

REDES NEURONALES

APRENDIZAJE DE MAQUINA I - CEIA - FIUBA

Dr. Ing. Facundo Adrián Lucianna

Dr. Ing. Álvaro Gabriel Pizá

REPASO CLASE ANTERIOR

- Arboles de decisión
 - Arboles de regresión
 - Arboles de clasificación

ARBOLES DE DECISIÓN

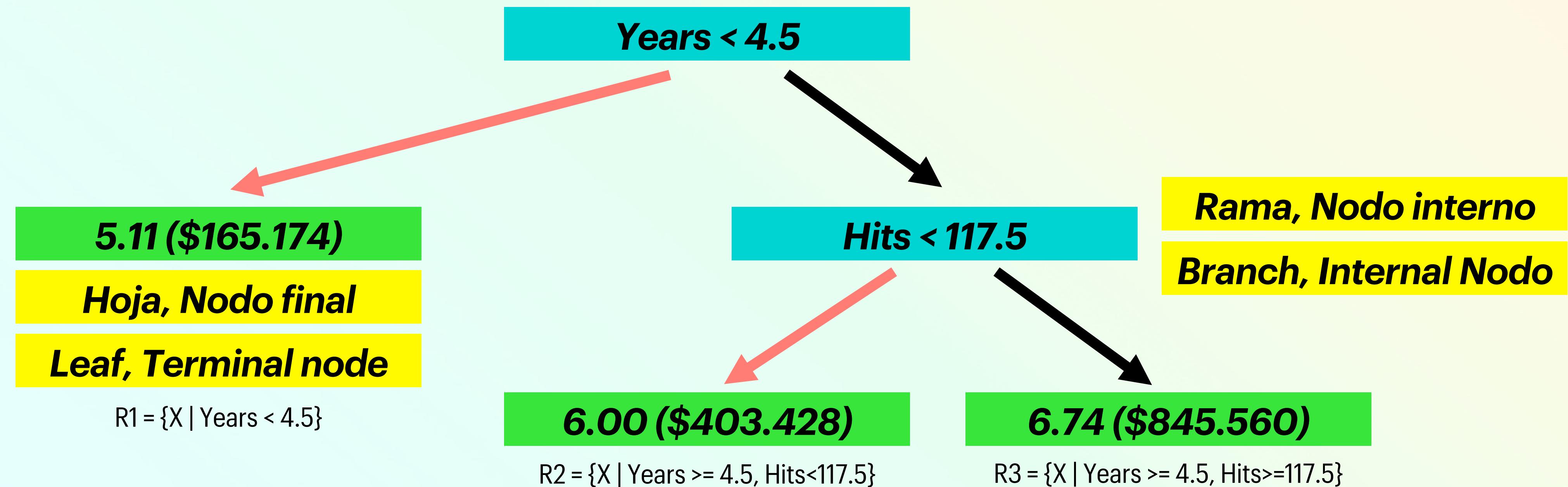
Los árboles de clasificación y regresión, conocidos como **CART** (Classification and Regression Trees), son una poderosa técnica de aprendizaje automático que se utiliza ampliamente para resolver problemas tanto de clasificación como de regresión.

Los árboles CART son modelos de decisión que utilizan una estructura de árbol para realizar predicciones basadas en reglas **lógicas sencillas y fáciles de interpretar**.

Arboles de clasificación

Arboles de regresión

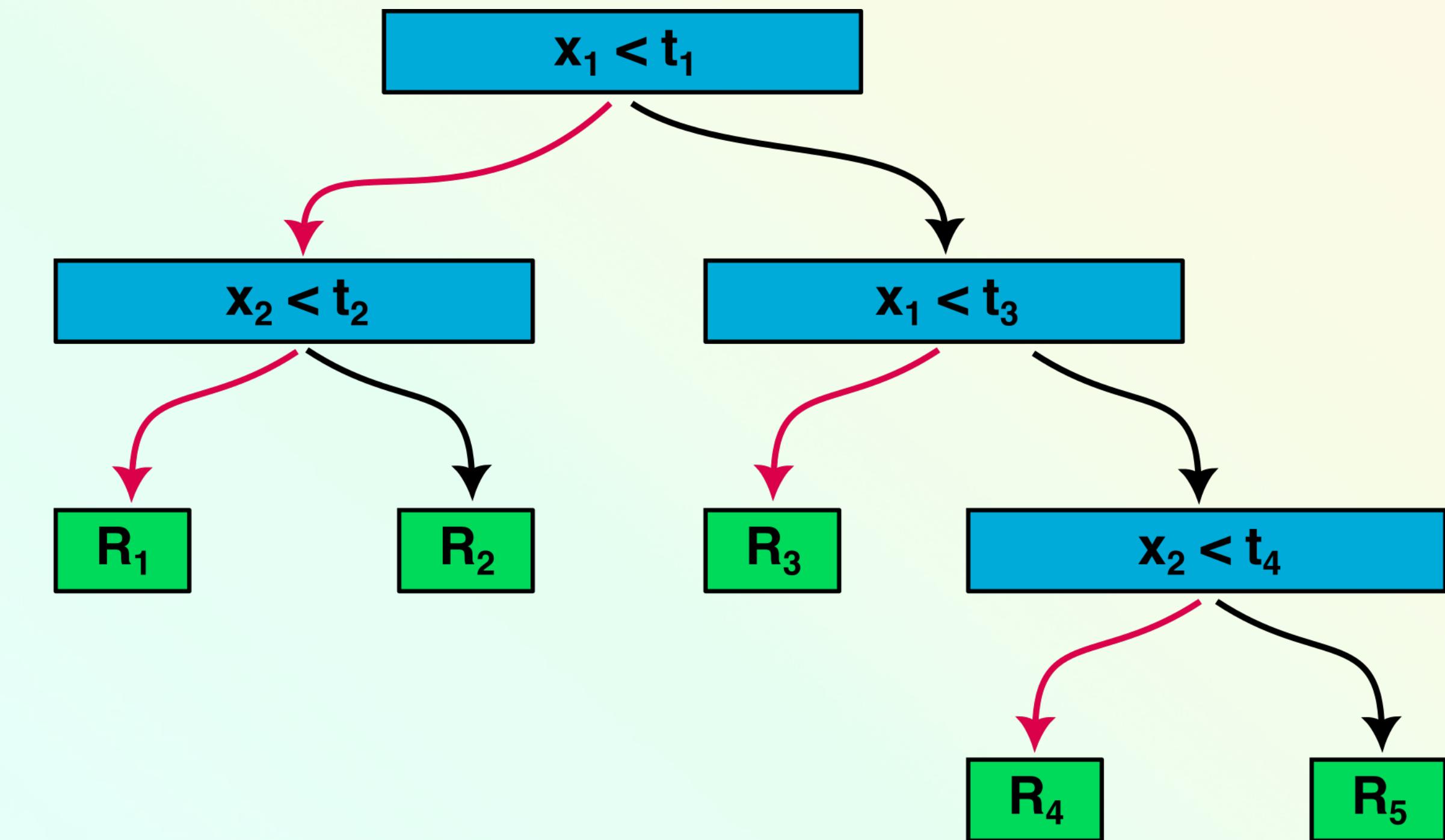
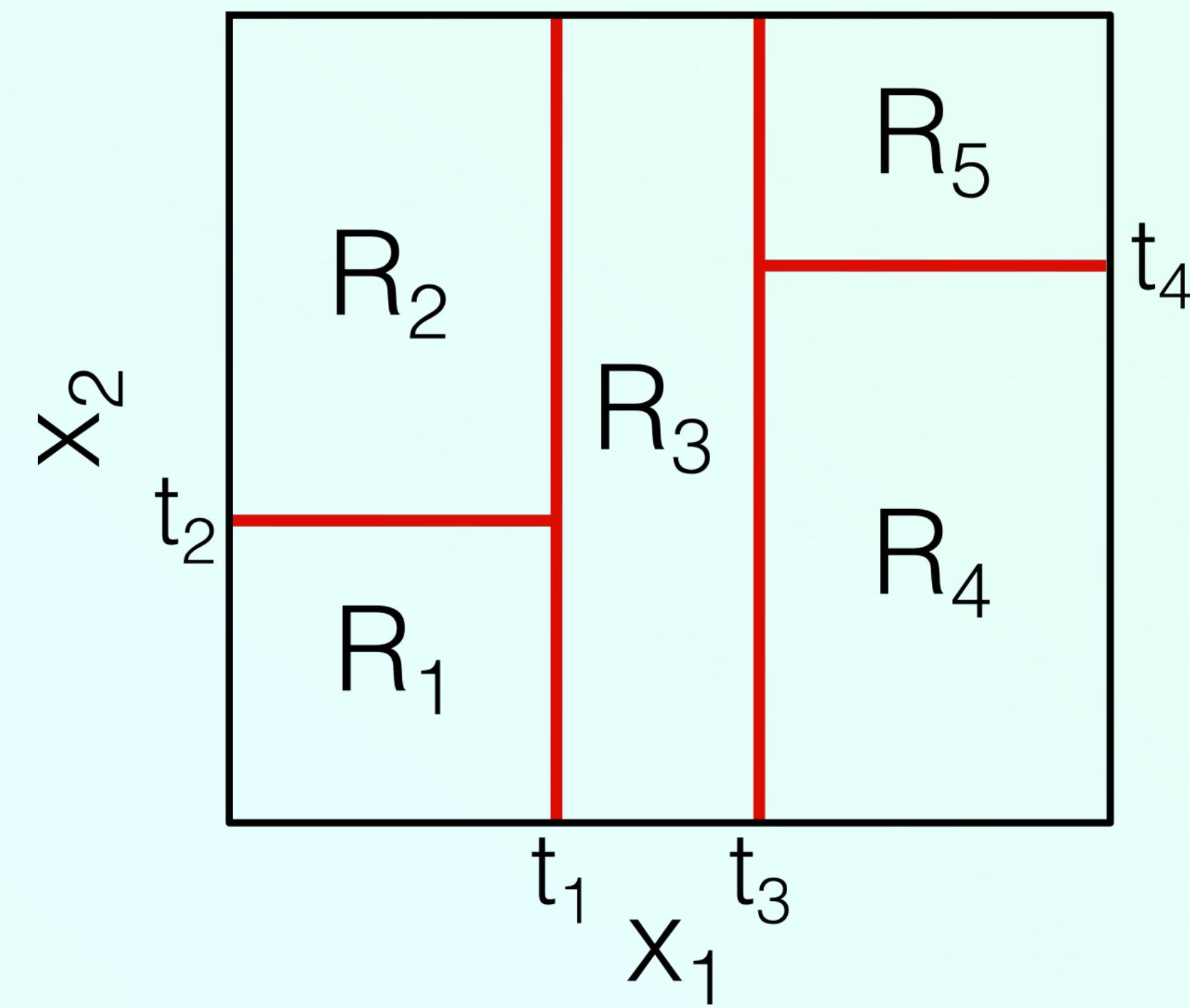
ARBOLES DE REGRESIÓN



ARBOLES DE REGRESIÓN

Recursive binary splitting

$$RSS = \sum_{j=1}^J \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$



ARBOLES DE REGRESIÓN

Podando los árboles

Introducimos un valor de penalización α a nuestra formula de RSS. Dado un valor de α , existe un sub-árbol T perteneciente a T_0 que:

$$\sum_{m=1}^L \sum_{i \in R_j} (y_i - \hat{y}_{R_m})^2 + \alpha L$$

...es mínimo.

L indica el número de hojas del sub-árbol.

α ($\alpha \geq 0$) presenta un trade-off entre complejidad del árbol y su capacidad de ajustar a los datos (Regularización).

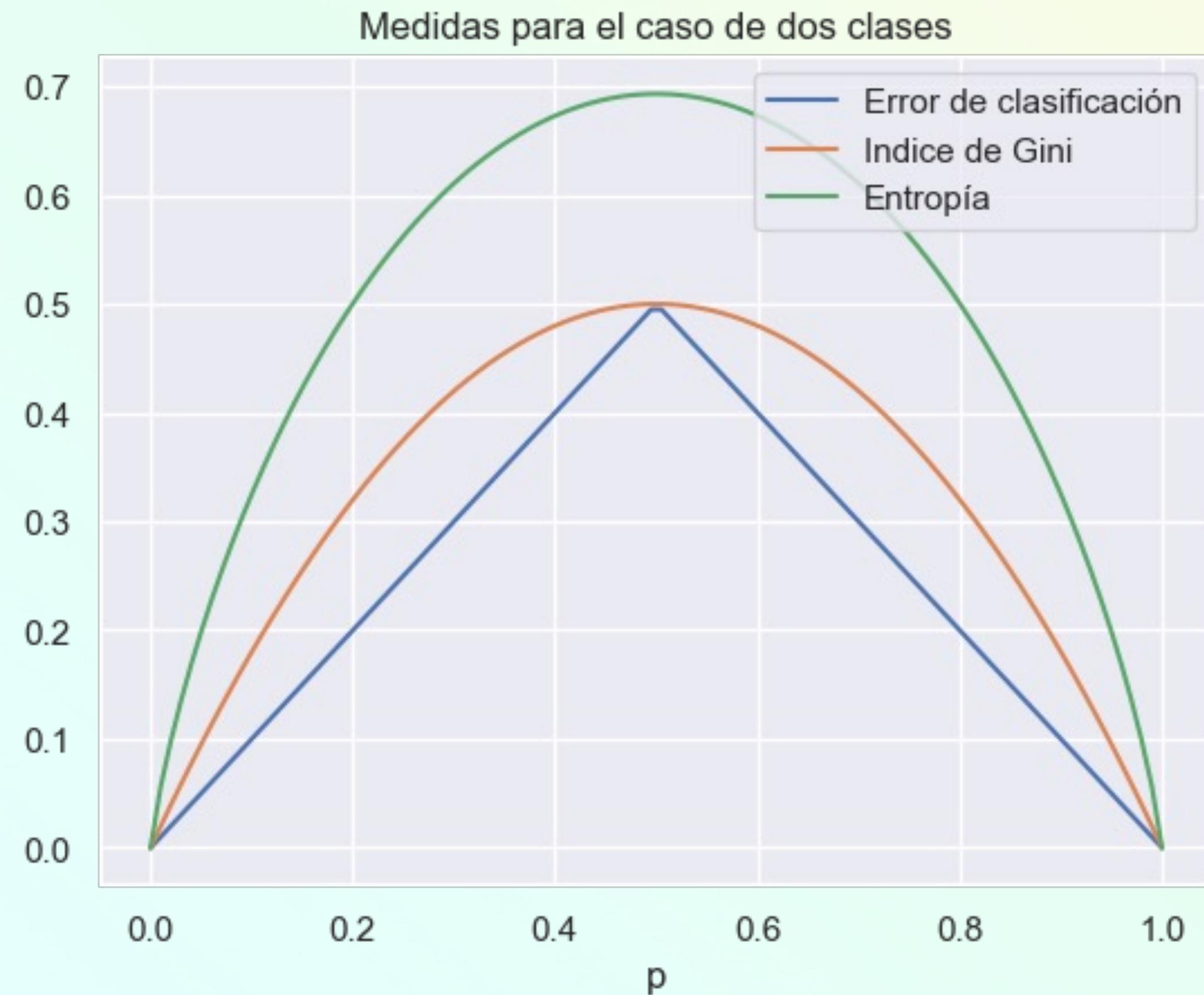
ARBOL DE CLASIFICACIÓN

Un árbol de clasificación es muy similar a uno de regresión, pero ahora se usa para predecir una **variable cualitativa**.

Obtenemos la clase en base a la **clase que más ocurre** en las muestras que están en la hoja.

Al interpretar los resultados de un **árbol de clasificación**, a menudo estamos interesados no sólo en la predicción de clase correspondiente a una región de nodo terminal particular, sino también en las **proporciones de clase entre las observaciones de entrenamiento** que caen en esa región.

ARBOL DE CLASIFICACIÓN



REDES NEURONALES

REDES NEURONALES

Las redes neuronales originalmente se plantean como algoritmos matemáticos que tratan de imitar los cálculos complejos que tienen lugar en el cerebro.

Se busca imitar no solo la gran cantidad de unidades de procesamiento (neuronas) sino la interconexión entre ellas (sinapsis).

Esto generó dos grandes campos de aplicación, uno el de la neurociencia computacional y por otro el de **Deep Learning**. Siendo este último el gran popular de hoy en día con avances que vemos constantemente en el público general.

REDES NEURONALES

Un poquito de historia...

Las redes neuronales podemos dar comienzo con la neurona de **McCulloch-Pitts** en 1943. Este modelo es la primera formulación del proceso de cálculo que lleva a cabo una neurona, basado en una formulación algebraica. La unidad actual usada de **Deep Learning** no es muy diferente a este modelo.

En 1952, Hodgkin y Huxley crean el modelo basado en conductancia de la neurona, que modela como los potenciales de acción de las neuronas se generan y se propagan. Es un modelo de ecuaciones diferenciales. Los resultados fueron tan importantes que se llevaron el premio Nobel de medicina en 1963.

Este modelo que describen como funciona la neurona siguieron camino al desarrollo de la **neurociencia computacional**.

