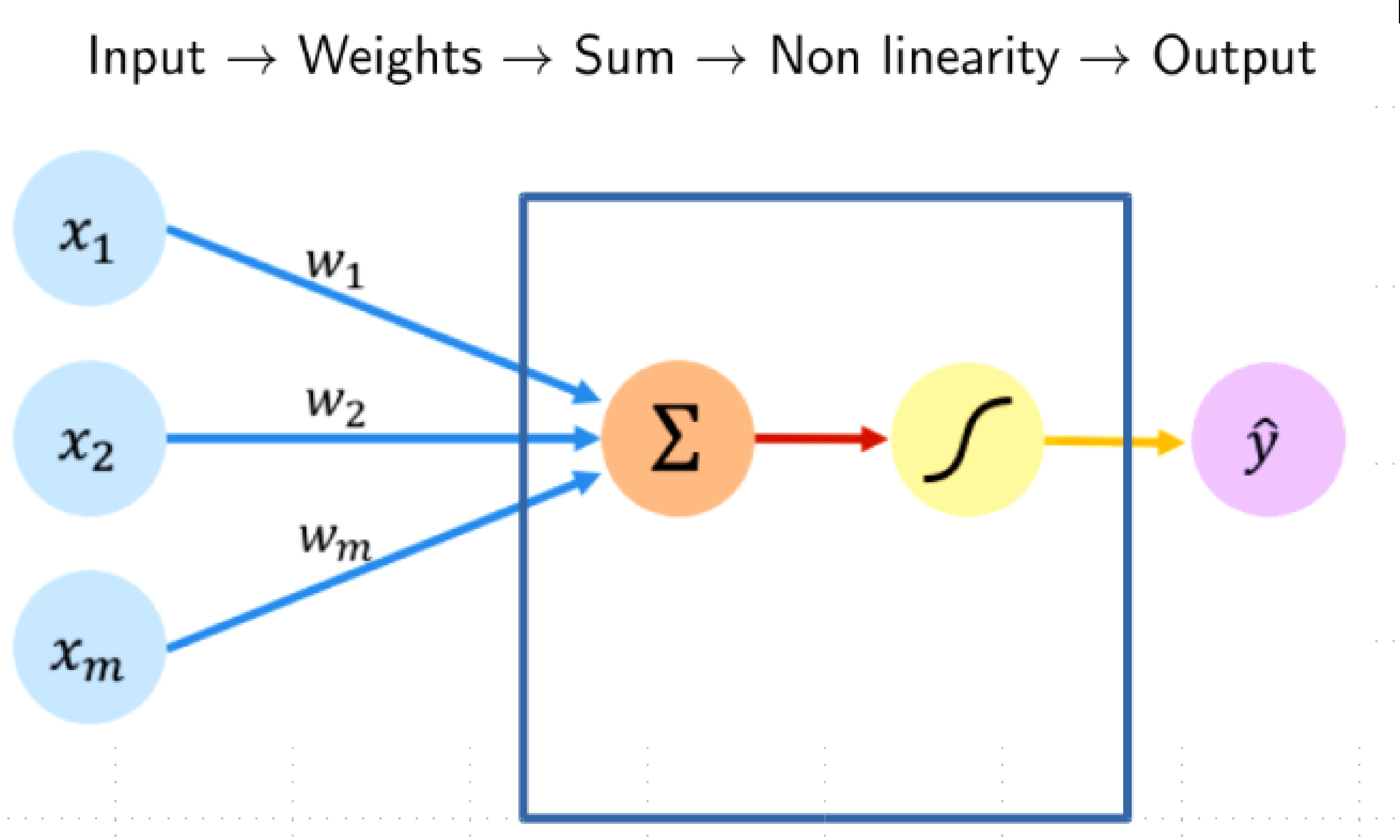
Perceptron

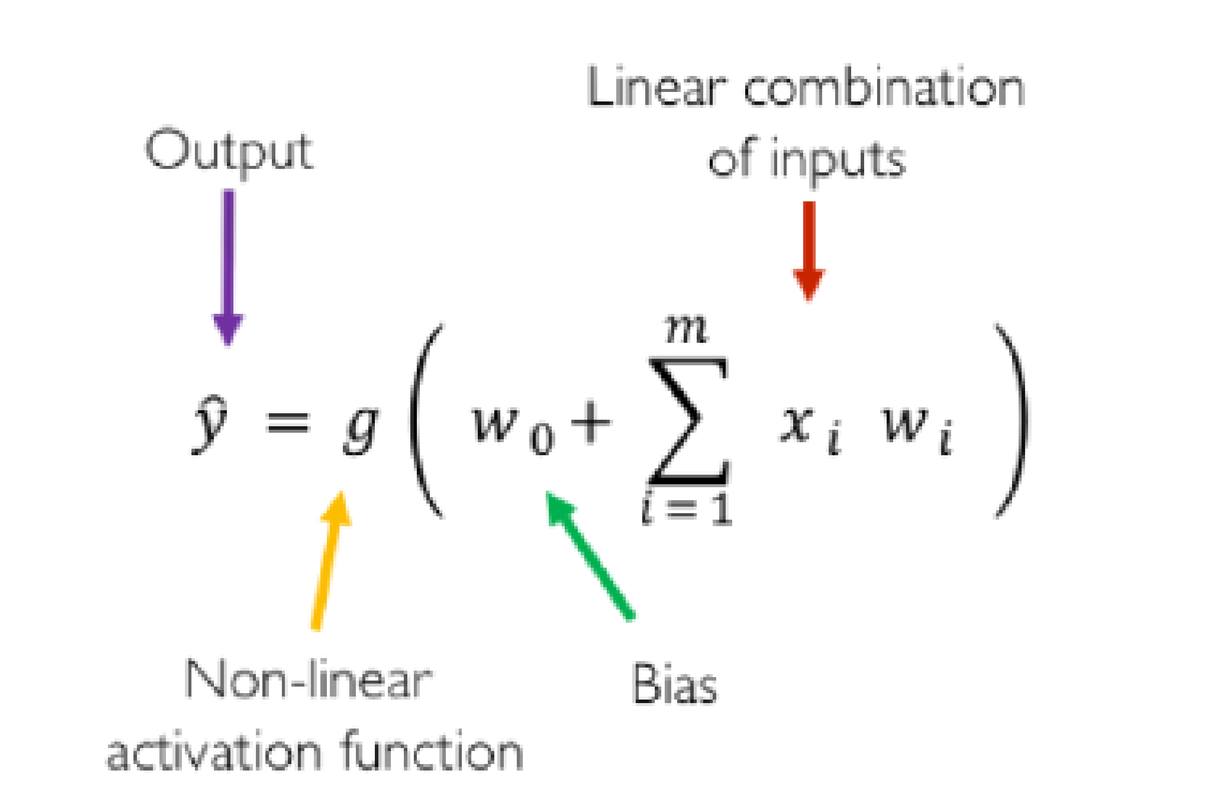
- Unidad fundamental para construir redes neuronales



- Cada una de las entradas corresponde a una característica (feature)
- A cada una de las xi se les aplica un peso (w) (ponderación de los valores de la entrada)
- Los pesos son las vbles que se ajustan durante el entrenamiento