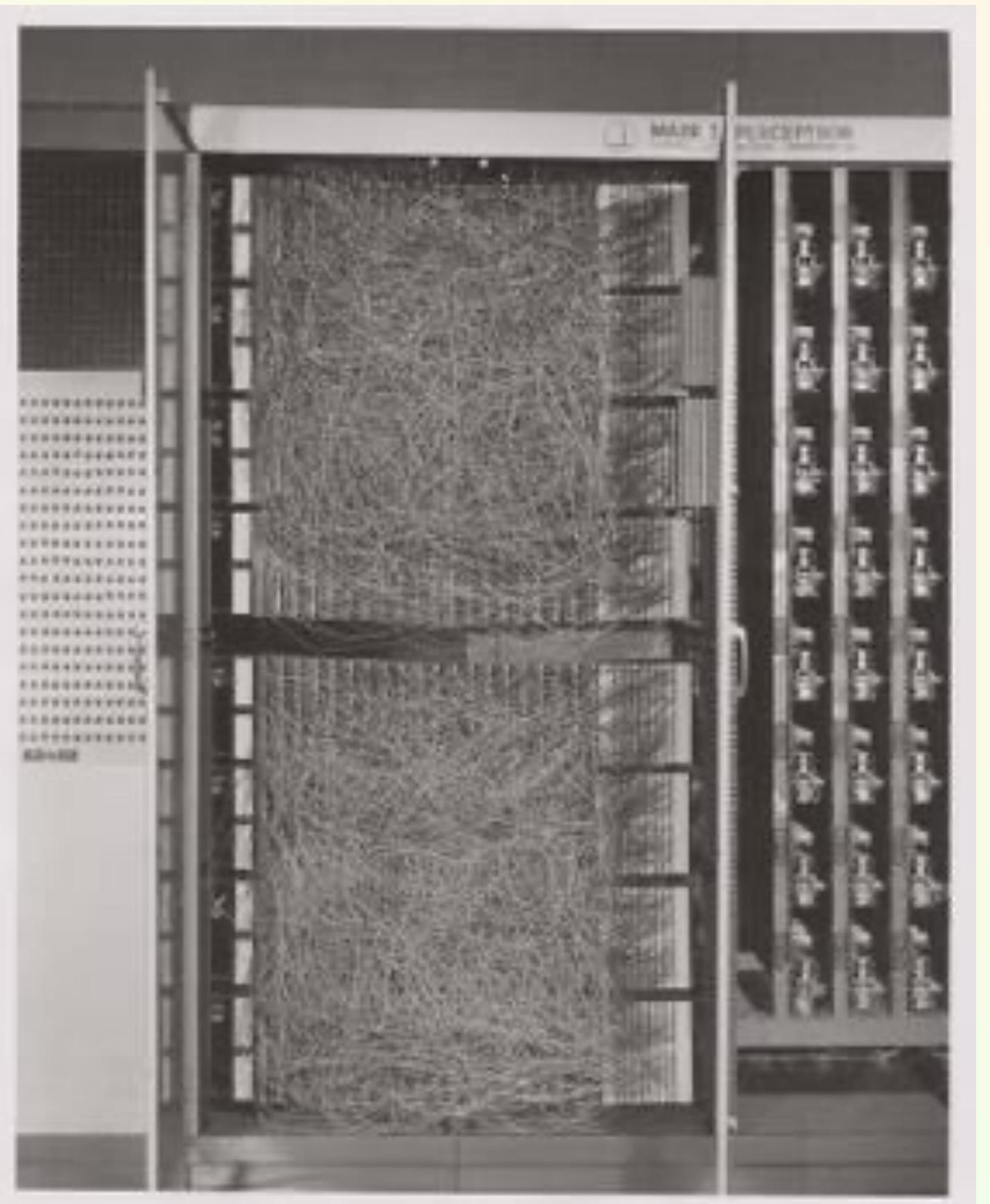
REDES NEURONALES

Lo importante es que, en este estadio embrionario del área, es que las investigaciones estaban en su cúspide.

En 1958, Rosenblatt realizó la primera implementación del perceptrón (basado en la neurona de McCulloch-Pitts).

Rosenblatt es el padre del Deep Learning.

Es quien perfeccionó el perceptrón moderno, y las redes de dos o tres capas.



REDES NEURONALES

Pero en 1969 llegó el **primer invierno** en redes neuronales. En 1969 se publicó un libro llamado Perceptrons de Minsky y Papert que enfatizaba los límites de lo que los perceptrones podían hacer.

Este libro y su popularidad mató toda financiación en investigaciones hasta 1980.

Entre muchas críticas, se planteó que el **perceptrón** como modelo neuronal **no** podía resolver la función lógica XOR, algo que una neurona biológica si podría...

REDES NEURONALES

Pero en 1969 llegó el **primer invierno** en redes neuronales. En 1969 se publicó un libro llamado Perceptrons de Minsky y Papert que enfatizaba los límites de lo que los perceptrones podían hacer.

Este libro y su popularidad mató toda financiación en investigaciones hasta 1980.

Entre muchas críticas, se planteó que el **perceptrón** como modelo neuronal **no** podía resolver la función lógica XOR, algo que una neurona biológica si podría...

Luego en experimentos fisiológicos se probó que la neurona biológica tampoco puede resolver la función lógica XOR, sino que necesita de una red.

REDES NEURONALES

Pero en 1969 llegó el primer invierno en redes neuronales. En 1969 se publicó un libro llamado Perceptrons de Minsky y Papert que enfatizaba los límites de lo que los perceptrones podían hacer.

Este libro y su popularidad mató toda financiación en investigaciones hasta 1980.

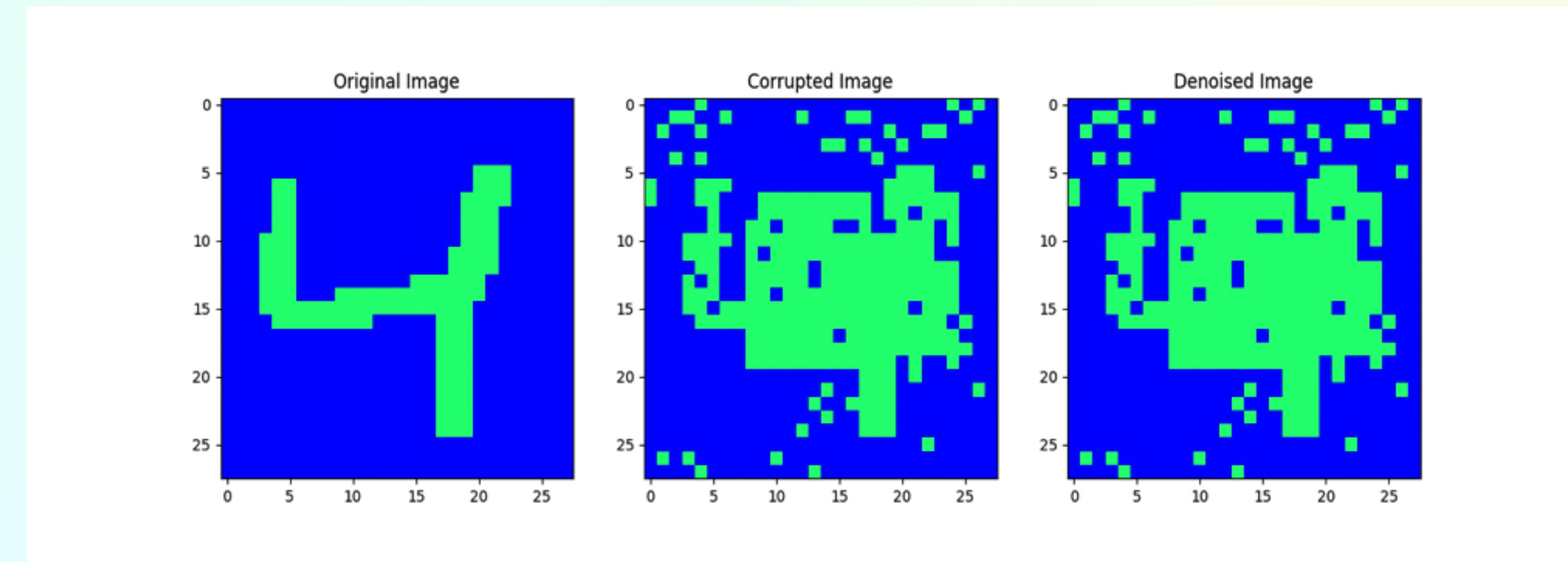
Entre muchas críticas, se planteó que el **perceptrón** como modelo neuronal **no** podía resolver la función lógica XOR, algo que una neurona biológica si podría...

Luego en experimentos fisiológicos se probó que la neurona biológica tampoco puede resolver la función lógica XOR, sino que necesita de una red.

The plot thickens (la trama se complica)... En 2020 se encontró que neuronas del cerebro humano si pueden.

REDES NEURONALES

El invierno termina en 1980-90 con los desarrollos de **Rumelhart**, **Hopfield**, entre otros. Aquí se desarrollaron el algoritmo de Back-propagation, redes de memoria y el concepto de propiedades emergentes.



Obtenido de <https://github.com/TarinZ/hopfield-nets>

REDES NEURONALES

El invierno termina en 1980-90 con los desarrollos de **Rumelhart**, **Hopfield**, entre otros. Aquí se desarrollaron el algoritmo de Back-propagation, redes de memoria y el concepto de propiedades emergentes.

Luego llegó **un nuevo invierno** de redes neuronales con la llegada de algoritmos muchos más sencillos de usar y sin tanta dificultad de retoques de hiper parámetros, tales como SVM y los bosques aleatorios. Estos algoritmos eran más fáciles de usar y rendían mucho mejor que las redes neuronales.

REDES NEURONALES

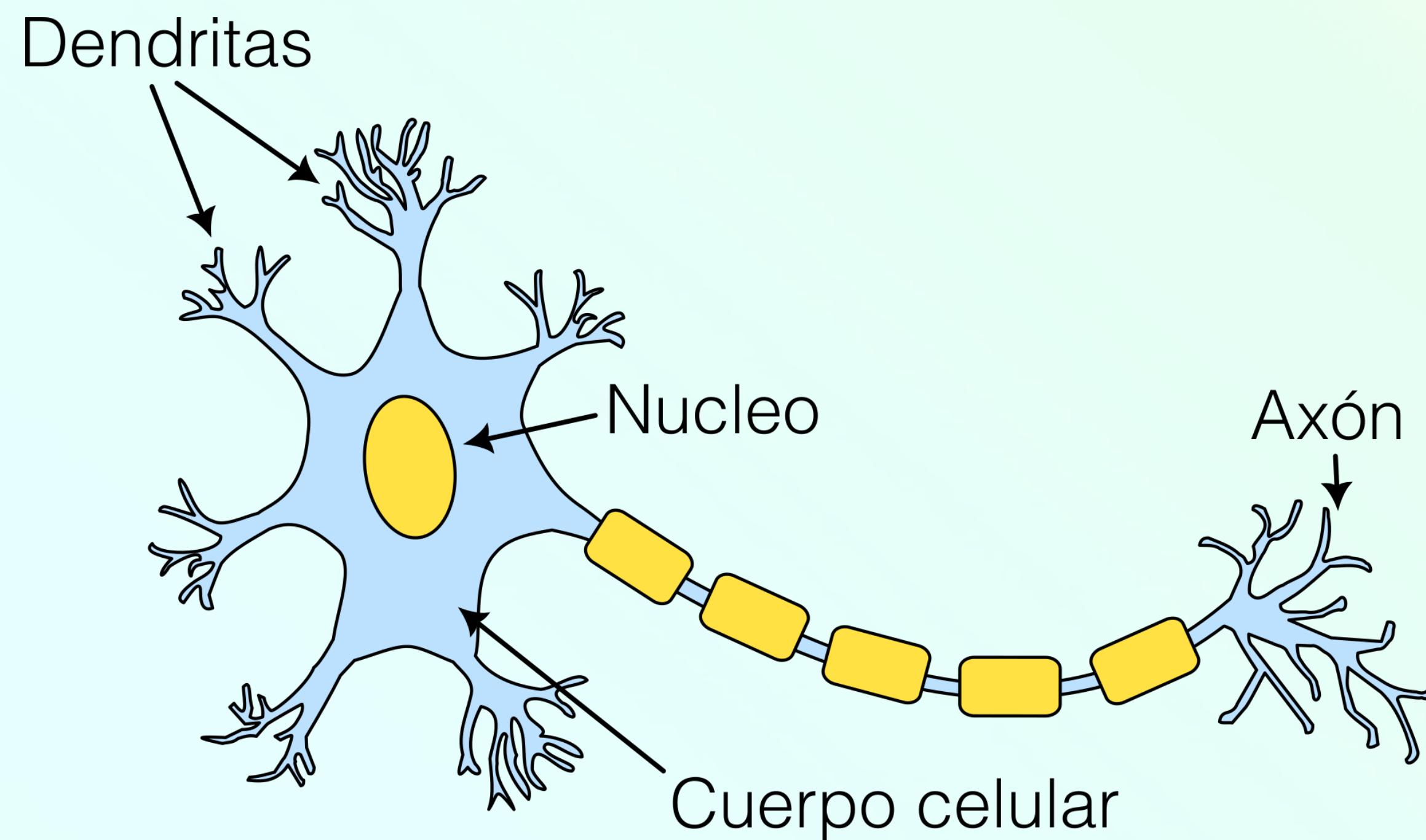
En 2010, volvió a la vida, y ahora, oficialmente se le empezó a llamar **Deep Learning**. Ahora se incorporaron nuevas arquitecturas de redes y características adicionales. Además, ahora estos algoritmos empezaron a superar a los demás en tareas de clasificación de imágenes y videos, y el modelado de voz y texto.

Parte del éxito es la posibilidad de contar con datasets más grandes (Big data) y mejores procesamientos de cálculo, además de la llegada de Google, Meta, Microsoft.

PERCEPTRON Y NEURONAS SIGMOIDEAS

PERCEPTRÓN

Empecemos con una neurona biológica:



Tenemos:

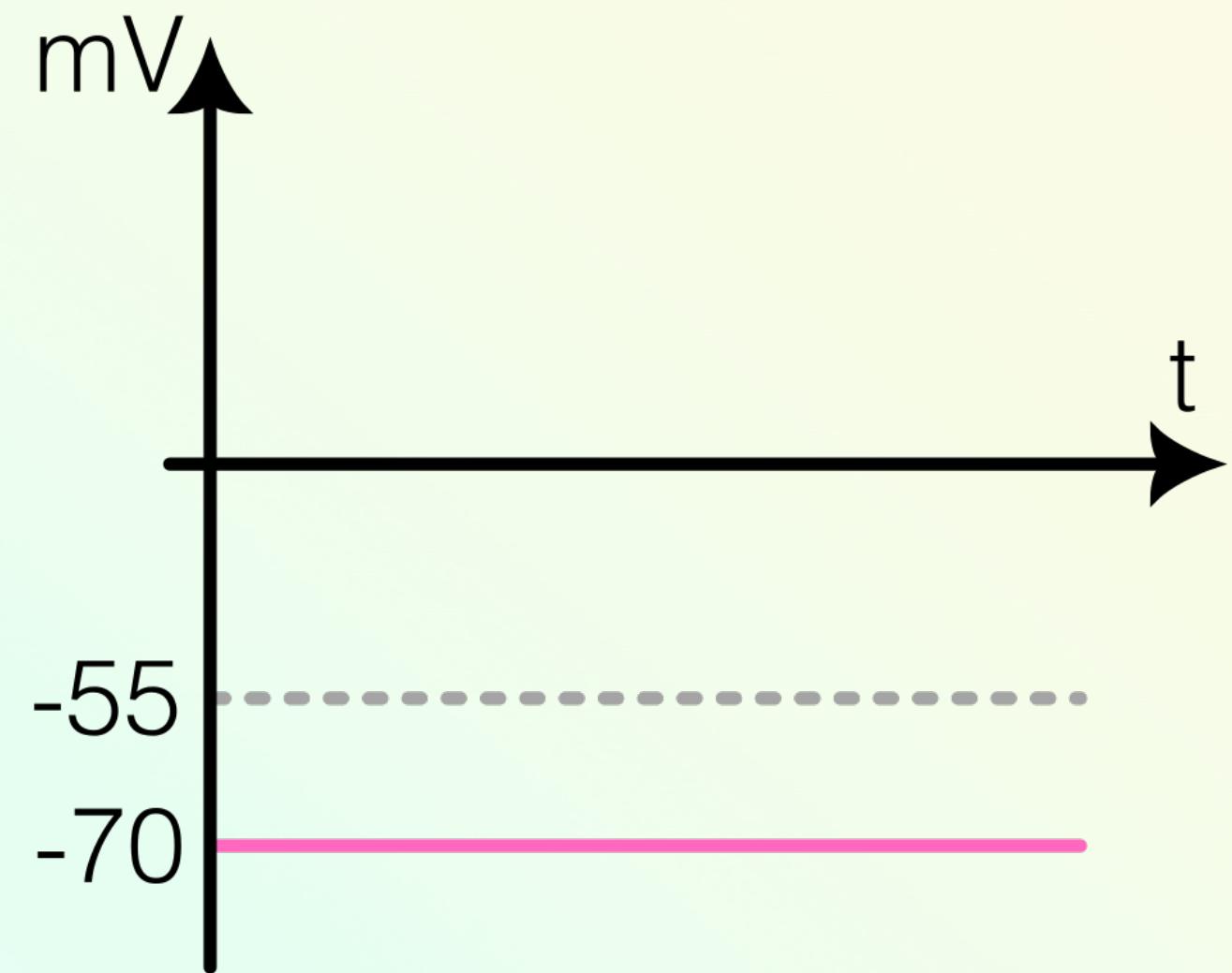
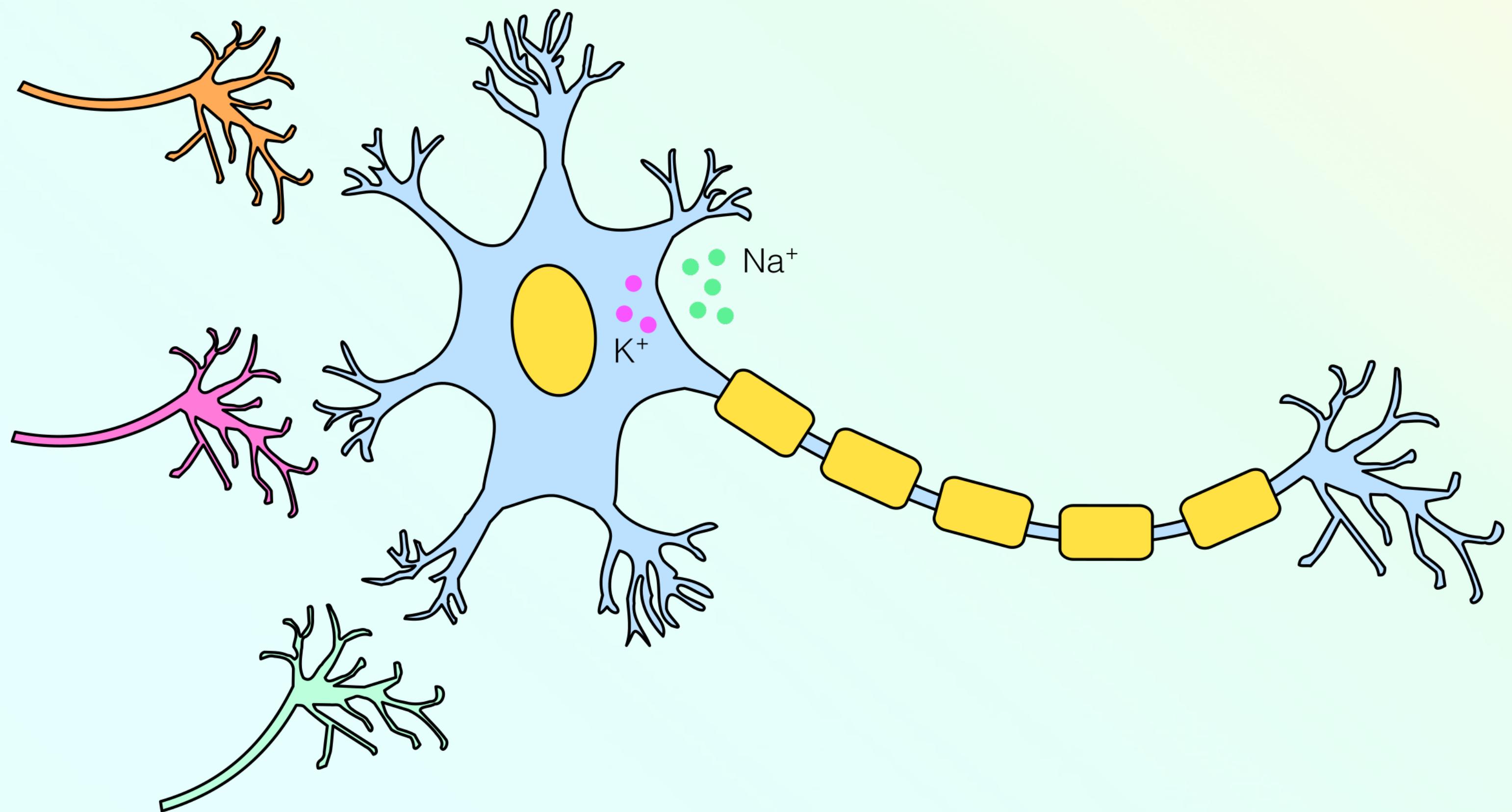
Cuerpo celular o soma: Donde está el núcleo y las organelas de la célula.