Perceptron

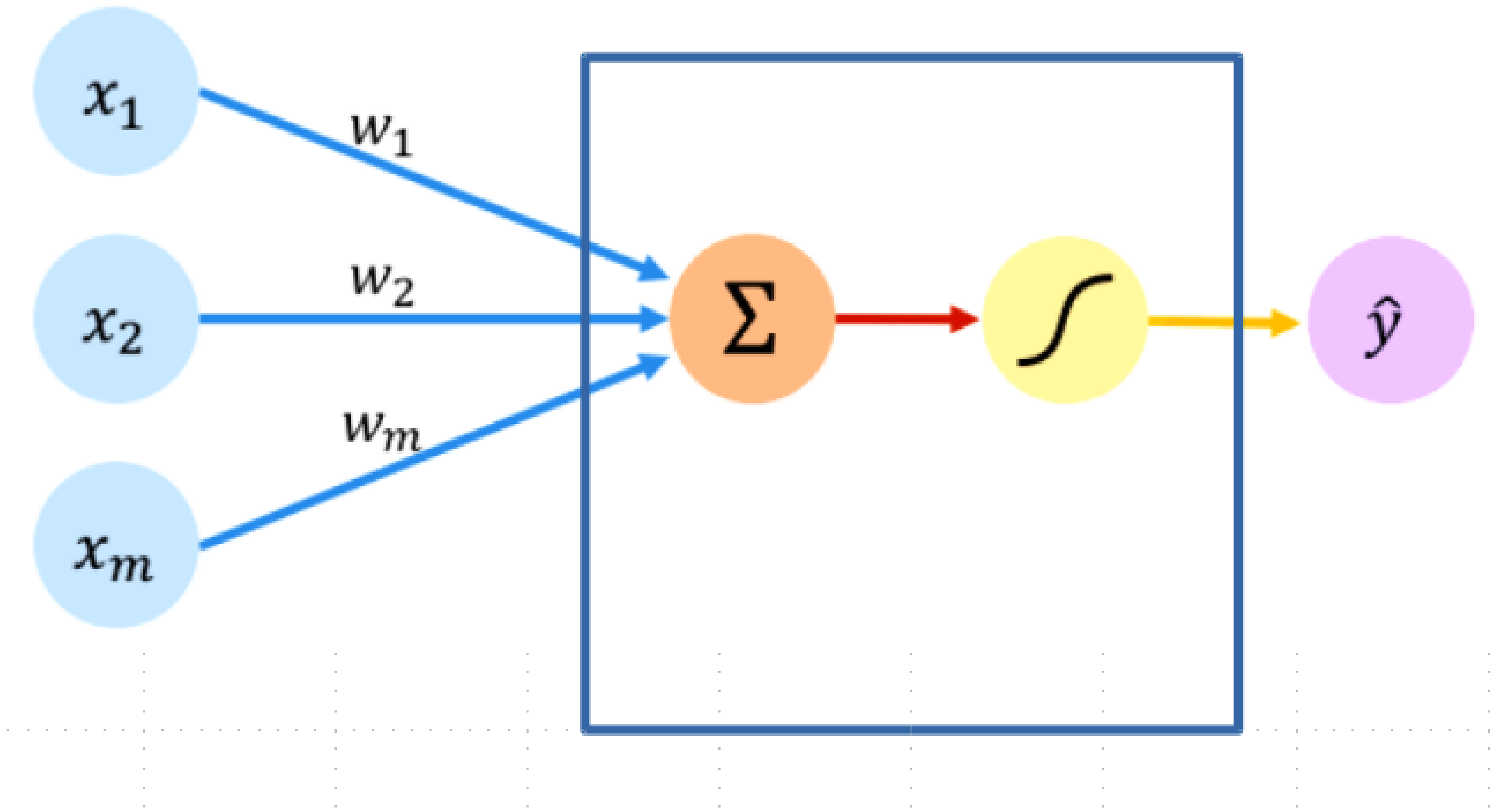




- Capa oculta (hidden layer).
- El bias es un término de ajuste
- La función de activación se aplica a la combinación lineal de las entradas ponderadas teniendo en cuenta el bias

Perceptron

Input → Weights → Sum → Non linearity → Output



$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i w_i \right)$$

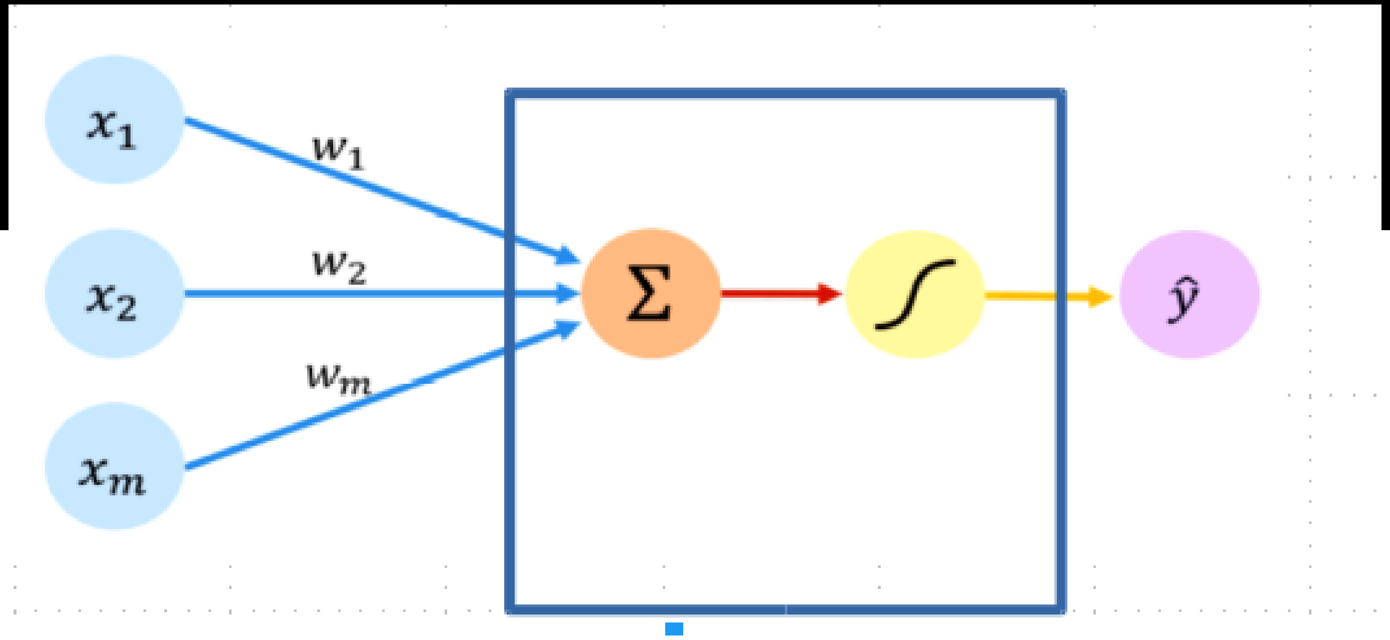
$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

Activation function

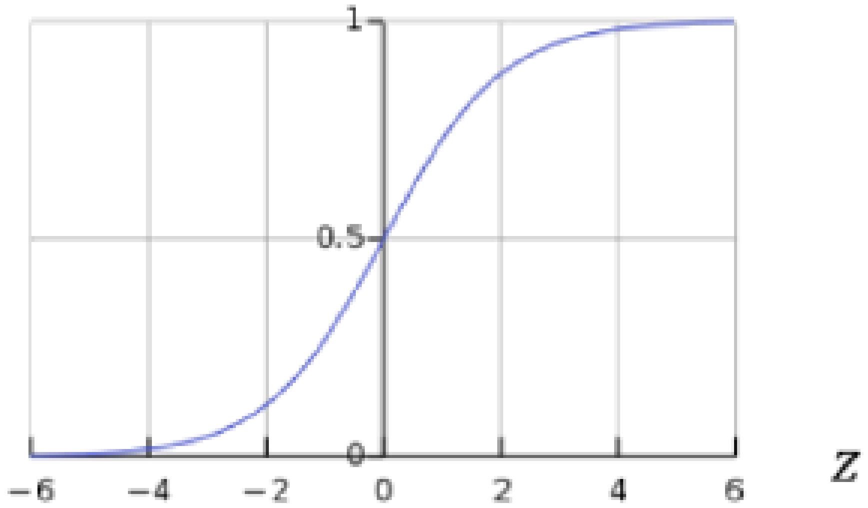
- Agrega no-linealidd al modelo

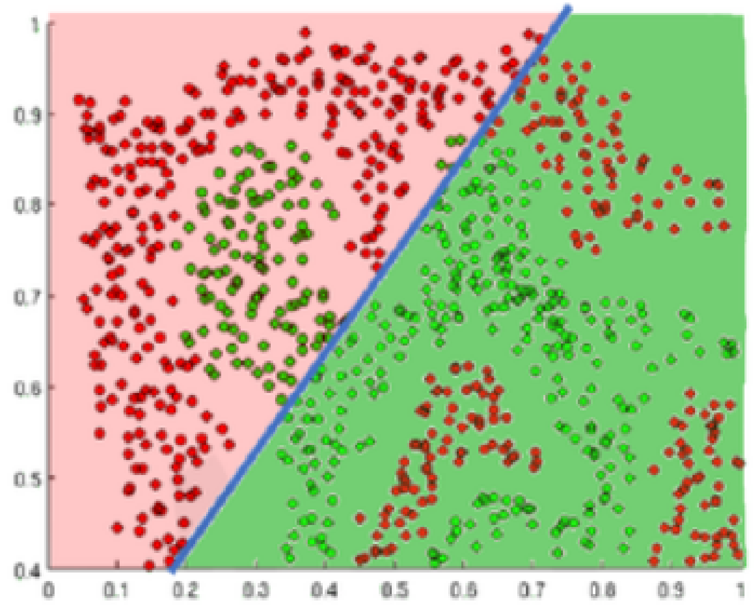
$$\hat{y} = g\left(w_0 + X^T W \right)$$