Dendritas: Ramificaciones del soma, es donde la neurona recibe las sinapsis de otras neuronas.

Axón: Es la prolongación encargada de transmitir el impulso nervioso. Es como se comunica la neurona.

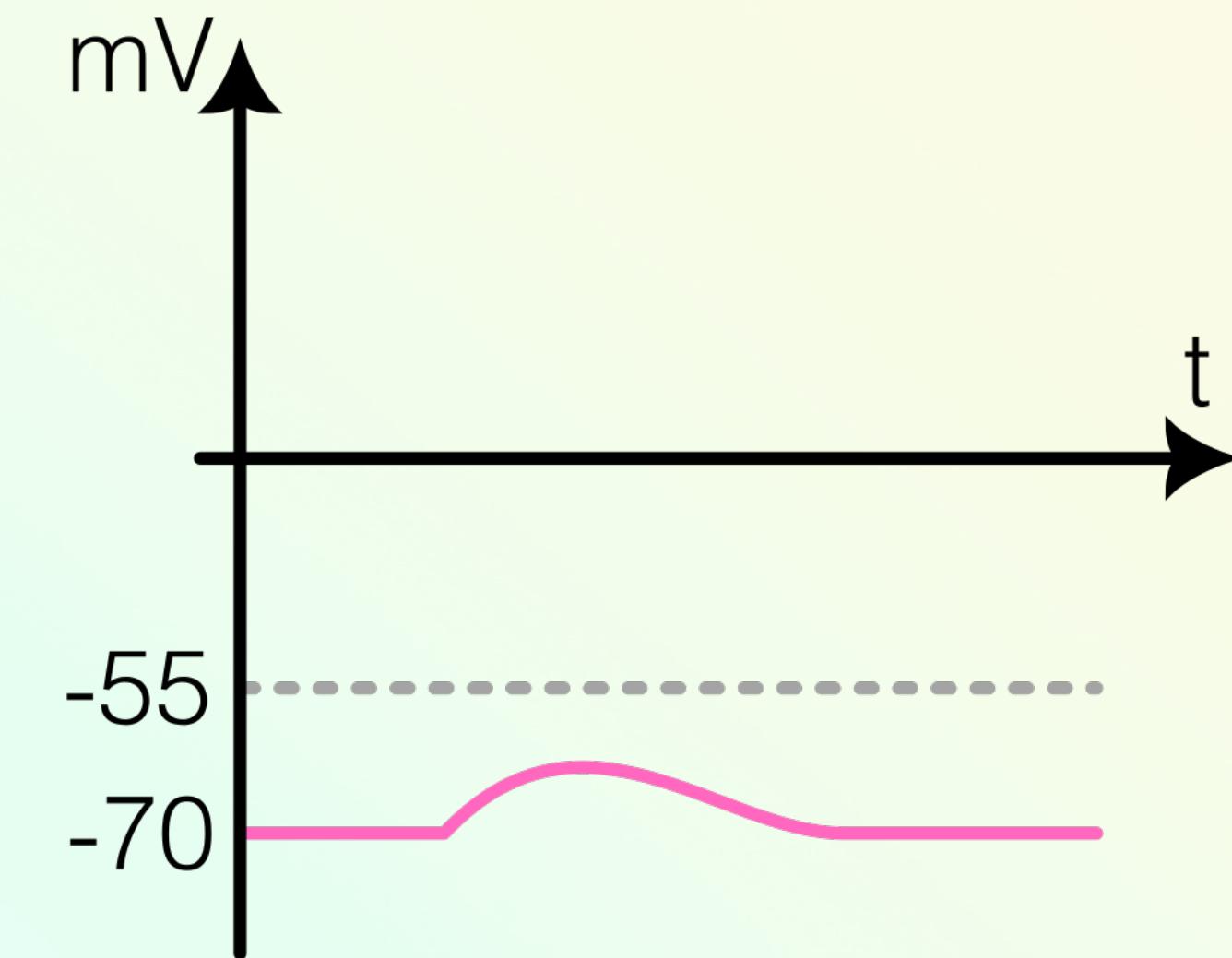
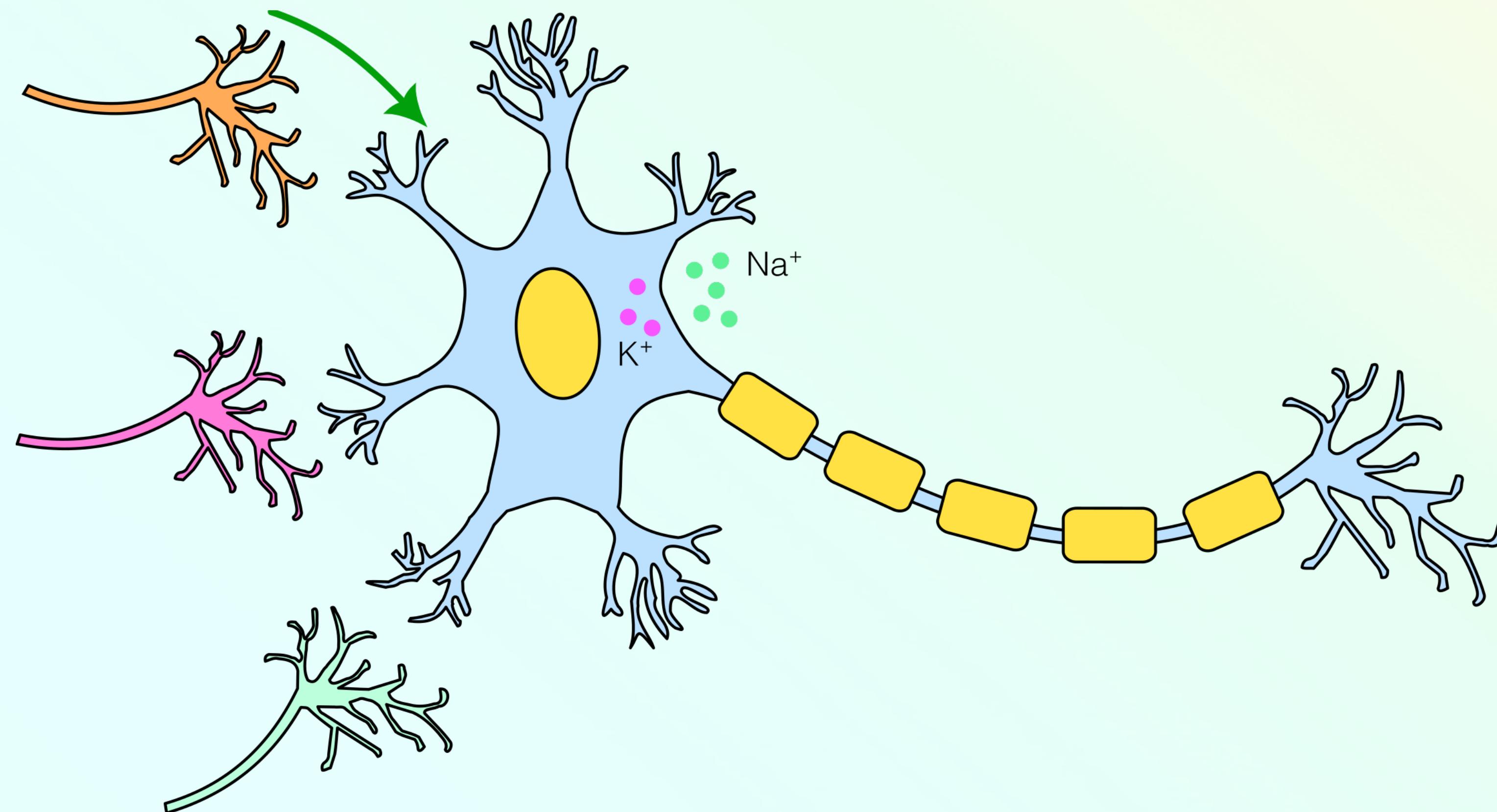
PERCEPTRÓN

En estado de reposo, tenemos...



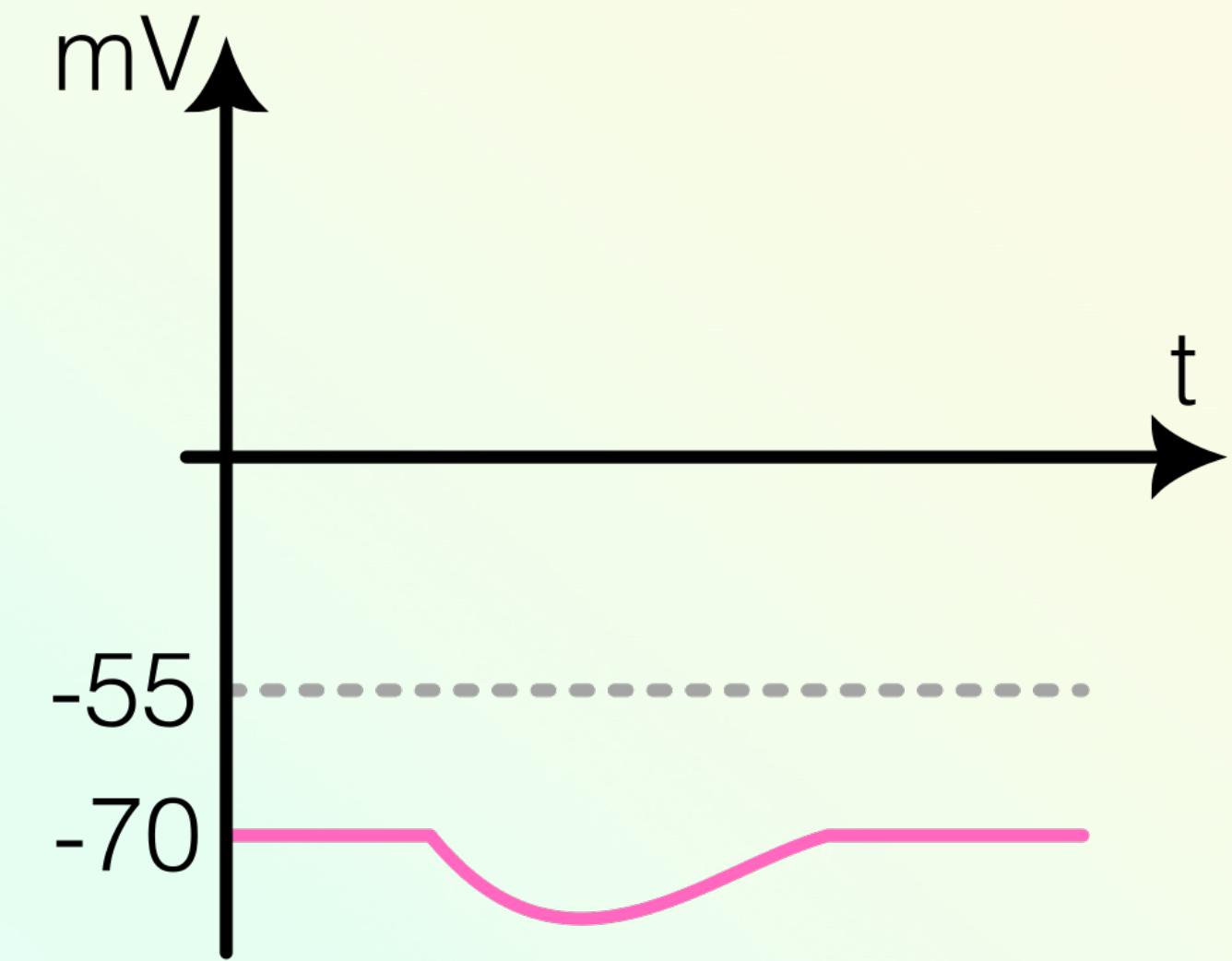
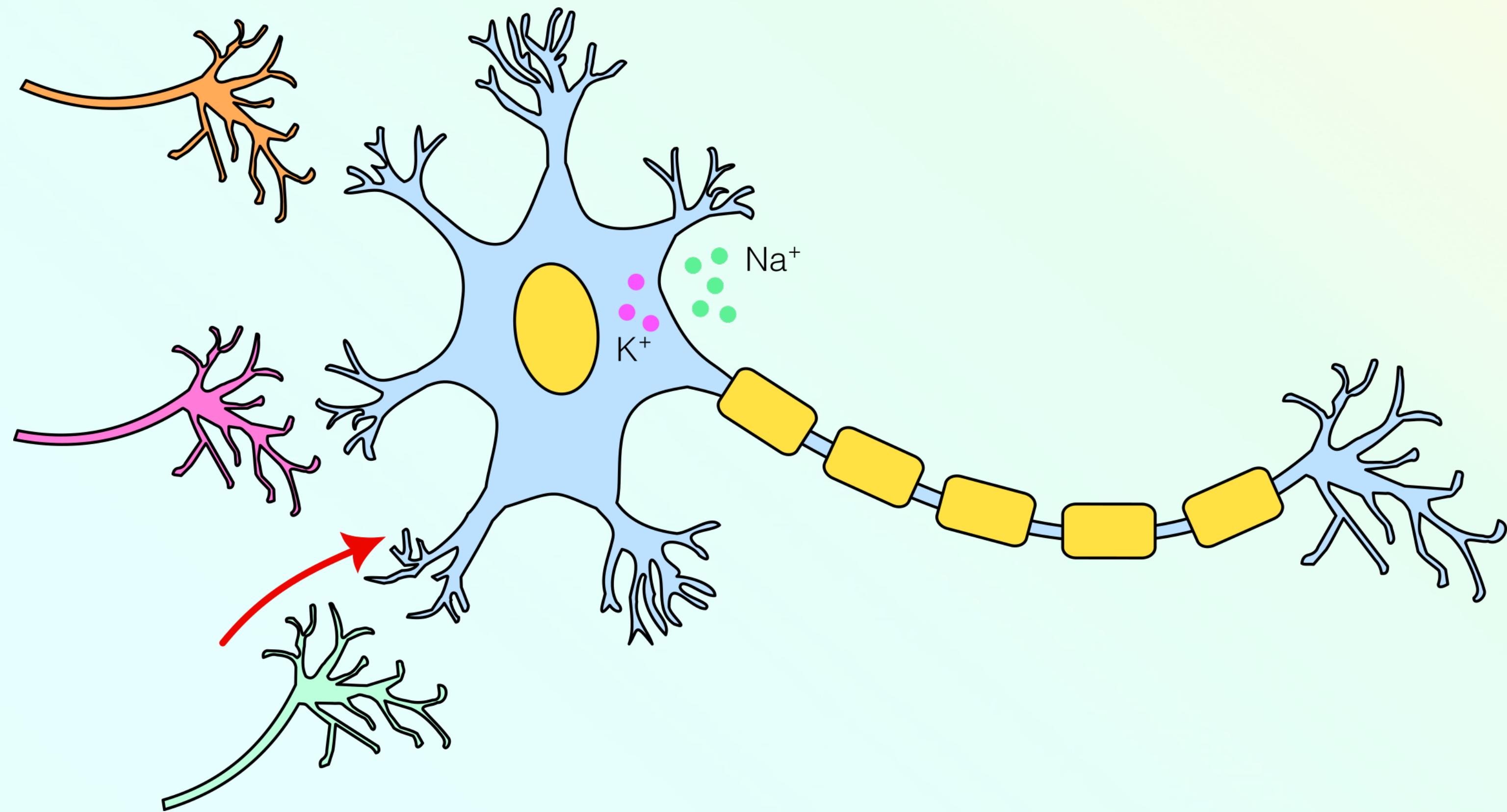
PERCEPTRÓN

Si otra neurona excita, pero poco, aumenta le voltaje, pero rápidamente vuelve a su estado.



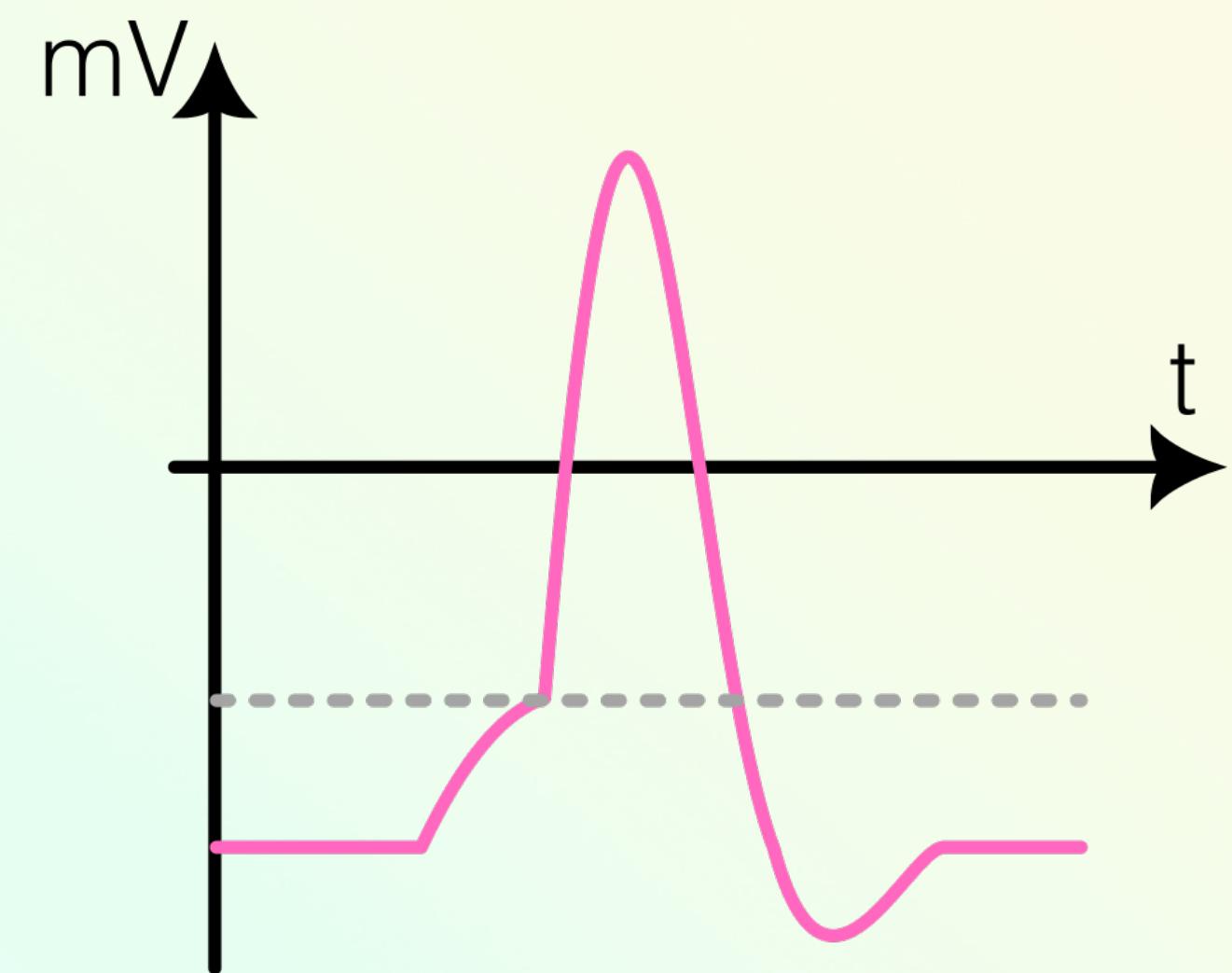
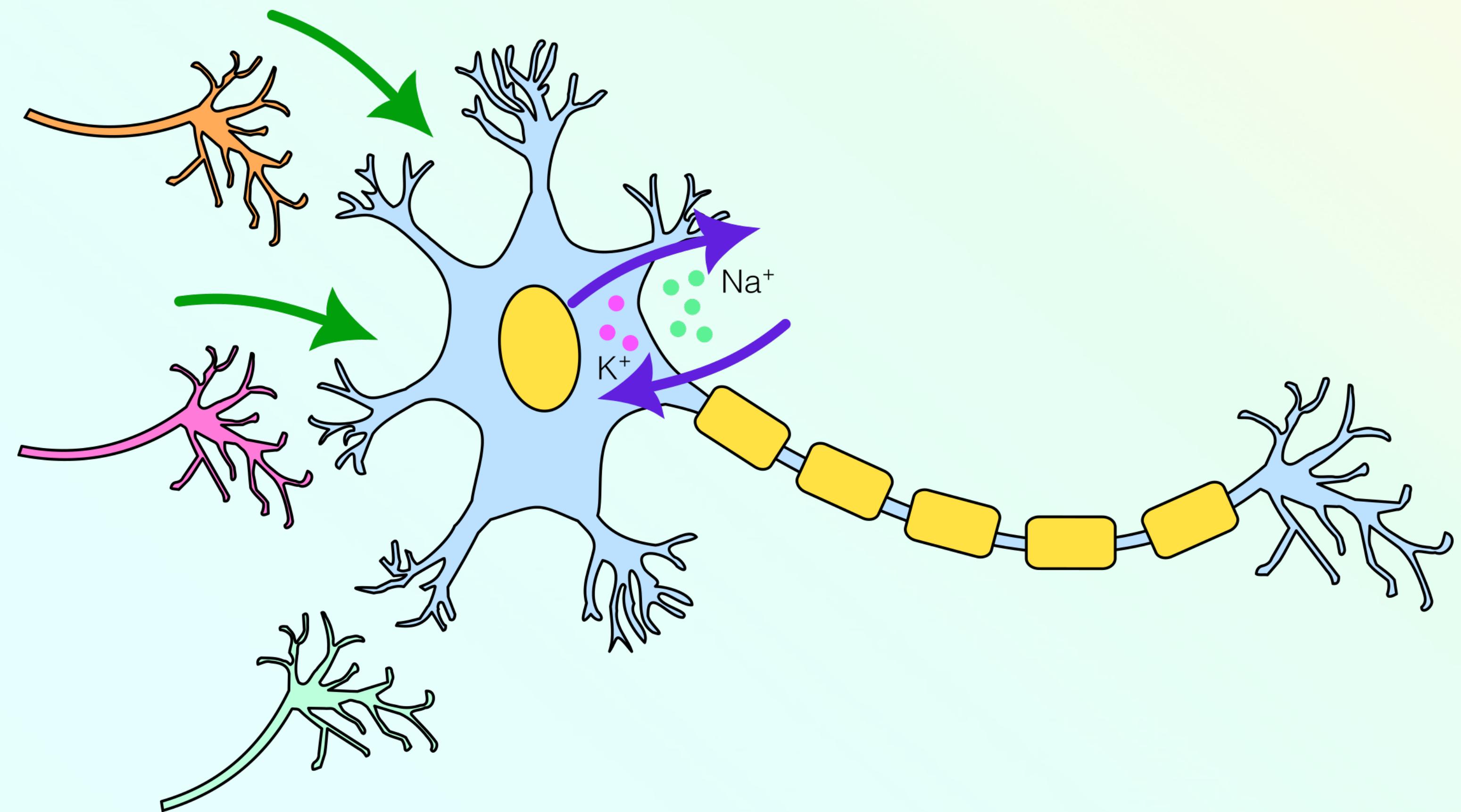
PERCEPTRÓN

De similar forma, una sinapsis puede ser inhibitoria



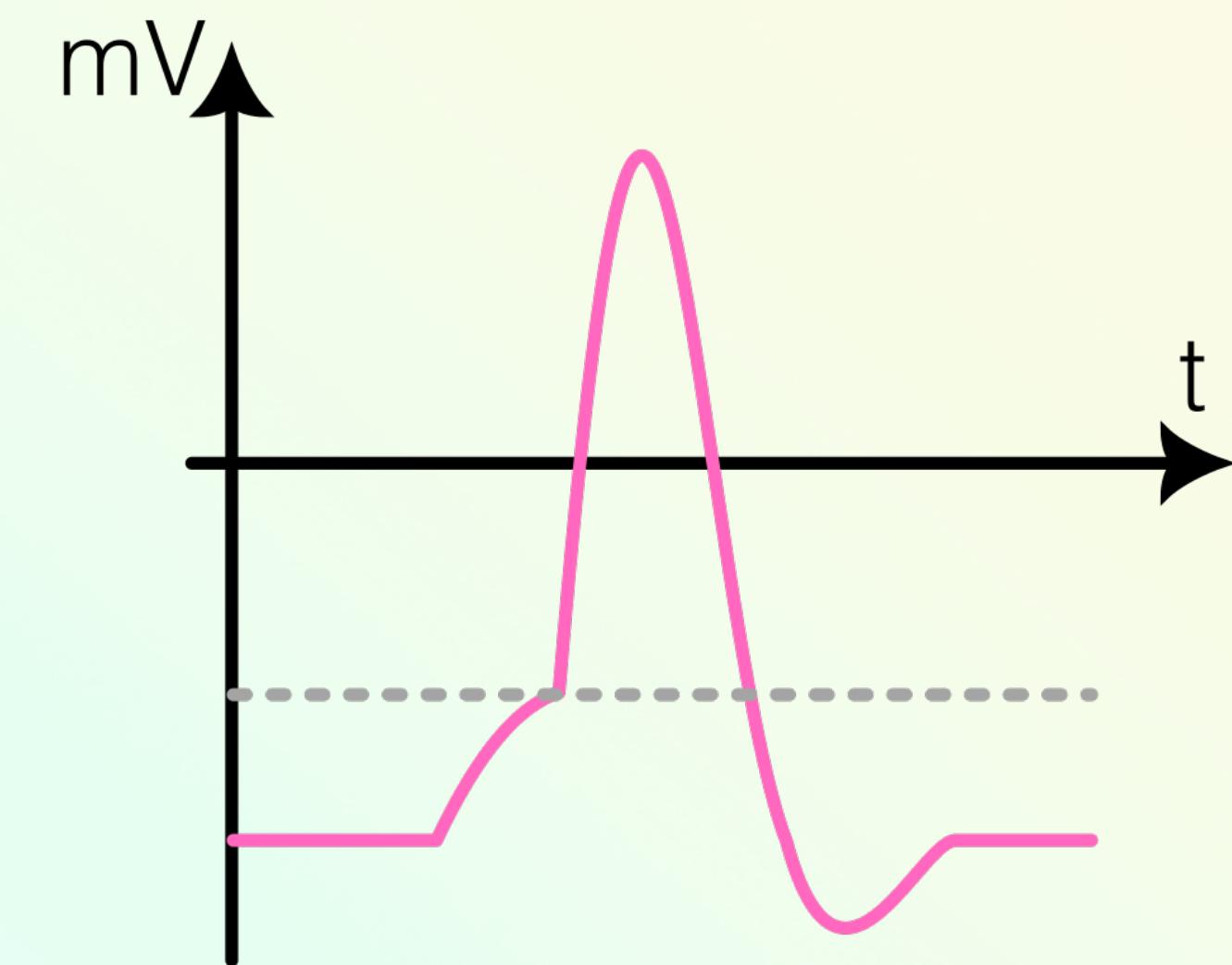
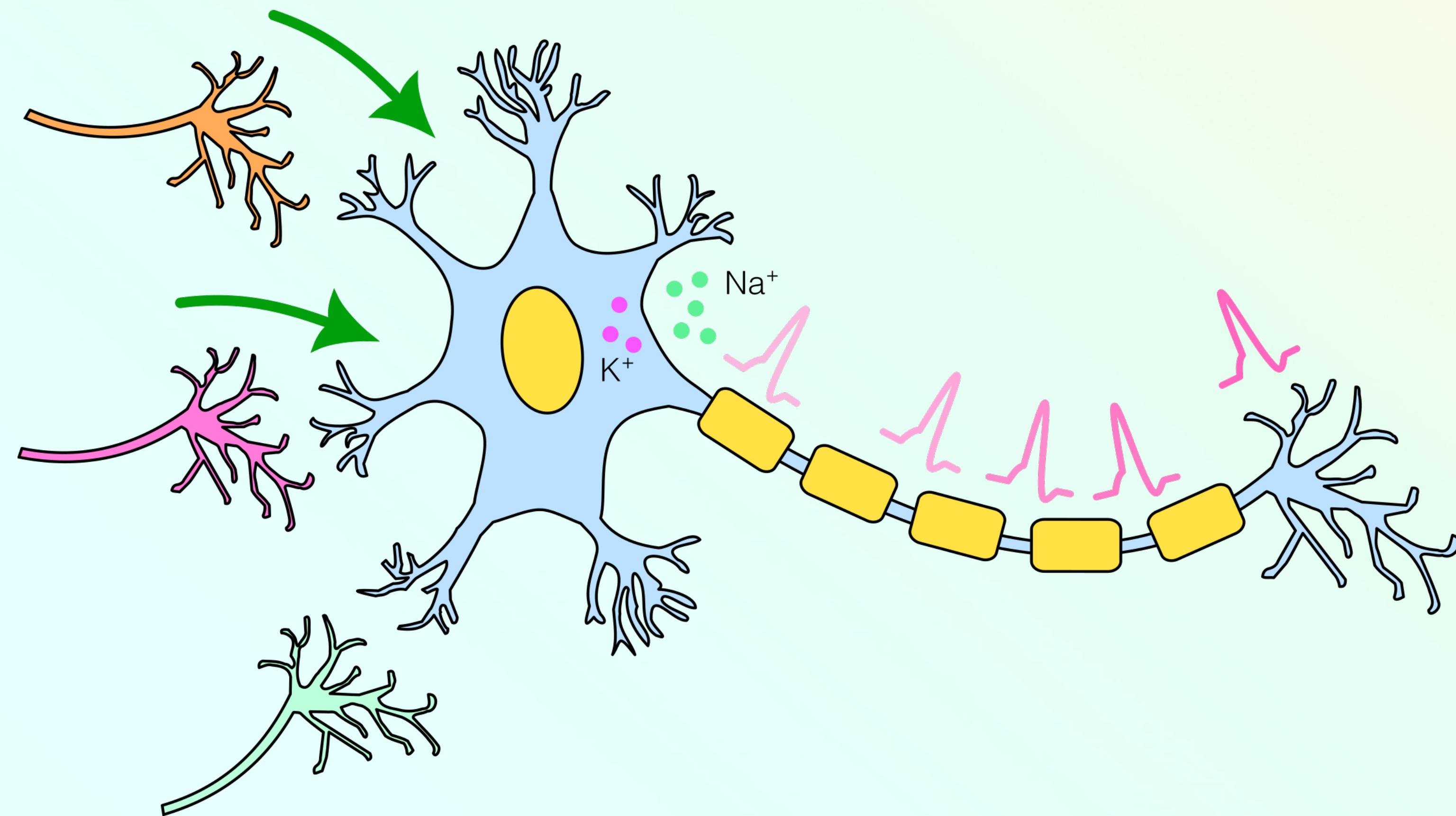
PERCEPTRÓN

Ahora, si la excitación supera el umbral, se genera un impulso dado un efecto **todo o nada**.