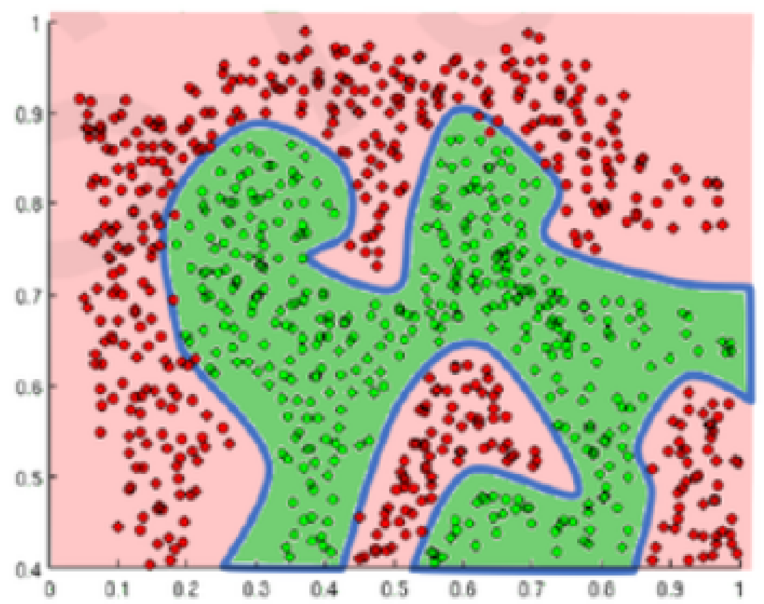
- Una de las más usadas en la función sigmoide.
- Produce resultados entre 0 y 1

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$





Linear activation functions produce linear decisions no matter the network size

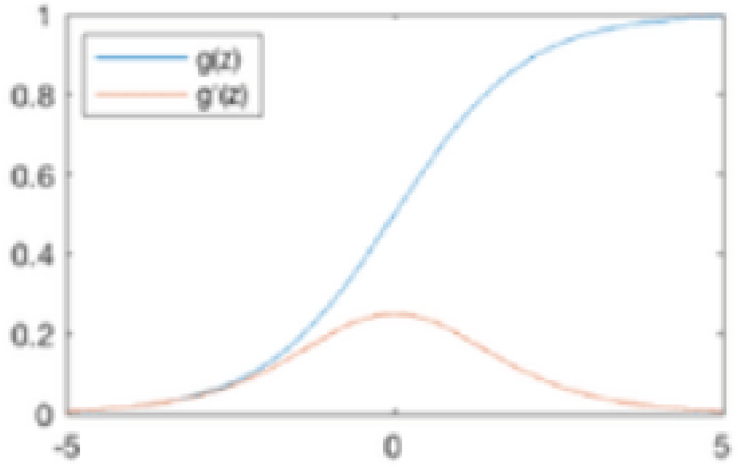


Non-linearities allow us to approximate arbitrarily complex functions

Activation function

$$\hat{y} = g\left(w_0 + X^T W \right)$$

Sigmoid Function

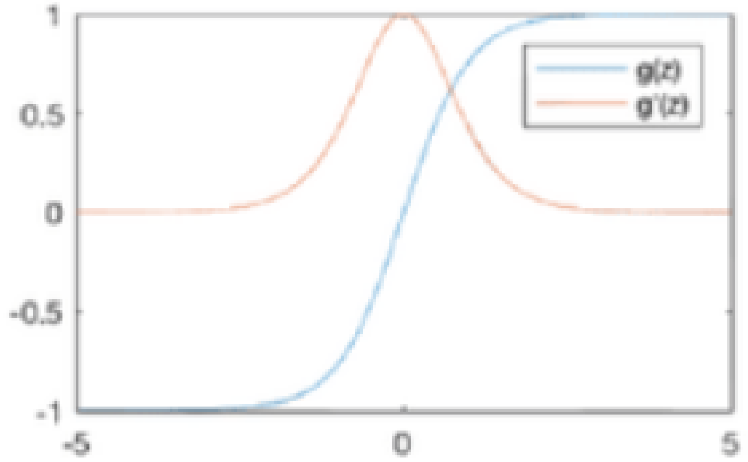


$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

```
tf.math.sigmoid(z)
```

Hyperbolic Tangent

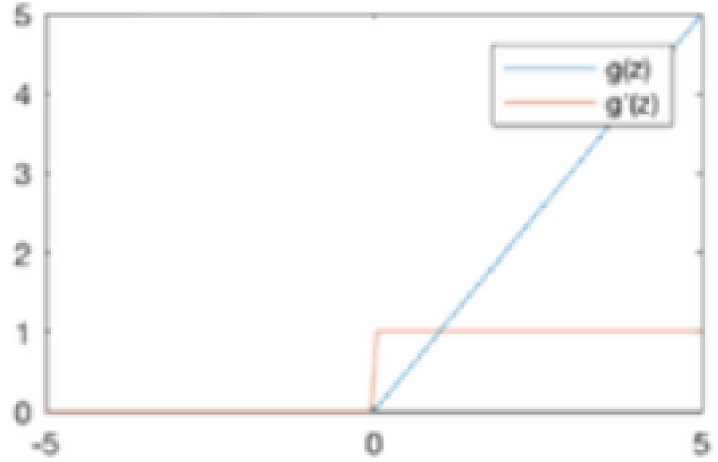


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

```
tf.math.tanh(z)
```

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

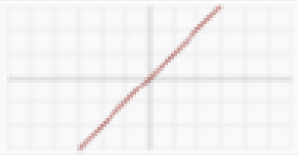
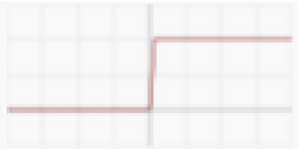
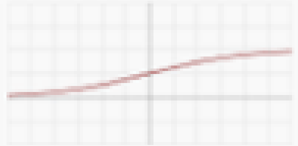
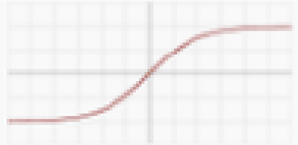
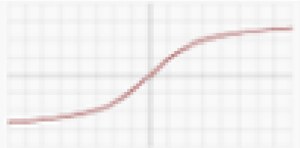
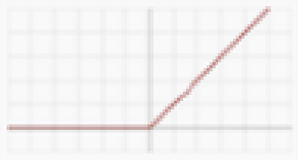
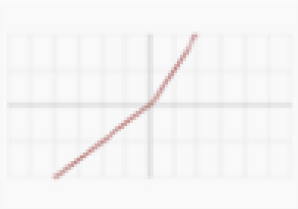
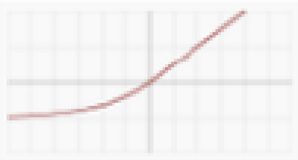
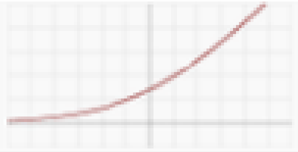
$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

```
tf.nn.relu(z)
```

Activation function

- Adds no-linearity to the model

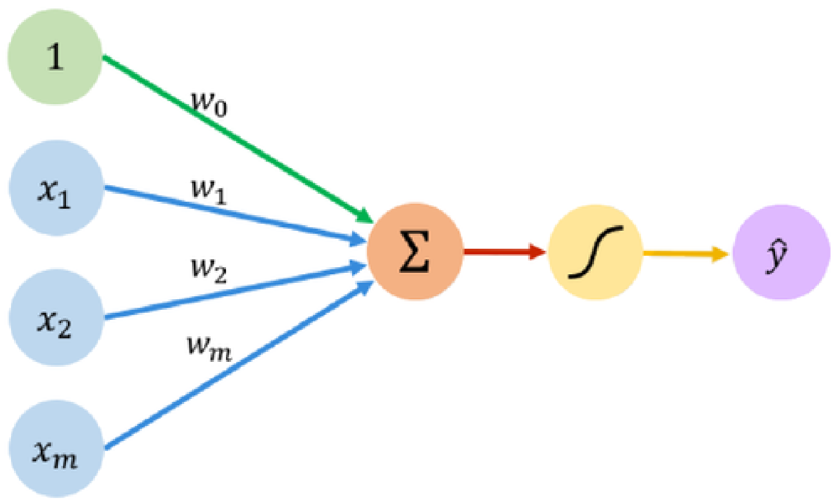
$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

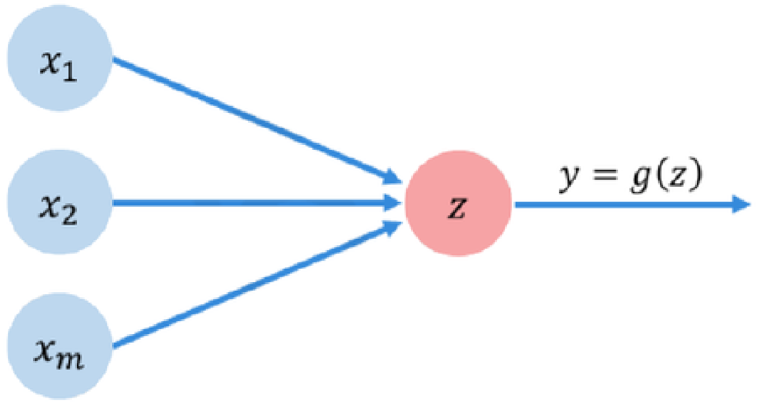
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \ne 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \ge 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \ge 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \ge 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \ge 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Feed Forward (FF)

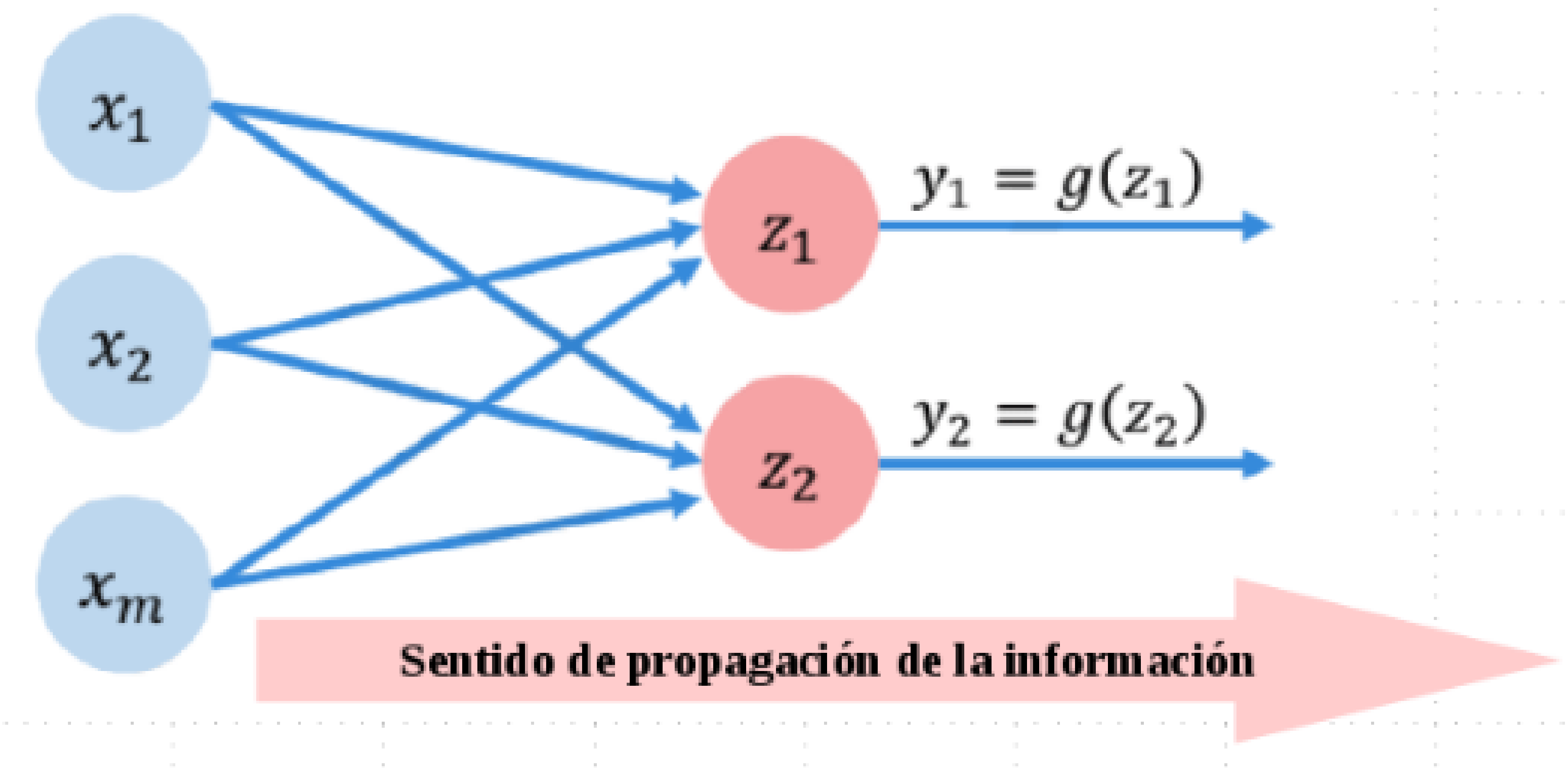
- Perceptrón simplificado (n entradas → 1 salida)

$$z = w_0 + \sum_{j=1}^{m} x_j\, w_j$$





- Armemos una red neuronal compuesta por múltiples perceptrones (n entradas → m salida)

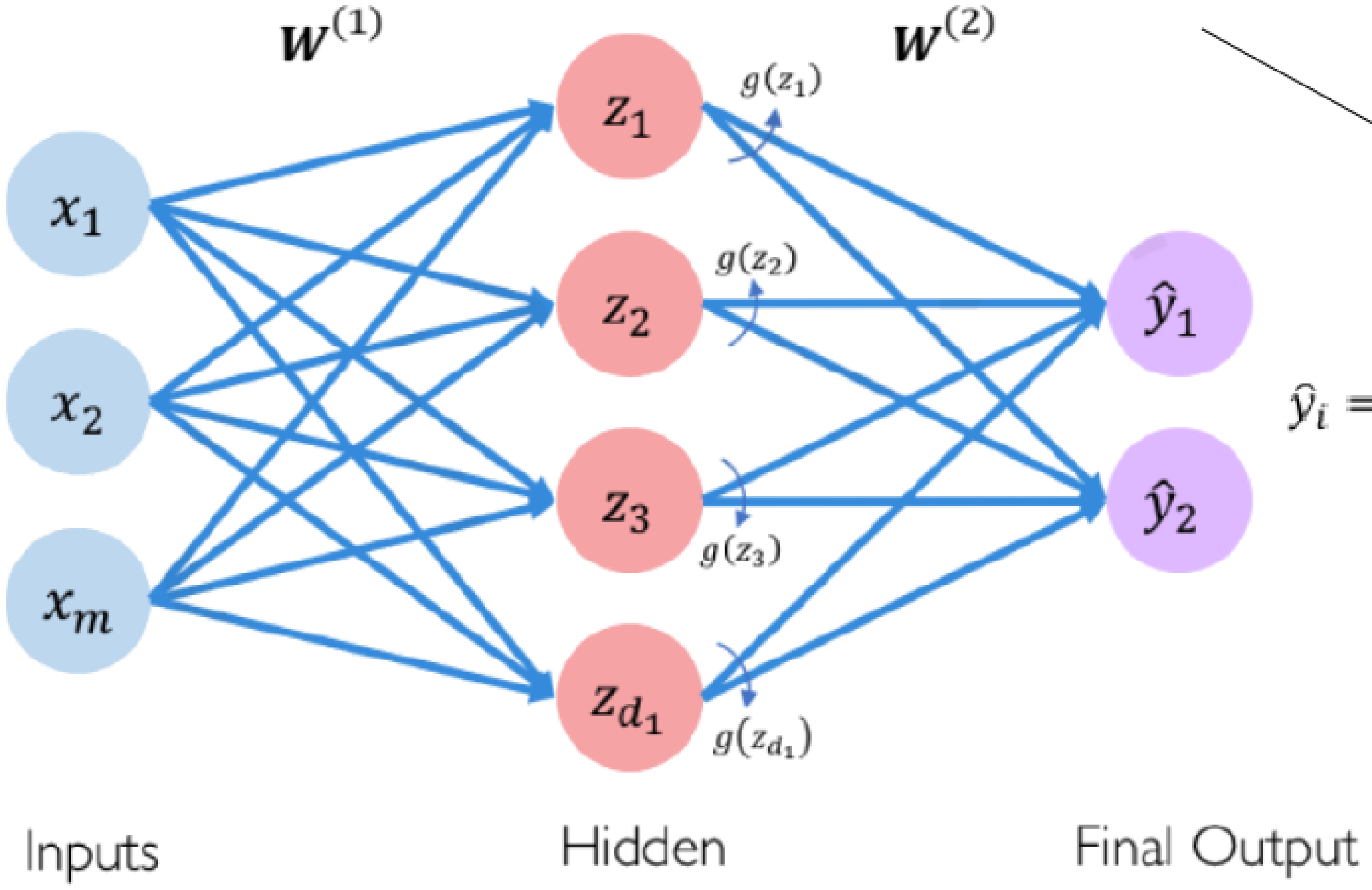


$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j\, w_{j,i}$$

Observación: todas las entradas están conectadas a todas las salidas → dense layers