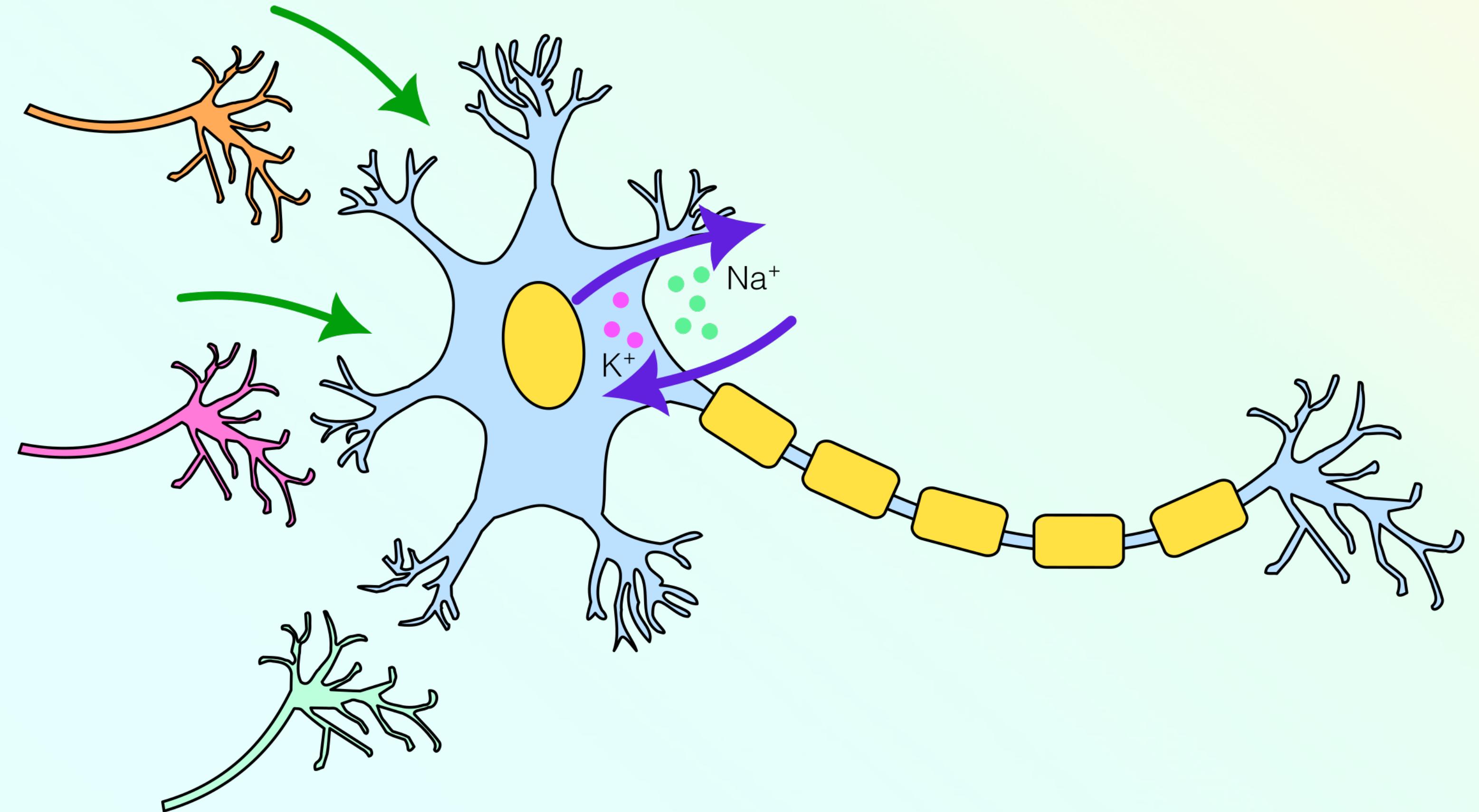
PERCEPTRÓN

Y este impulso se propaga del cuerpo hasta el terminal axónico, el cual la neurona da su salida.



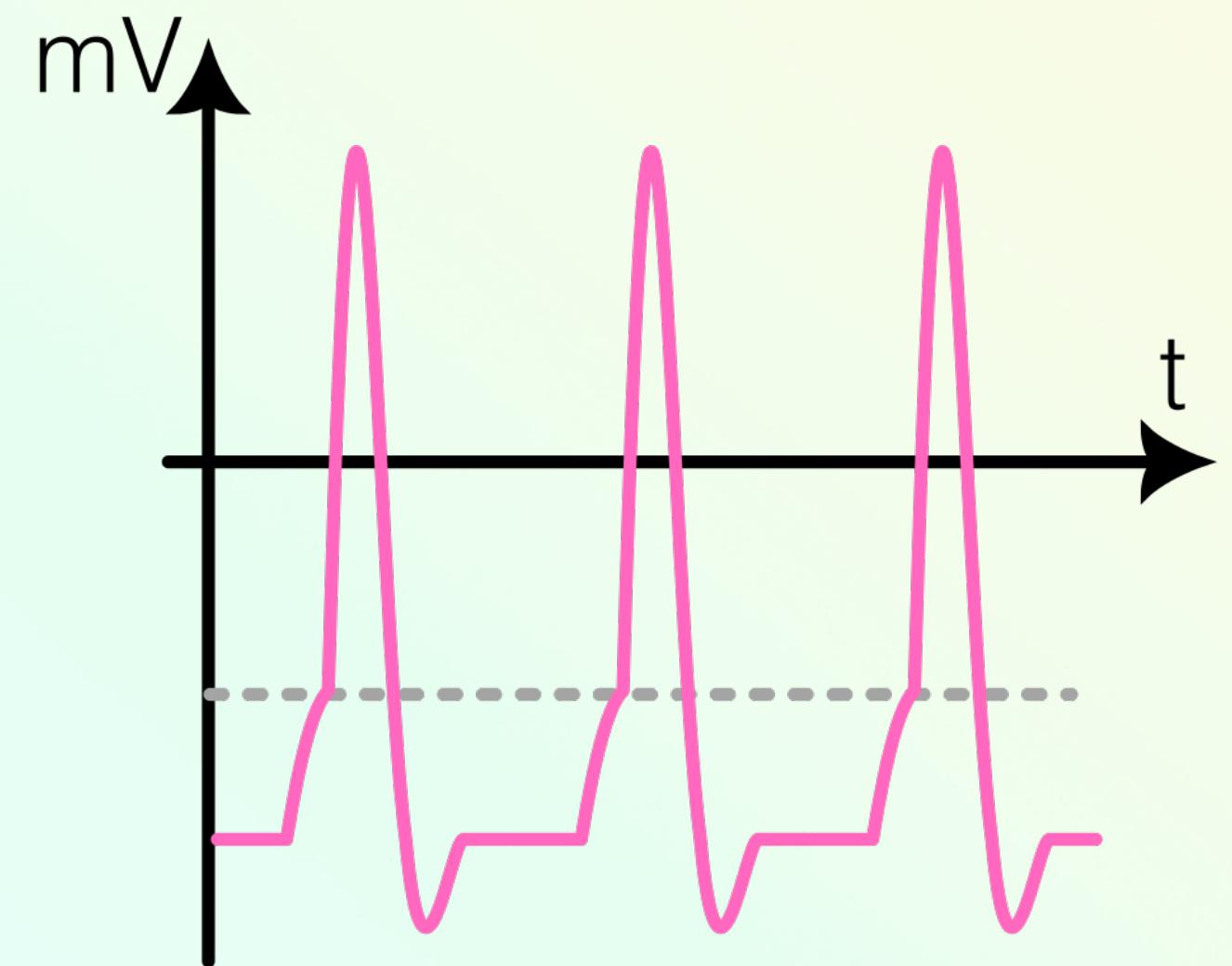
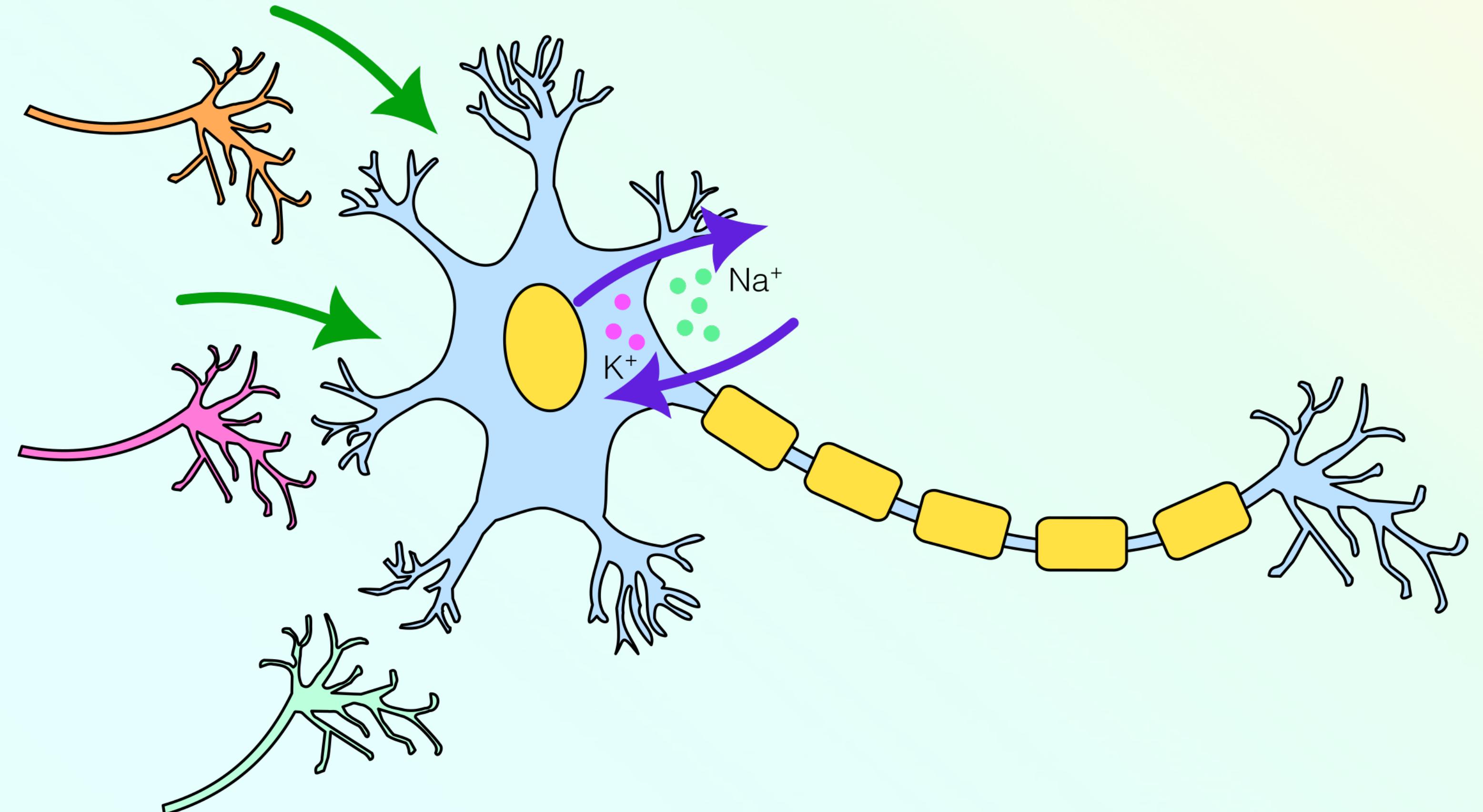
PERCEPTRÓN

Niveles de excitación nos dan diferentes tasas de disparo...



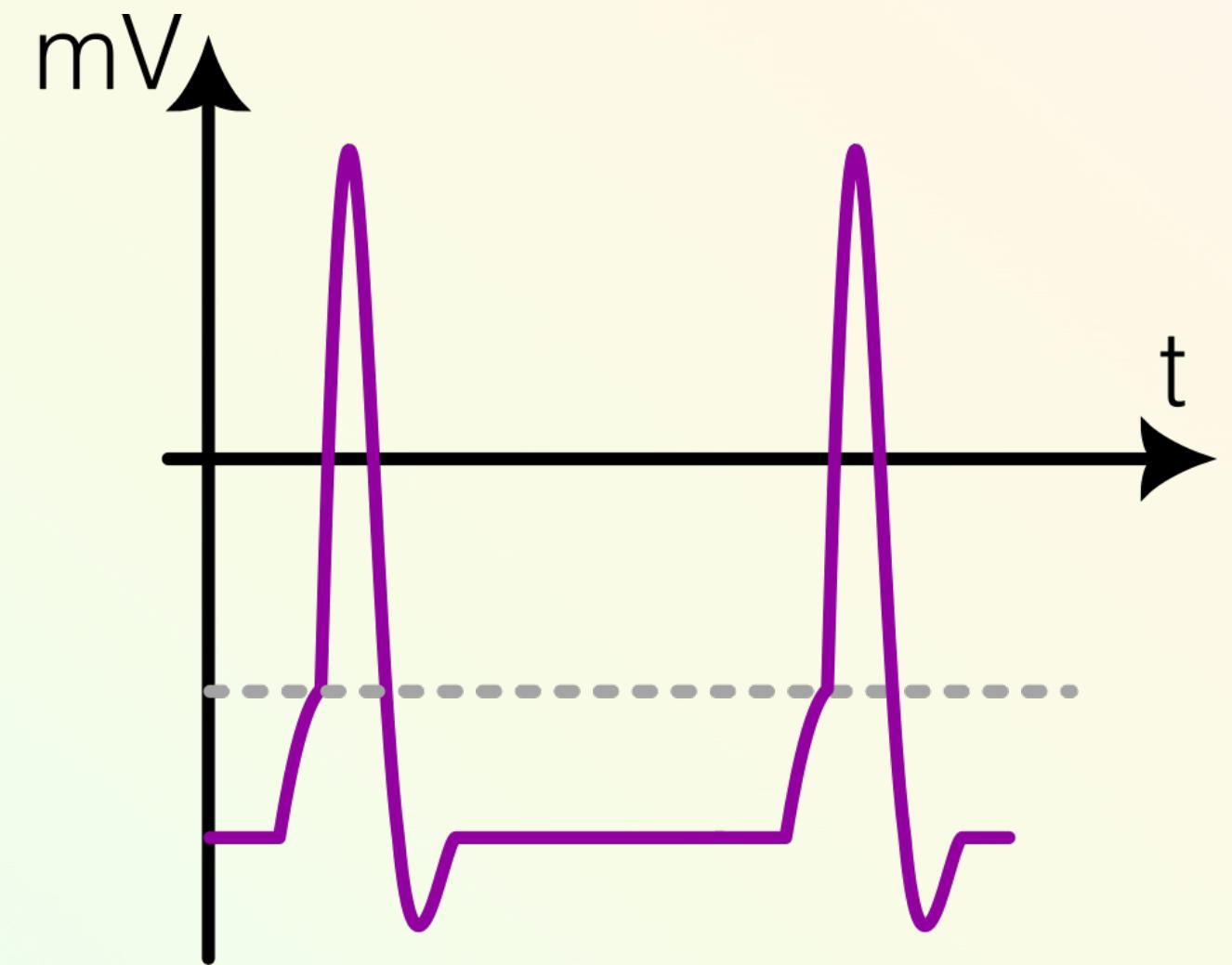
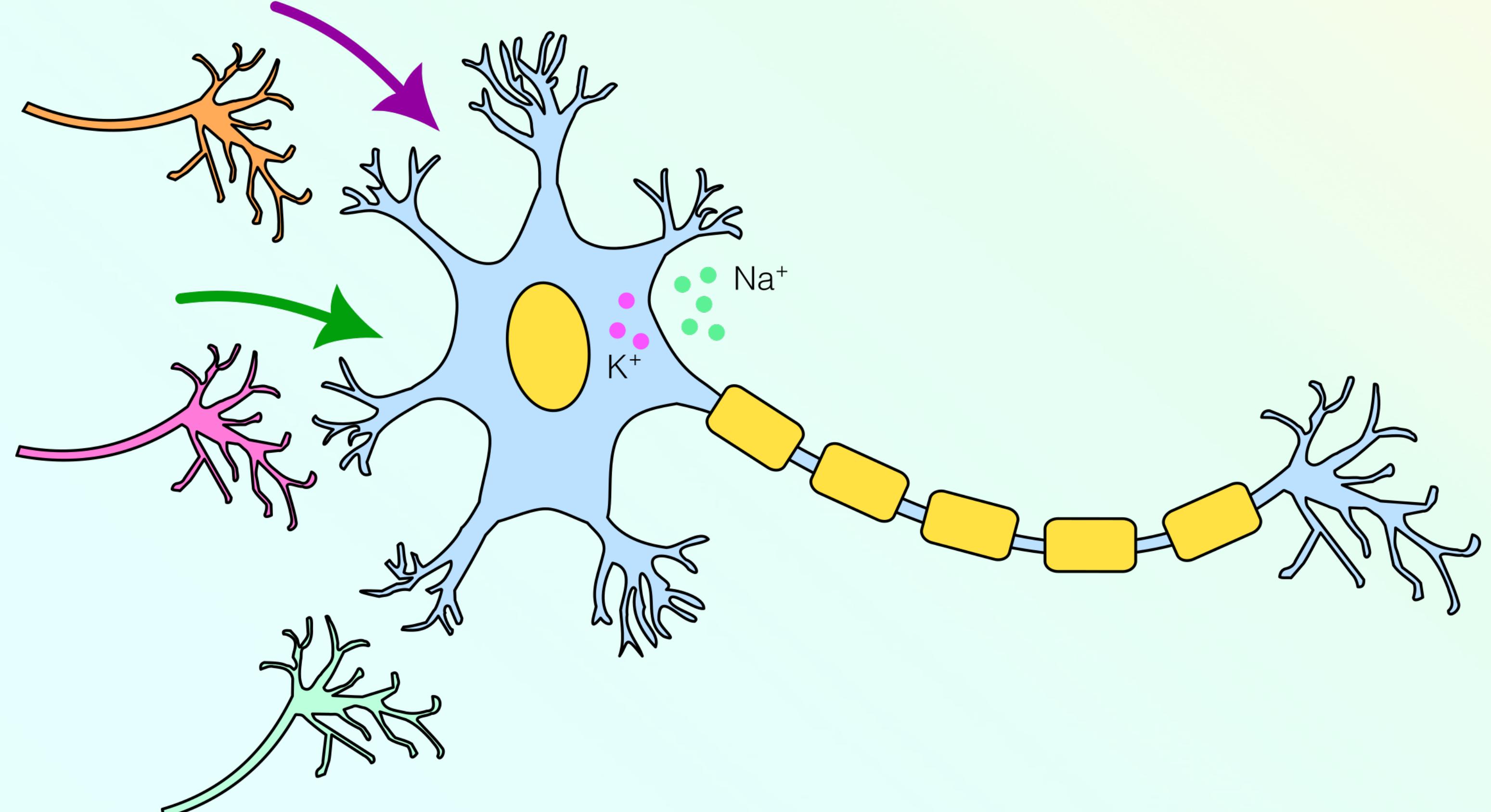
PERCEPTRÓN

Niveles de excitación nos dan diferentes tasas de disparo...