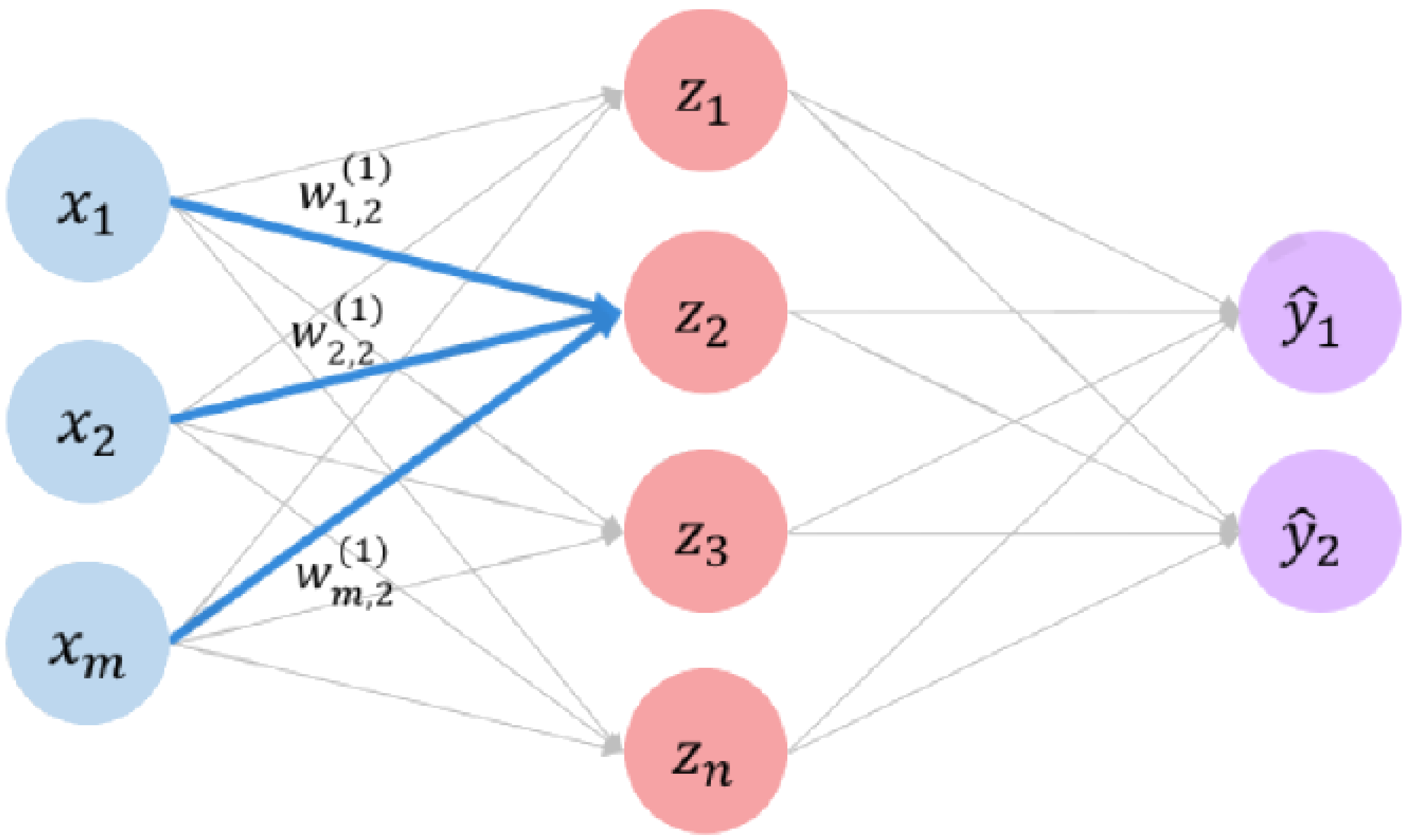
1-Layer Neural Network

$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)}$$



$$\hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} g(z_j) \, w_{j,i}^{(2)} \right)$$

1-Layer Neural Network

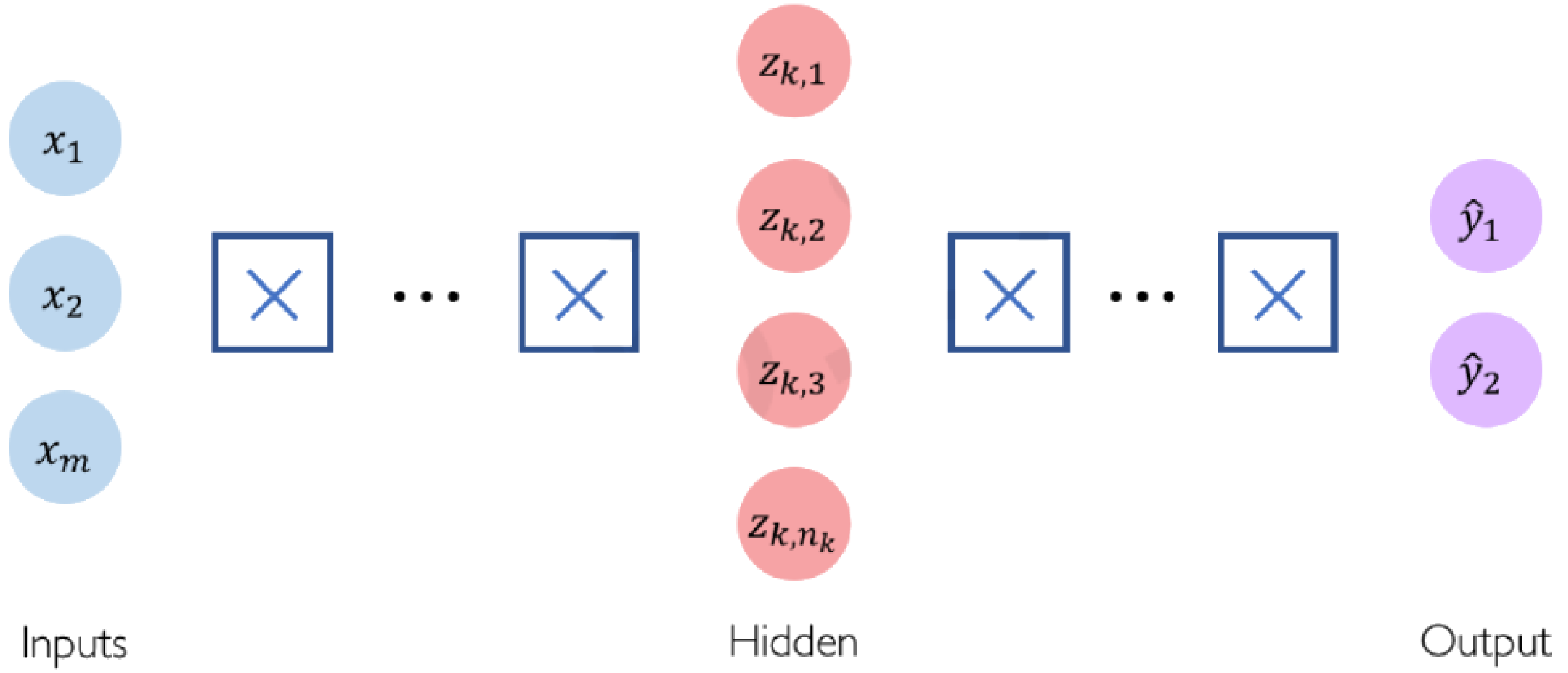


$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,2}^{(1)}$$

$$= w_{0,2}^{(1)} + x_1 \, w_{1,2}^{(1)} + x_2 \, w_{2,2}^{(1)} + x_m \, w_{m,2}^{(1)}$$

M-layers Neural Network (multi-layer)

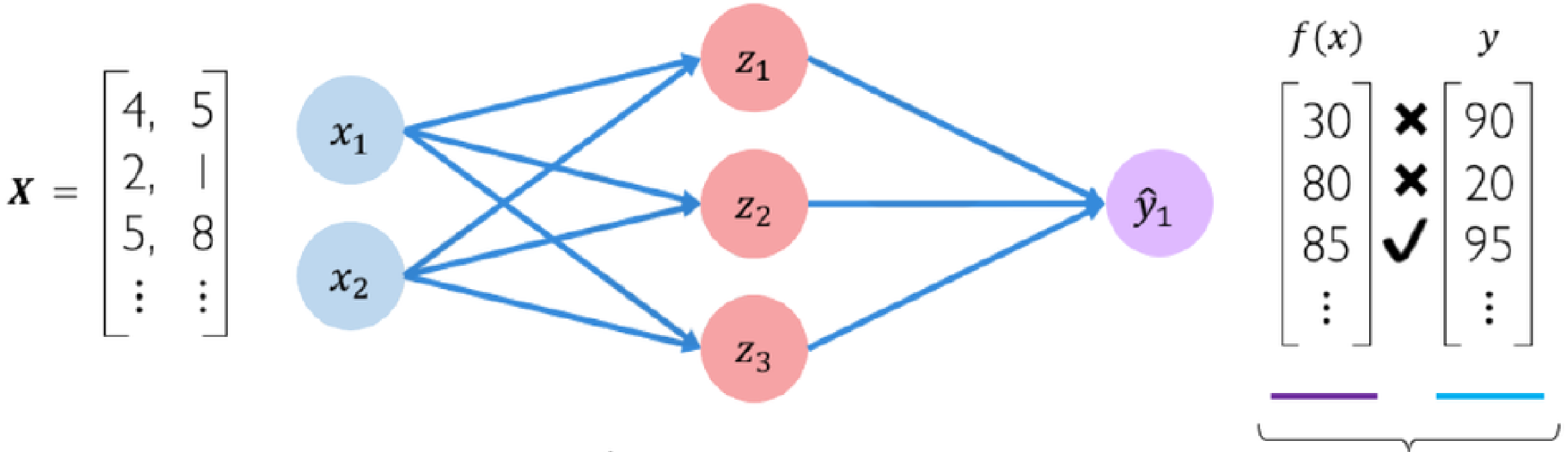
- Deep Neural Network



$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

Loss Function

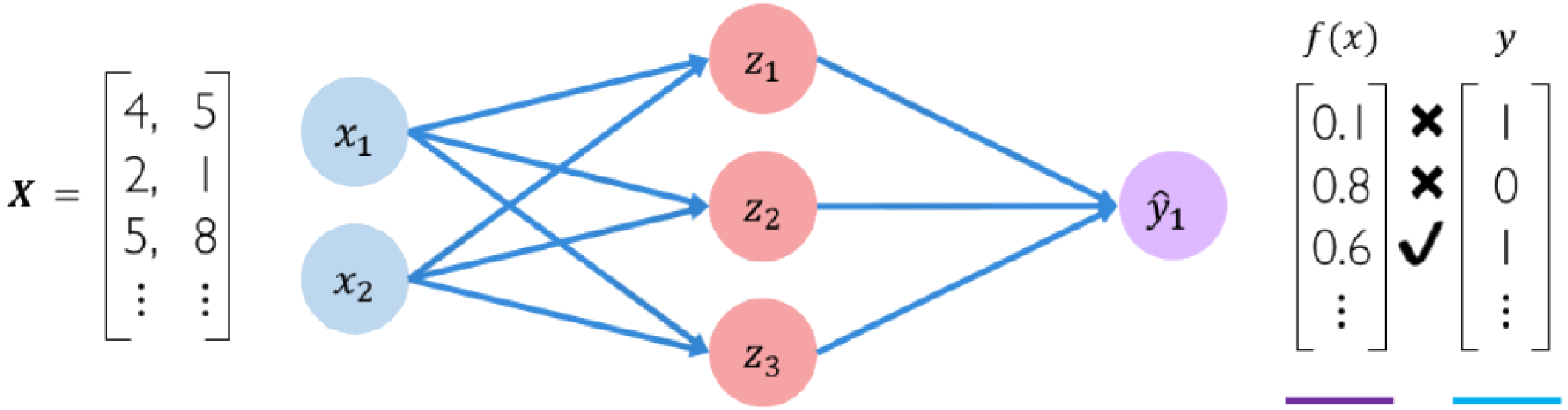
Mean squared error loss can be used with regression models that output continuous real numbers



Final Grades (percentage)

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\Big(\underbrace{y^{(i)}}_{\text{Actual}} - \underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big)^2$$

Cross entropy loss can be used with models that output a probability between 0 and 1



$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\underbrace{y^{(i)}}_{\text{Actual}}\log\Big(\underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big) + (1-\underbrace{y^{(i)}}_{\text{Actual}})\log\Big(1-\underbrace{f\big(x^{(i)};\boldsymbol{W}\big)}_{\text{Predicted}}\Big)$$

symbol	name	equation
$\mathcal{L}_1$	L_1 loss	$\Vert\mathbf{y}-\mathbf{o}\Vert_1$
$\mathcal{L}_2$	L_2 loss	$\Vert\mathbf{y}-\mathbf{o}\Vert_2^2$
$\mathcal{L}_1 \circ \sigma$	expectation loss	$\Vert\mathbf{y}-\sigma(\mathbf{o})\Vert_1$
$\mathcal{L}_2 \circ \sigma$	regularised expectation loss[1]	$\Vert\mathbf{y}-\sigma(\mathbf{o})\Vert_2^2$
$\mathcal{L}_\infty \circ \sigma$	Chebyshev loss	$\max_j \vert\sigma(\mathbf{o})^{(j)} - \mathbf{y}^{(j)}\vert$
hinge	hinge [13] (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})$
hinge^2	squared hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^2$
hinge^3	cubed hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^3$
log	log (cross entropy) loss	$-\sum_j \mathbf{y}^{(j)} \log\sigma(\mathbf{o})^{(j)}$
$\log^2$	squared log loss	$-\sum_j [\mathbf{y}^{(j)} \log\sigma(\mathbf{o})^{(j)}]^2$
tan	Tanimoto loss	$\frac{-\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\Vert\sigma(\mathbf{o})\Vert_2^2 + \Vert\mathbf{y}\Vert_2^2 - \sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}$
D_{CS}	Cauchy-Schwarz Divergence [3]	$-\log\frac{\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\Vert\sigma(\mathbf{o})\Vert_2 \Vert\mathbf{y}\Vert_2}$

- https://arxiv.org/pdf/1702.05659.pdfs

Training (loss optimization)

- Encontrar los valores de los **pesos** de la red neuronal tal que la **pérdida o costo sea mínimo**

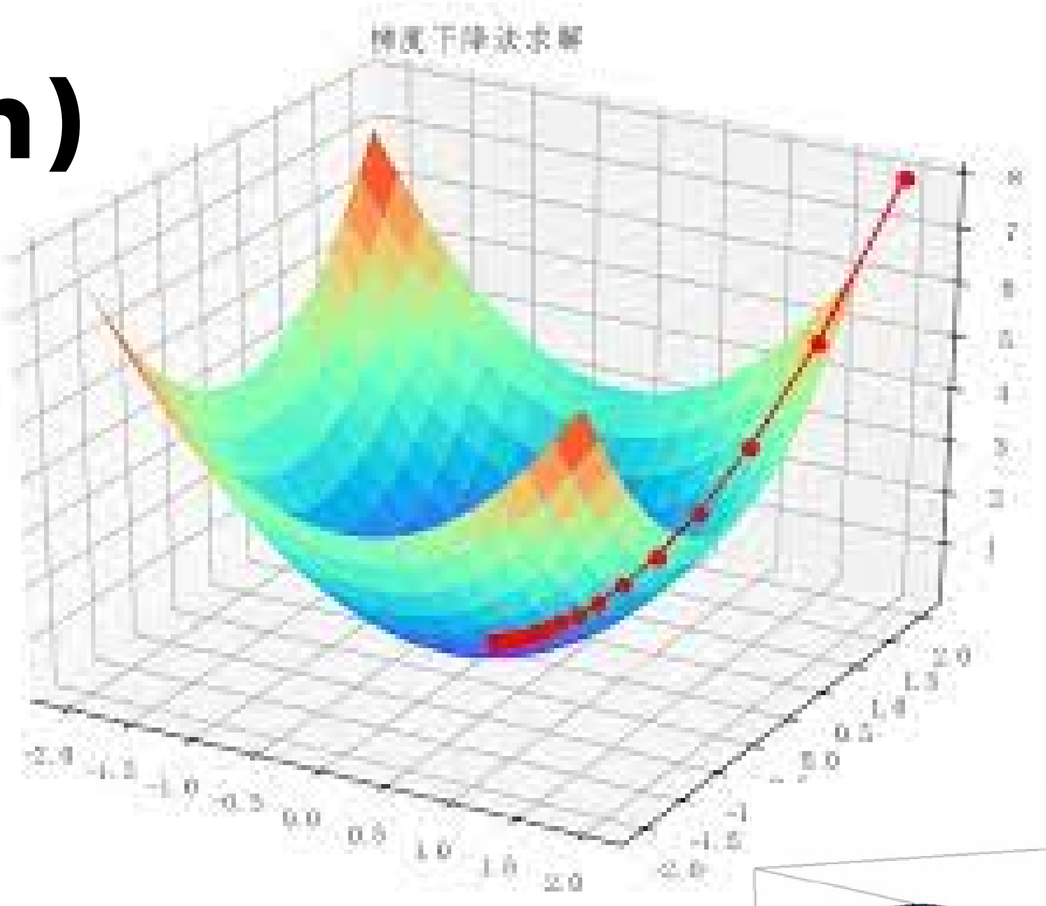
$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W})$$