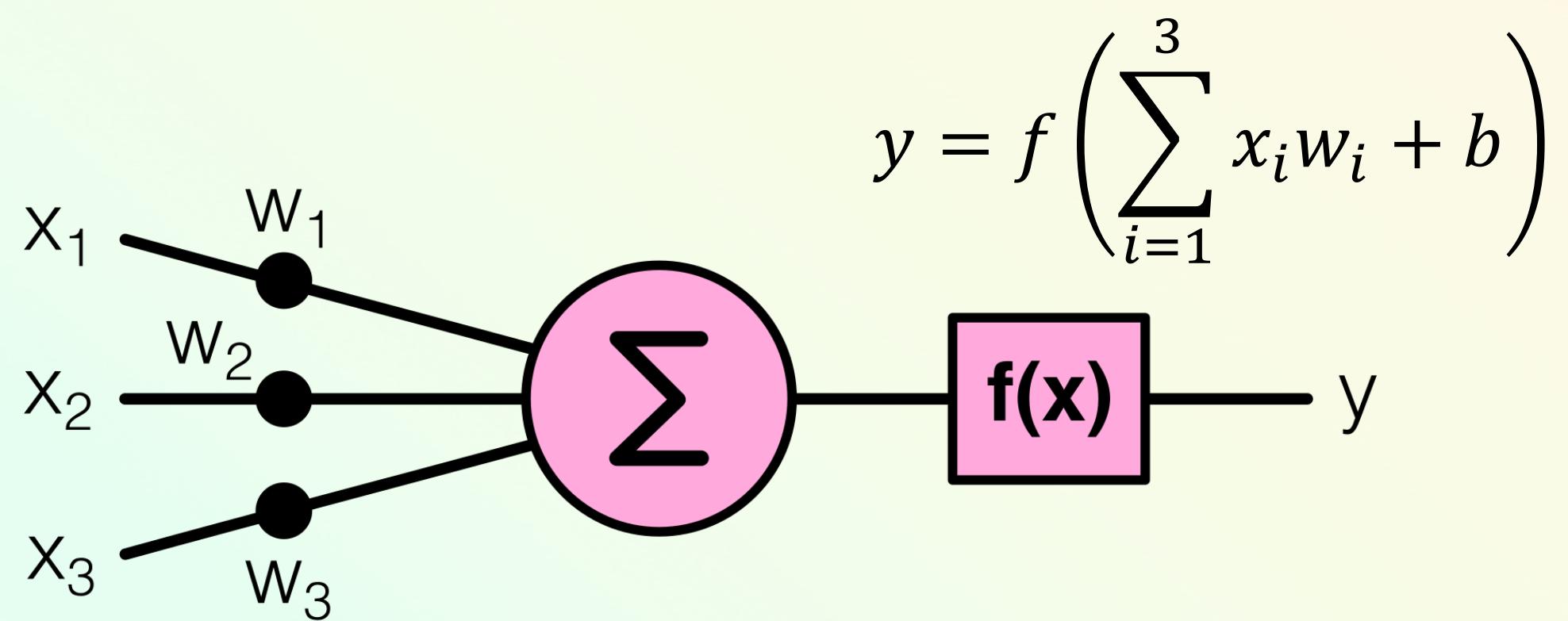
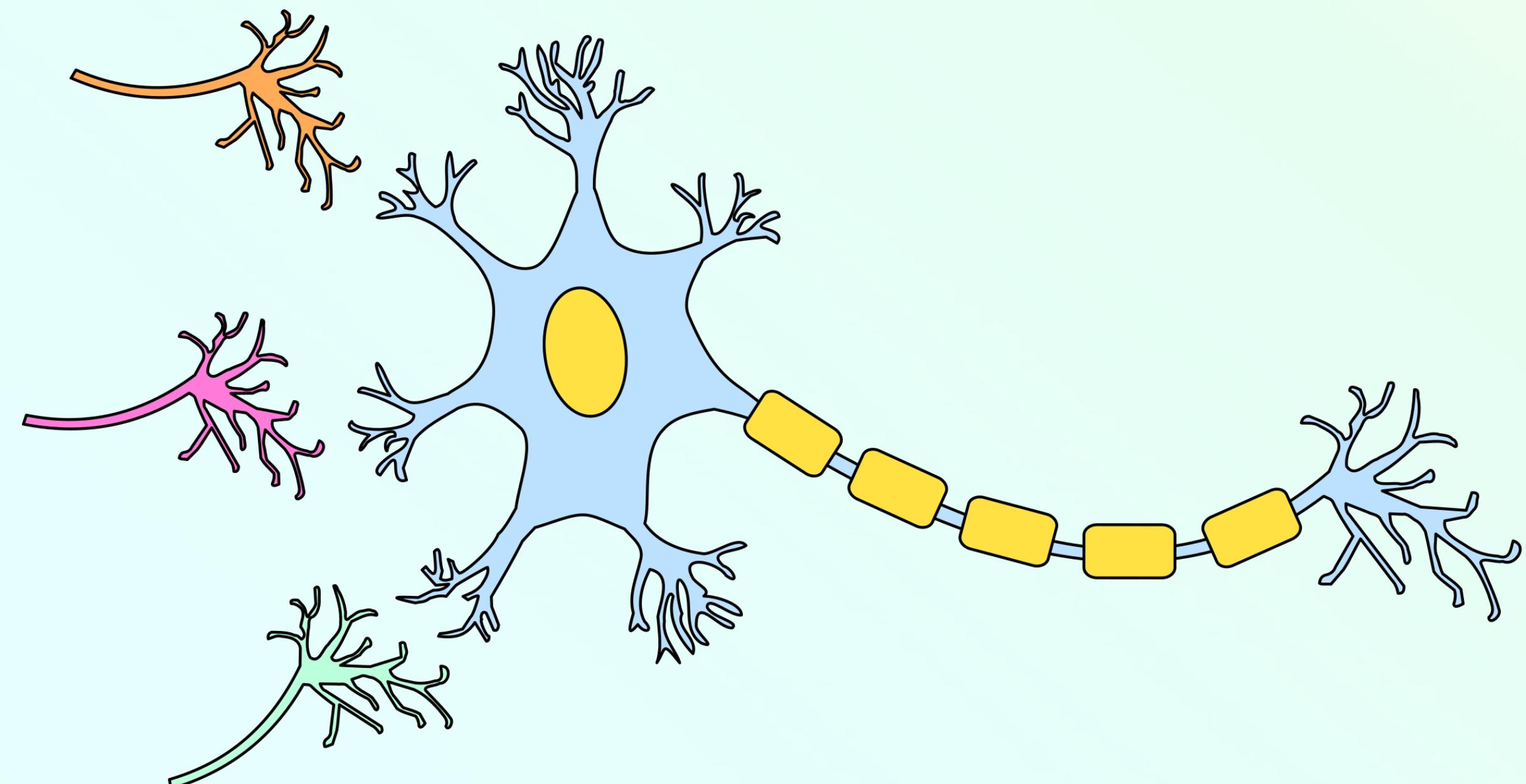
PERCEPTRÓN

Y ante una misma excitación, hay sinapsis más sensibles que otras



PERCEPTRÓN

Con esto en mente, armemos el modelo de neurona

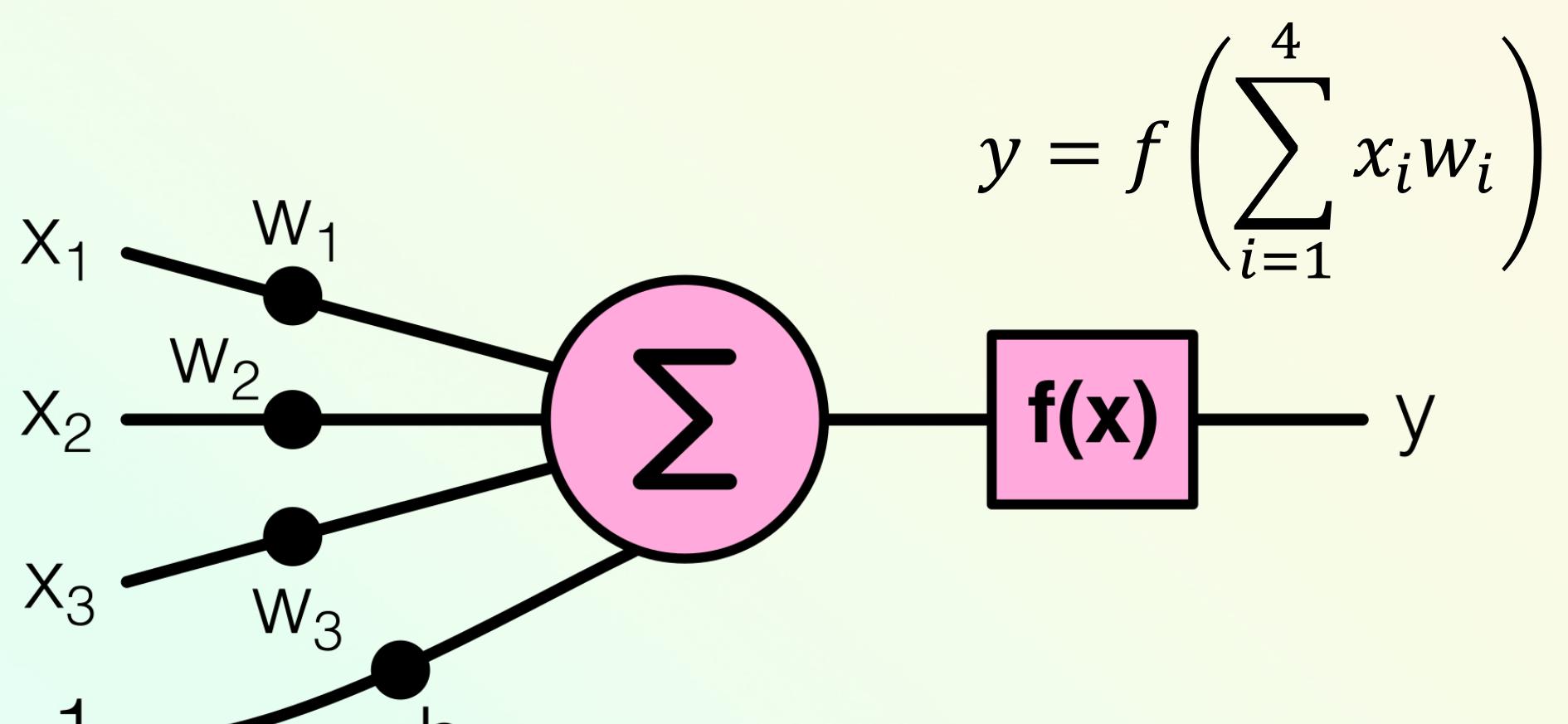
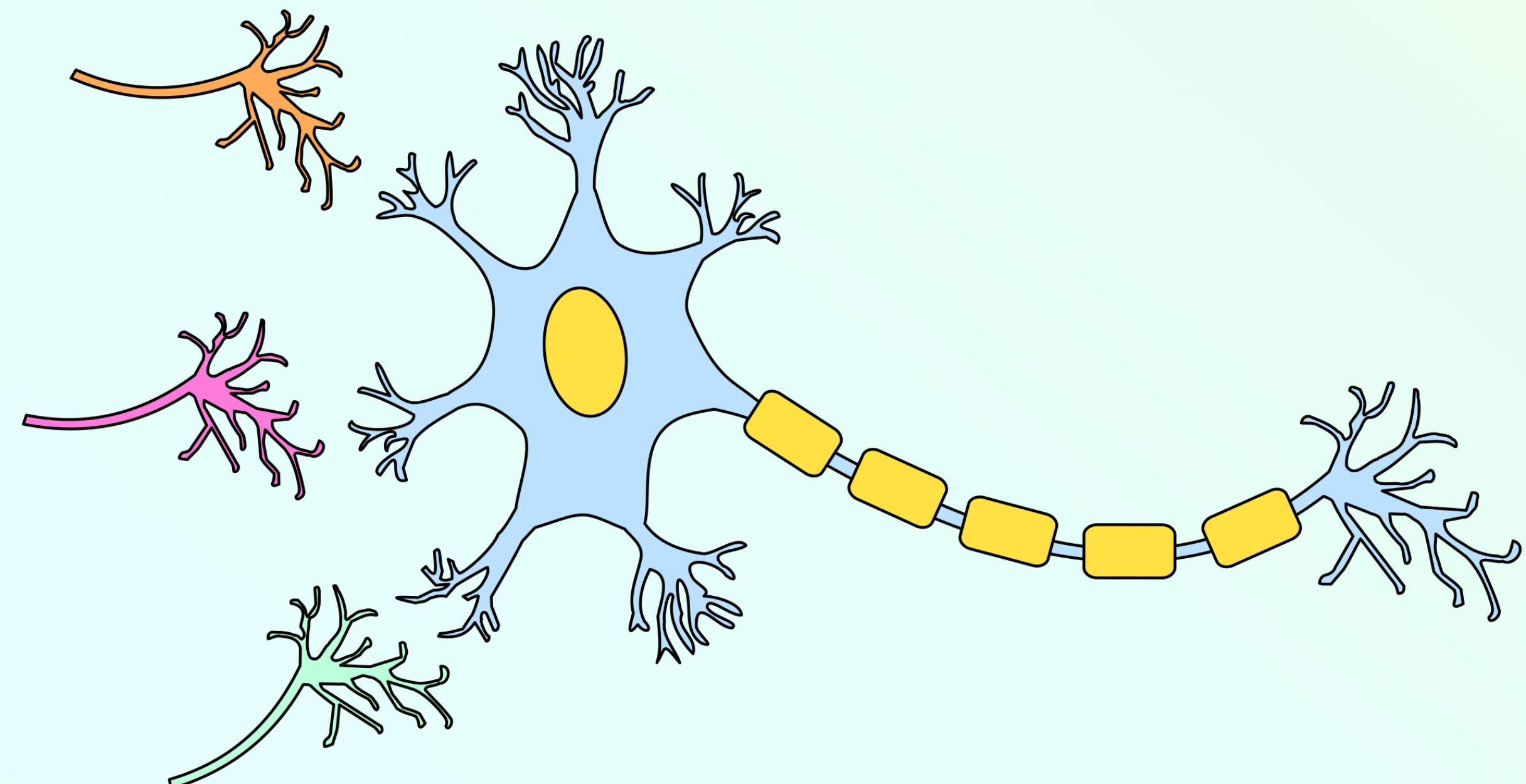


$$x_1 w_1 + x_2 w_2 + x_3 w_3 + b$$

$$\sum_{i=1}^3 x_i w_i + b$$

PERCEPTRÓN

Con esto en mente, armemos el modelo de neurona



$$x_1 w_1 + x_2 w_2 + x_3 w_3 + b$$

$$\sum_{i=1}^4 x_i w_i$$

$$y = f \left(\sum_{i=1}^4 x_i w_i \right)$$

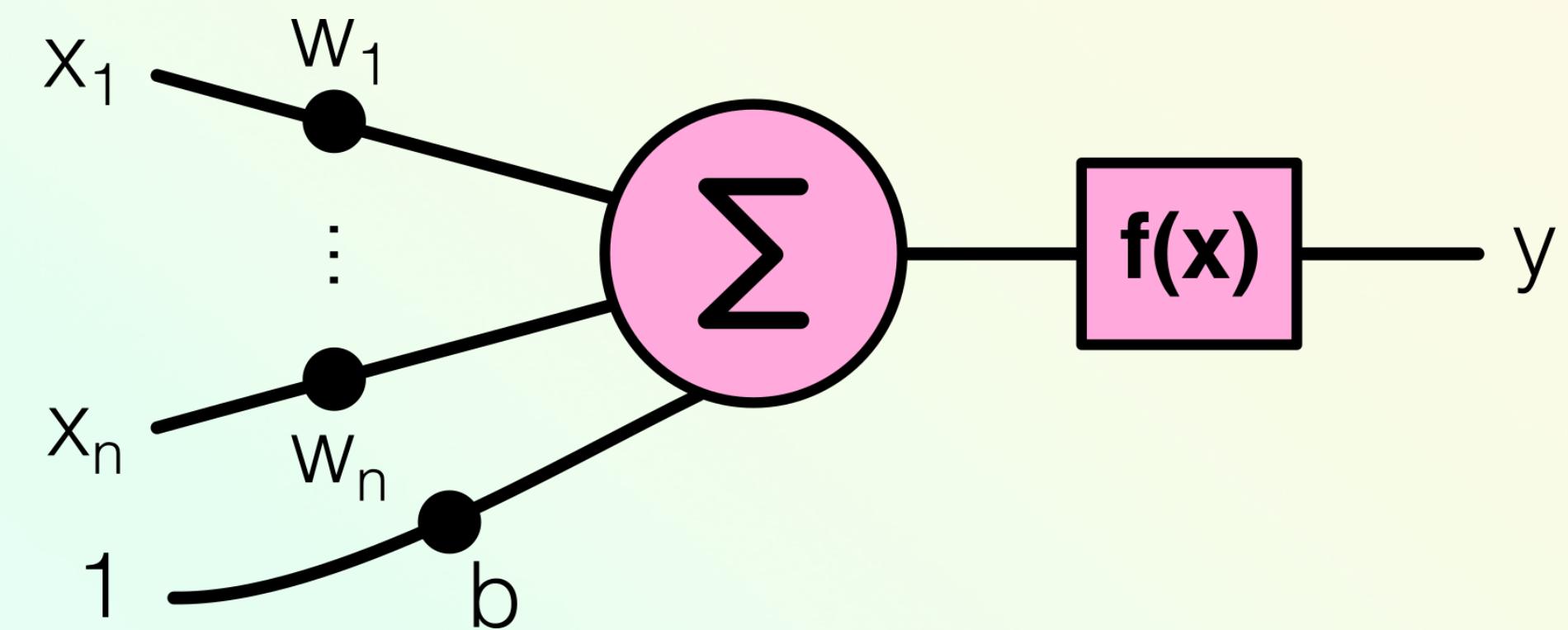
$$f(x) \rightarrow y$$

PERCEPTRÓN

Si tenemos una observación de **n** atributos, la neurona tendrá **n** entradas con **n+1** pesos sinápticos.