- Es un problema de optimización
- Podemos usar algún método numérico de optimización

MÉTODO DEL GRADIENTE DESCENDIENTE

Training (loss optimization)



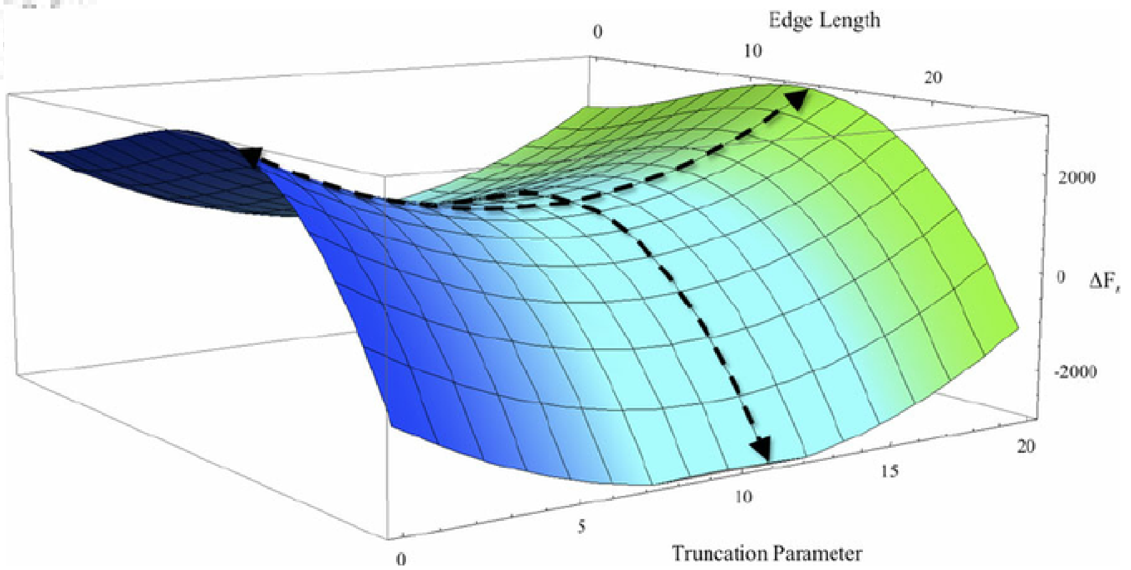
$$W^{*} = \underset{W}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; W\right), y^{(i)}\right)$$

$$W^{*} = \underset{W}{\operatorname{argmin}}\, J(W)$$

$\downarrow$

Algorithm

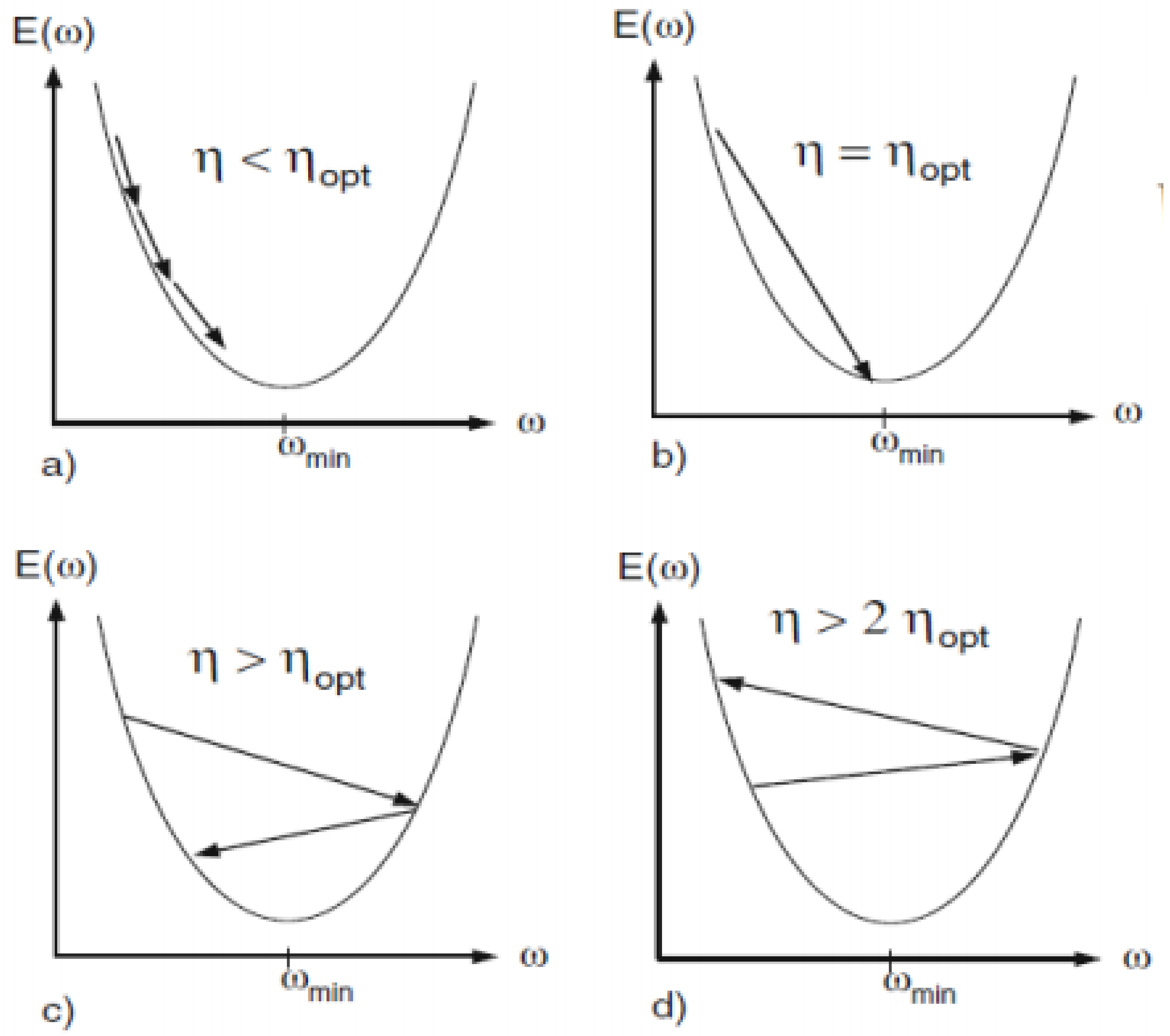
1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$
4. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
5. Return weights



Principales problemas:

- Mínimos locales
- Silla de montar: el g. d. un vez que llega a una región con gradiente cero, no puede escapar de allí independientemenete de la calidad del mínimo.

Training (loss optimization)



$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \boxed{\eta} \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

How can we set the learning rate?

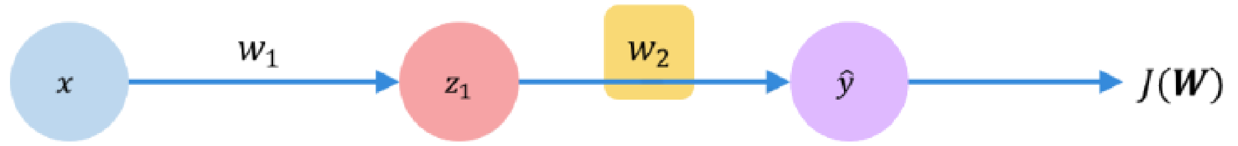
- https://ruder.io/optimizing-gradient-descent/index.html

Gradient Descent Algorithms

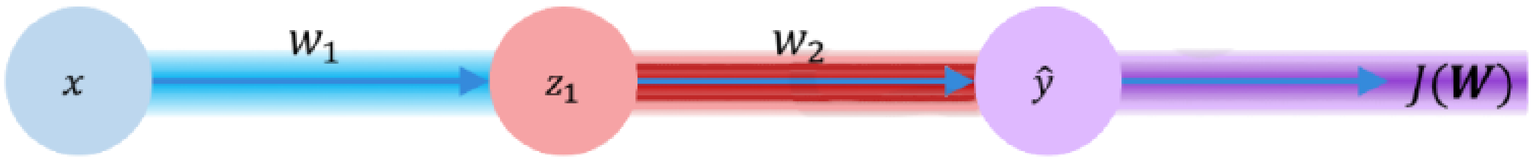
Algorithm	TF Implementation	Reference
• SGD	`tf.keras.optimizers.SGD`	Kiefer & Wolfowitz. "Stochastic Estimation of the Maximum of a Regression Function." 1952.
• Adam	`tf.keras.optimizers.Adam`	Kingma et al. "Adam: A Method for Stochastic Optimization." 2014.
• Adadelta	`tf.keras.optimizers.Adadelta`	Zeiler et al. "ADADELTA: An Adaptive Learning Rate Method." 2012.
• Adagrad	`tf.keras.optimizers.Adagrad`	Duchi et al. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." 2011.
• RMSProp	`tf.keras.optimizers.RMSProp`	

Backpropagation

- Qué tanto puede afectar a la función de pérdida un pequeño cambio en uno del los pesos?



- Aplicando la regla de la cadena



$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

- Repetir para cada uno de los pesos en la red usando los gradientes de las capas siguiente!

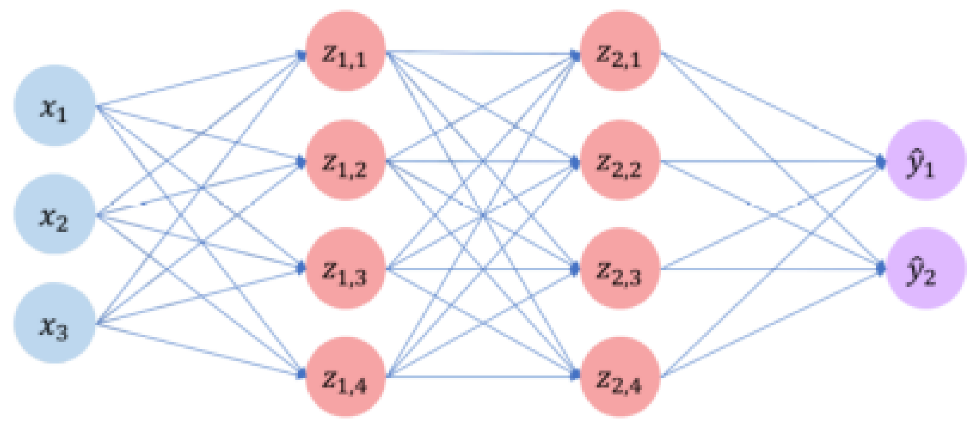
- Info se propaga desde las entradas hacia adelante mediante sus parámetros hasta que logra hacer una predicción

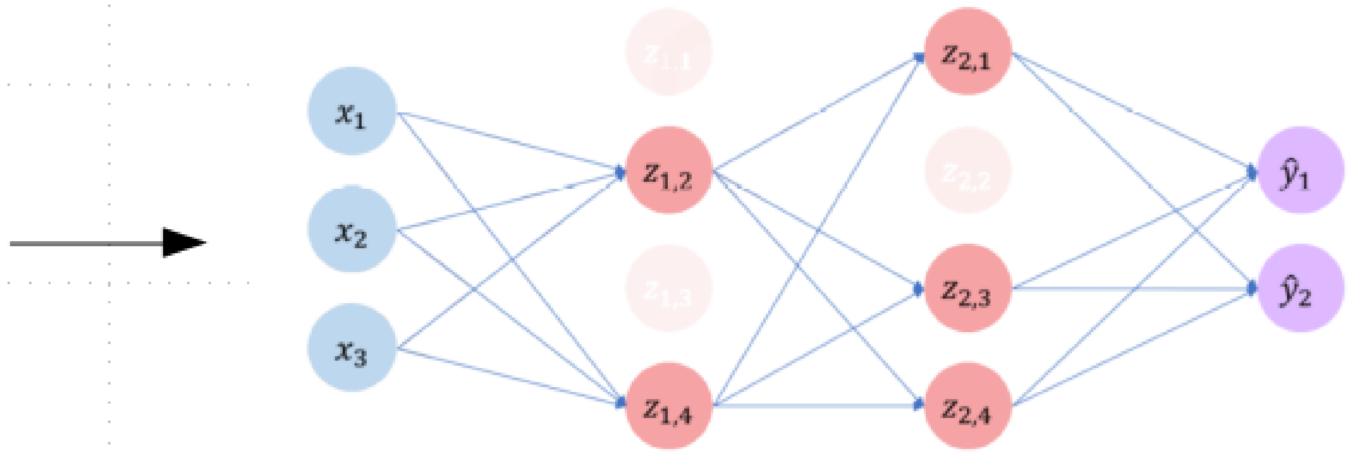
- Luego realiza una propagación hacia atrás a lo largo de la red para ir modificando los parámetros de manera que el error final sea el mínimo.

- El error asociado a una mala predicción es usado para ajustar los parámetros (el aprendizaje puede ser visto como una optimización). Para esto la salida y es comparada con el valor esperado (target) del conjunto de entrenamiento (z). La diferencia se entre el valor esperado y el de salida se llama error o residuo.

Regularización

- Técnica que consiste en establecer restricciones al problema de optimización para evitar llegar a modelos complejos
- La idea es poder generalizar nuestro modelo para nuevos datos



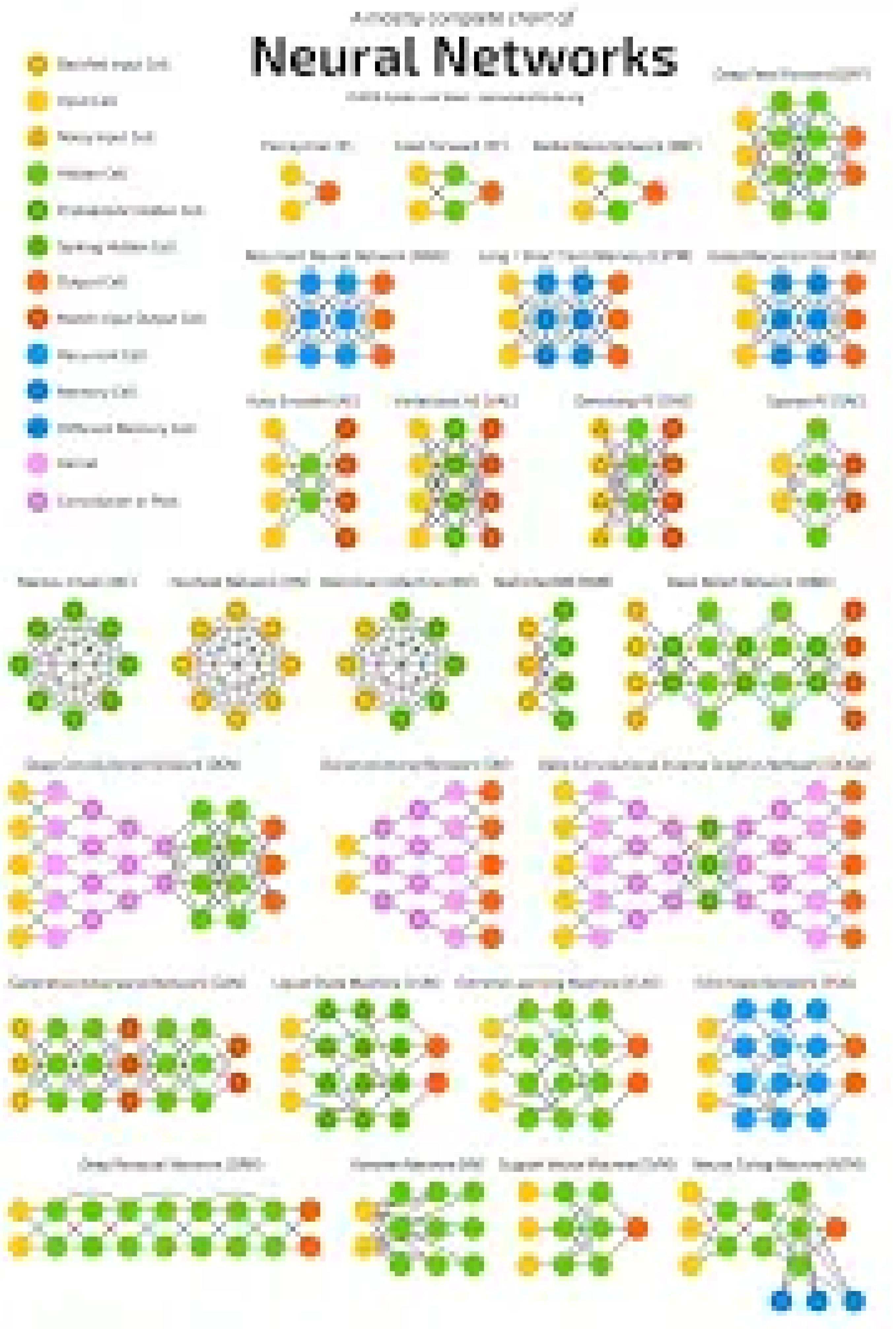


DROP OUT

- Durante el entrenamiento de manera aleatoria se setean algunas activaciones en 0.
- << overfitting

EARLY STOPING

- Consiste en parar antes de llegar al overfitting observando los datos de training y testing.


Neural Networks


ZOO

HANDS-ON LAB

TP - 1

2 problems

- Radar signal classification (mandatory)
- Flare classification (optional)