Por lo que la parte lineal es:

$$\sum_{i=1}^n x_i w_i + b$$



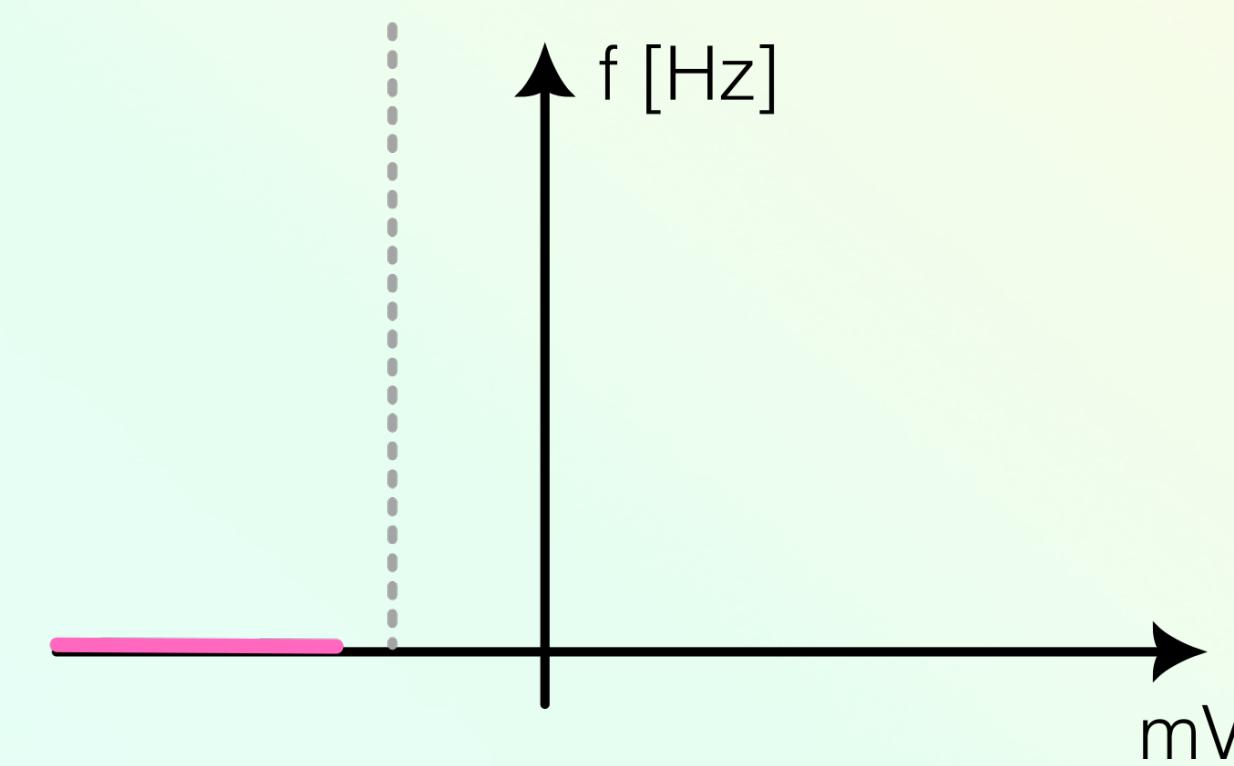
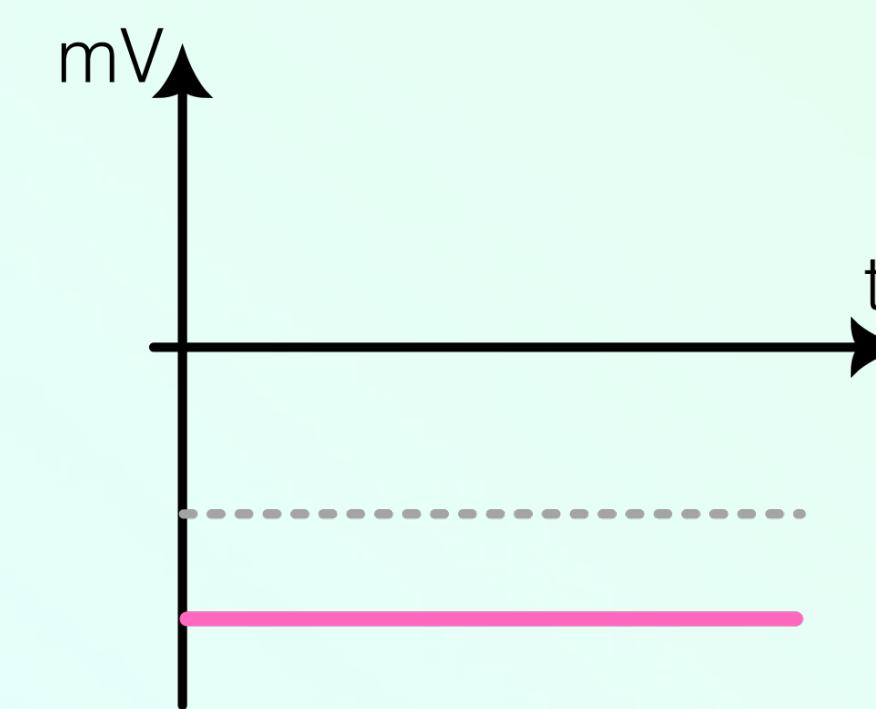
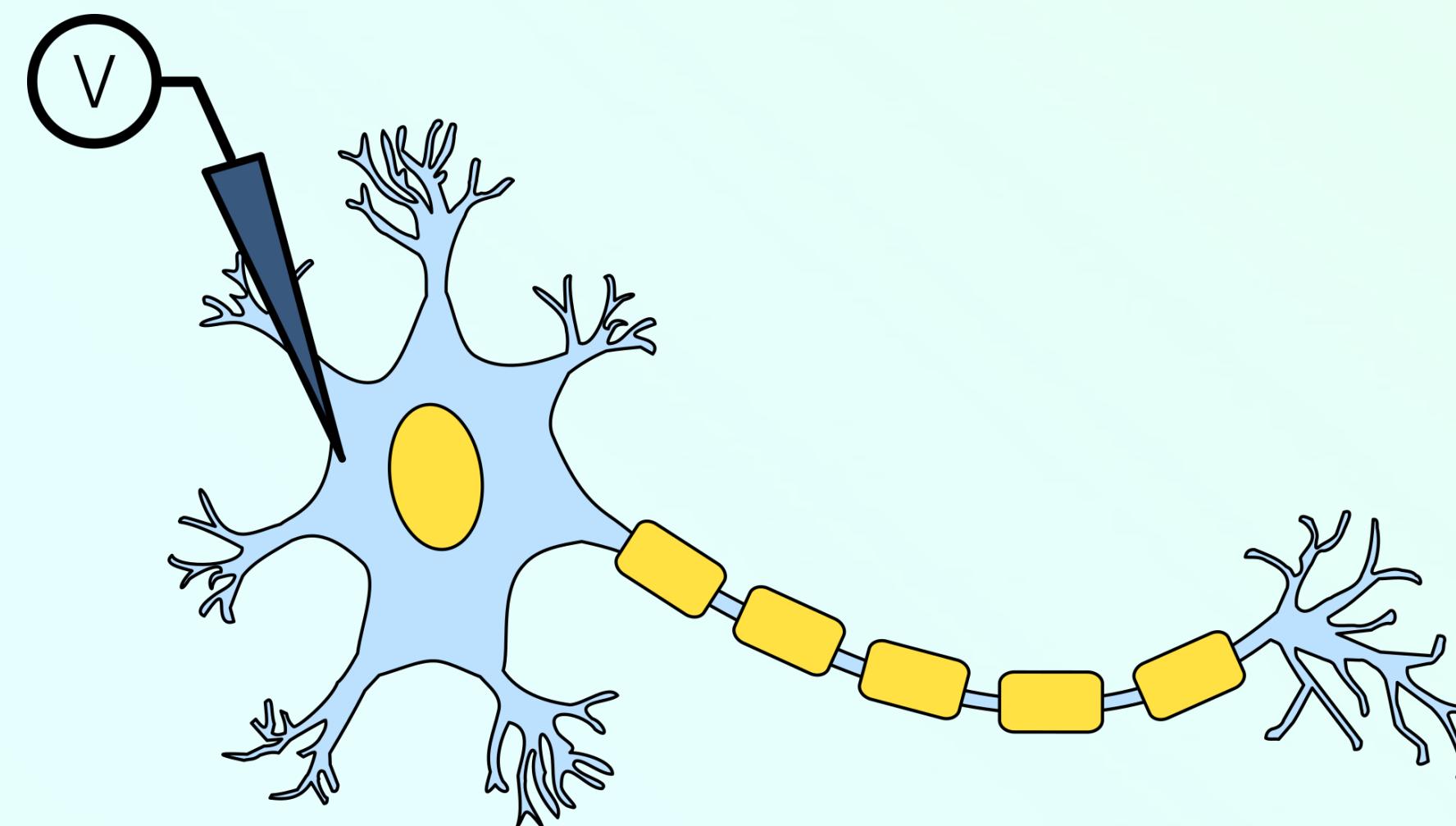
Donde w_i , b son los pesos sinápticos, que representan si la conexión es excitatoria ($w_i > 0$) o inhibitoria ($w_i < 0$), o que no haya conexión sináptica ($w_i = 0$)

PERCEPTRÓN

Veamos la función de activación. ¿Qué estamos modelando con ella?

En nuestro modelo, no nos interesa modelar el impulso nervioso, ni siquiera cuando ocurre, sino modelar la tasa de impulsos.

Hagamos un experimento: Pinchamos una neurona e inyectemos voltaje, y midamos la tasa de impulsos nervioso versus el voltaje. Al principio, antes que llegue al valor umbral de disparo, la neurona no dispara y por lo tanto la tasa es 0 Hz.

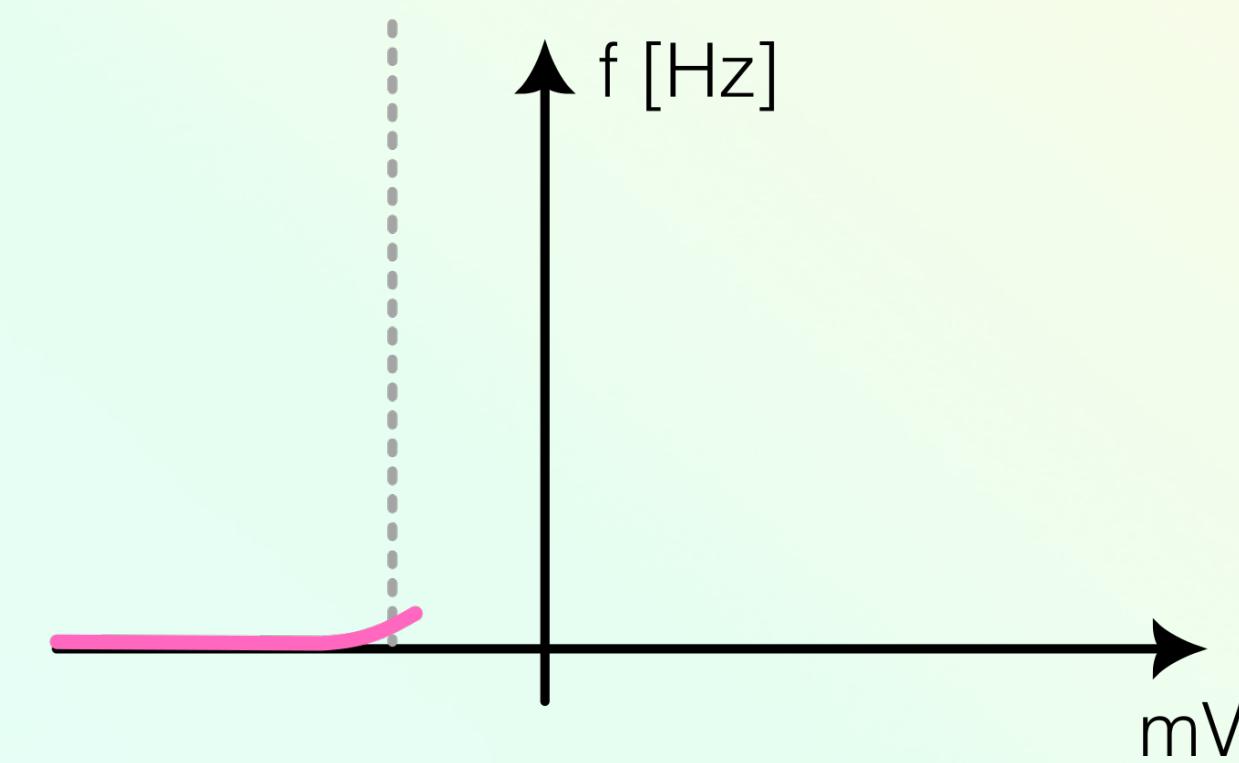
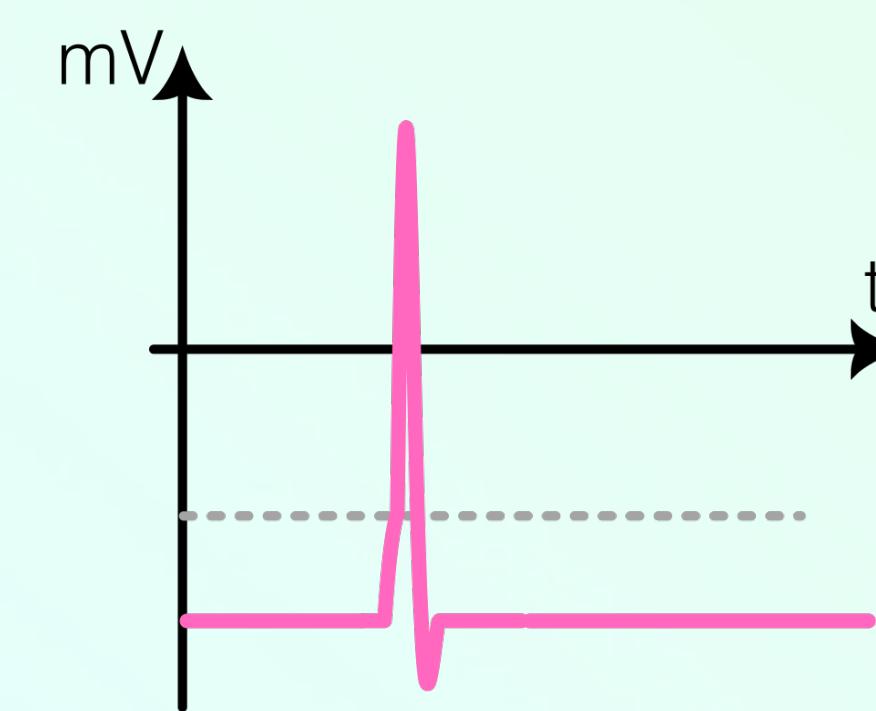
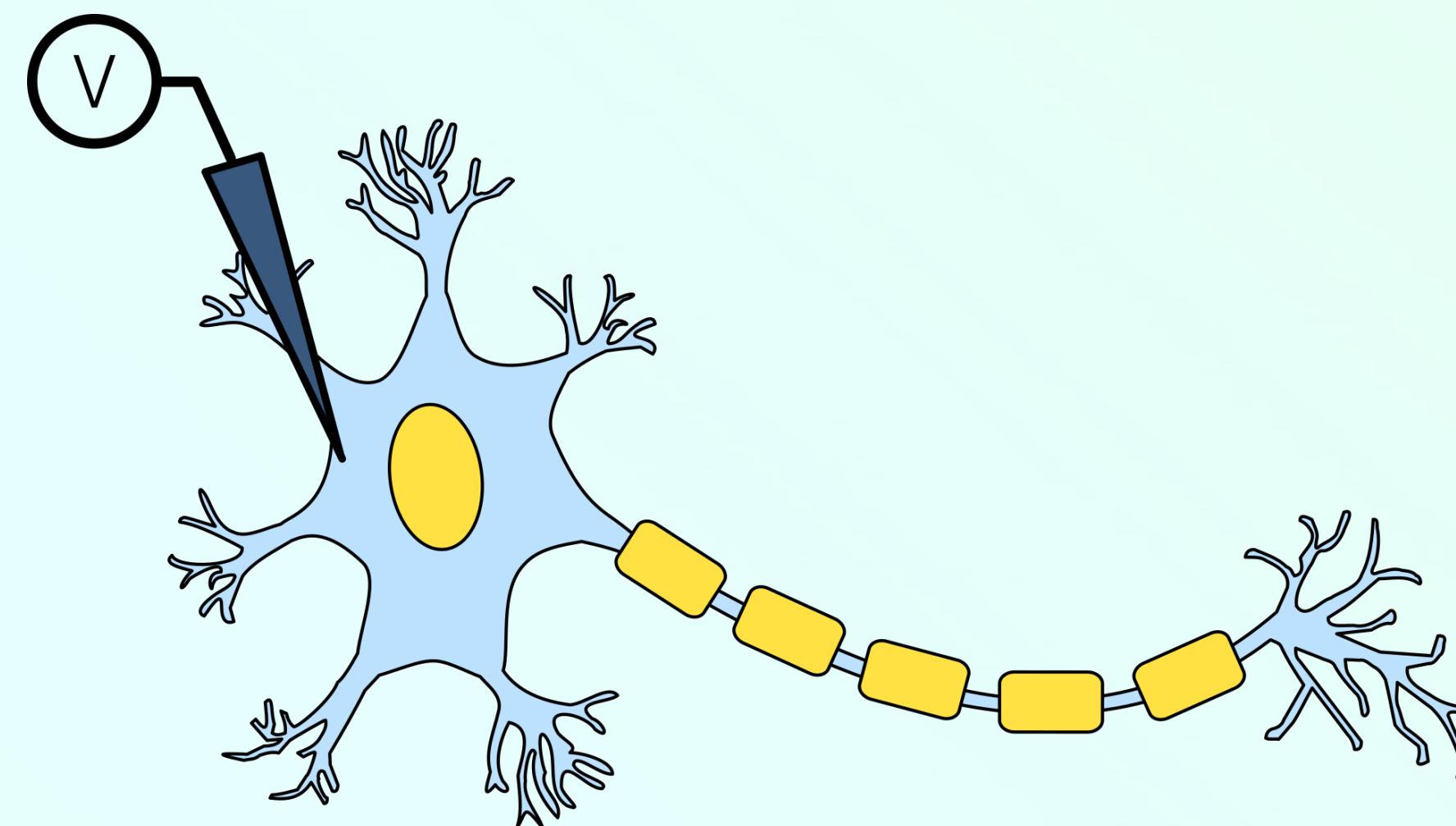


PERCEPTRÓN

Veamos la función de activación. ¿Qué estamos modelando con ella?

En nuestro modelo, no nos interesa modelar el impulso nervioso, ni siquiera cuando ocurre, sino modelar la tasa de impulsos.

Hagamos un experimento: Pinchamos una neurona e inyectemos voltaje, y midamos la tasa de impulsos nerviosos versus el voltaje. Cuando aumentamos y pasamos el umbral, la neurona empieza a disparar.

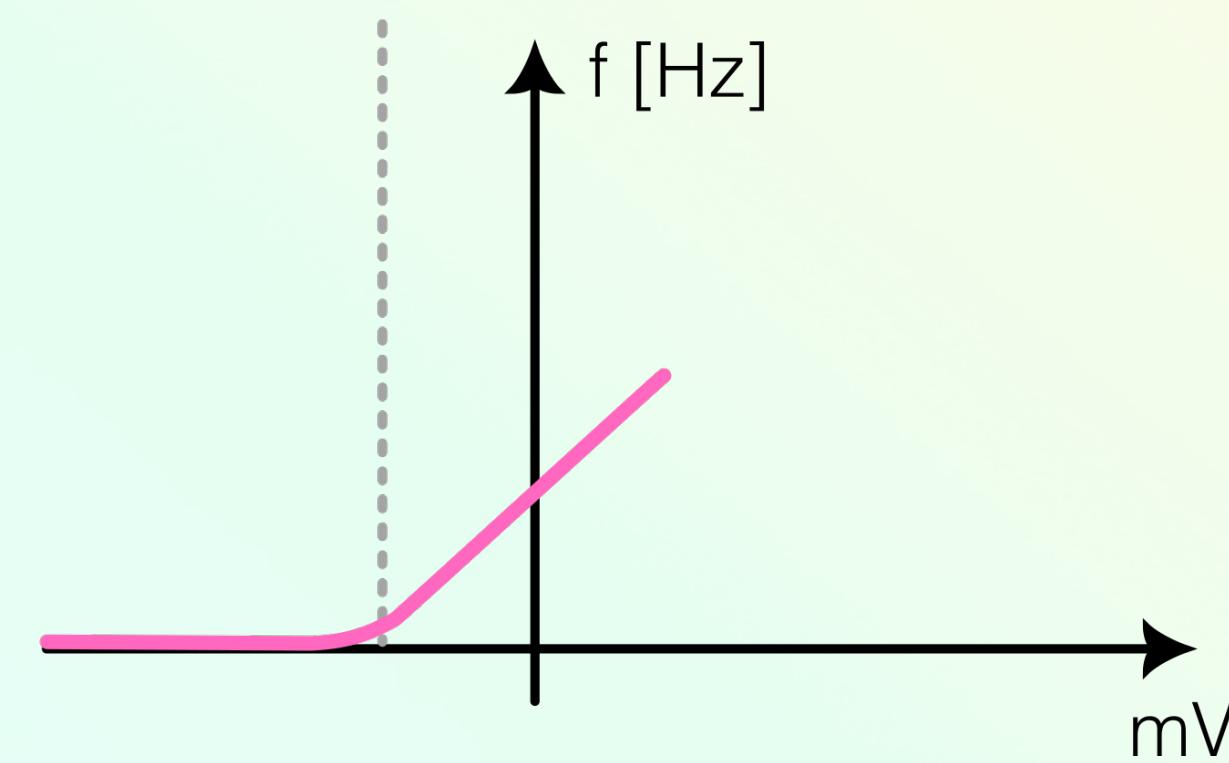
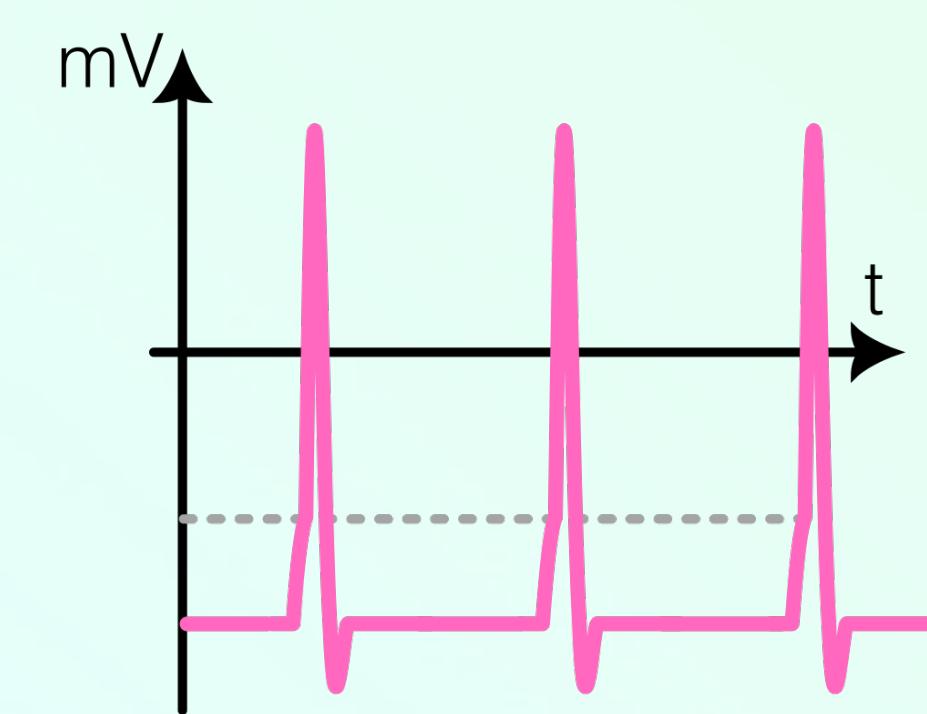
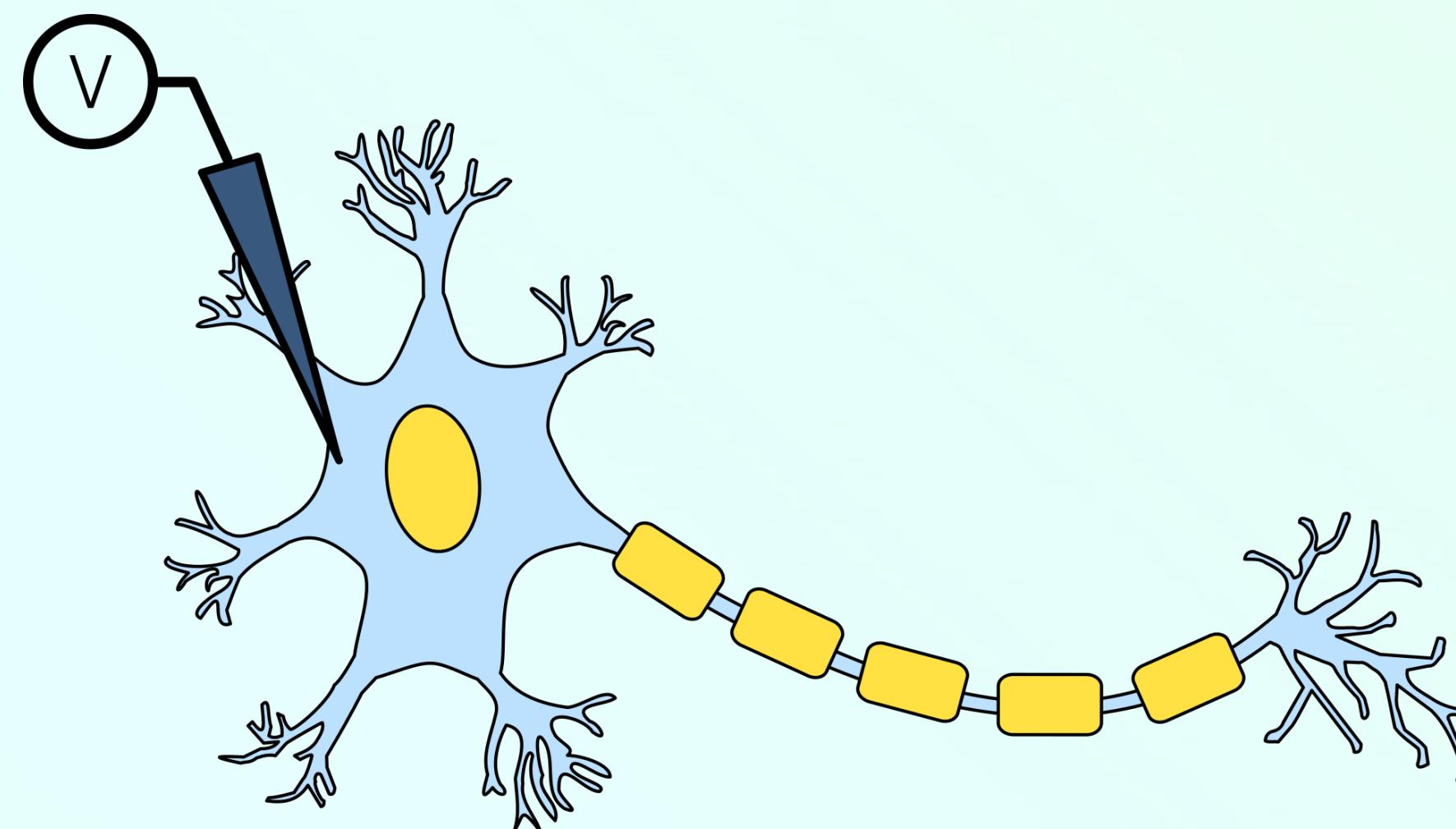


PERCEPTRÓN

Veamos la función de activación. ¿Qué estamos modelando con ella?

En nuestro modelo, no nos interesa modelar el impulso nervioso, ni siquiera cuando ocurre, sino modelar la tasa de impulsos.

Hagamos un experimento: Pinchamos una neurona e inyectemos voltaje, y midamos la tasa de impulsos nervioso versus el voltaje. Durante la etapa lineal, si duplicamos el voltaje, la neurona dispara al doble de la frecuencia.

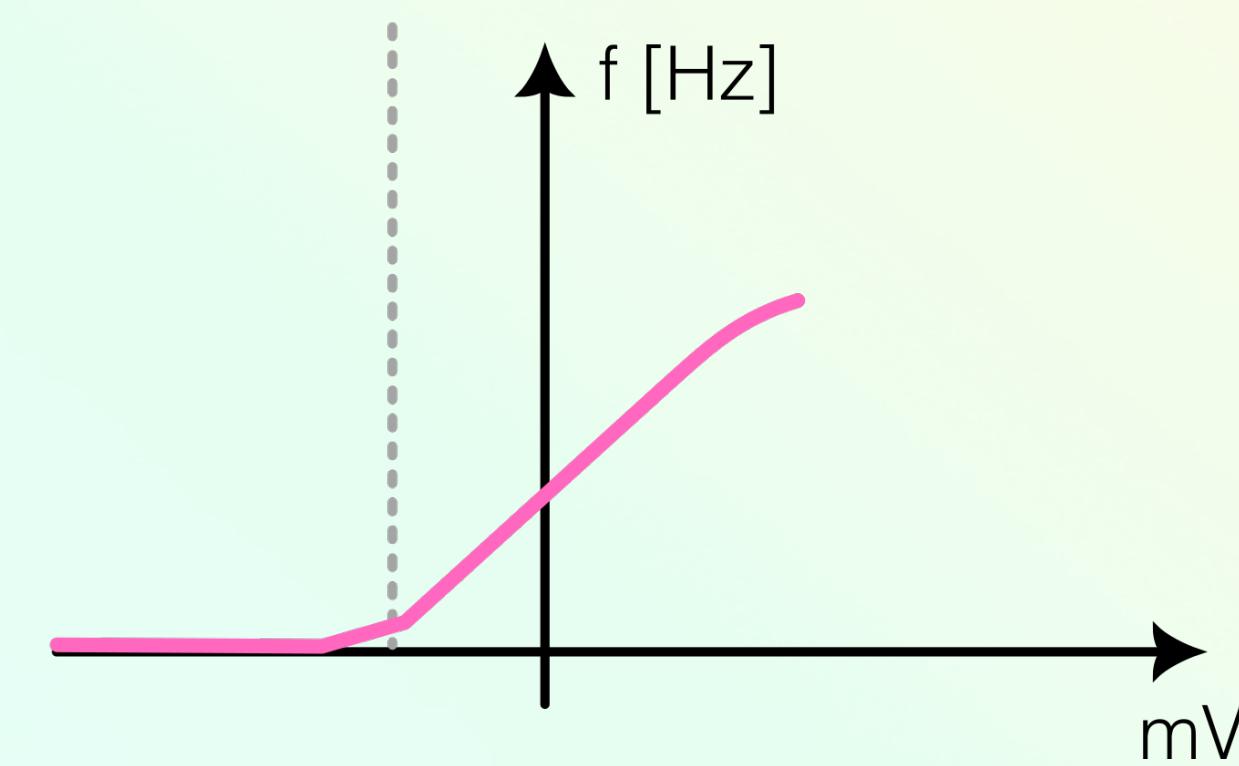
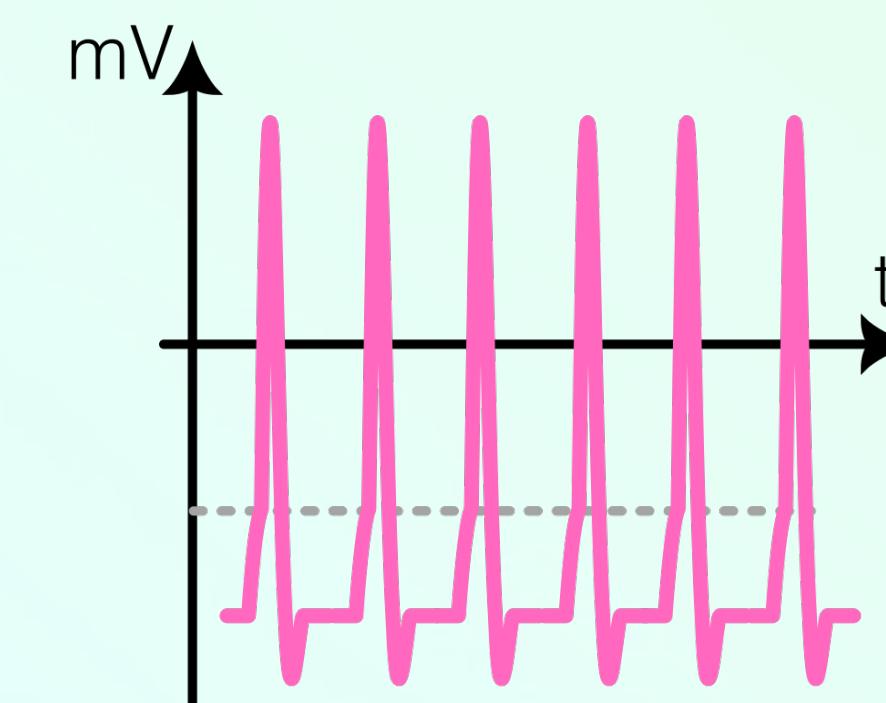
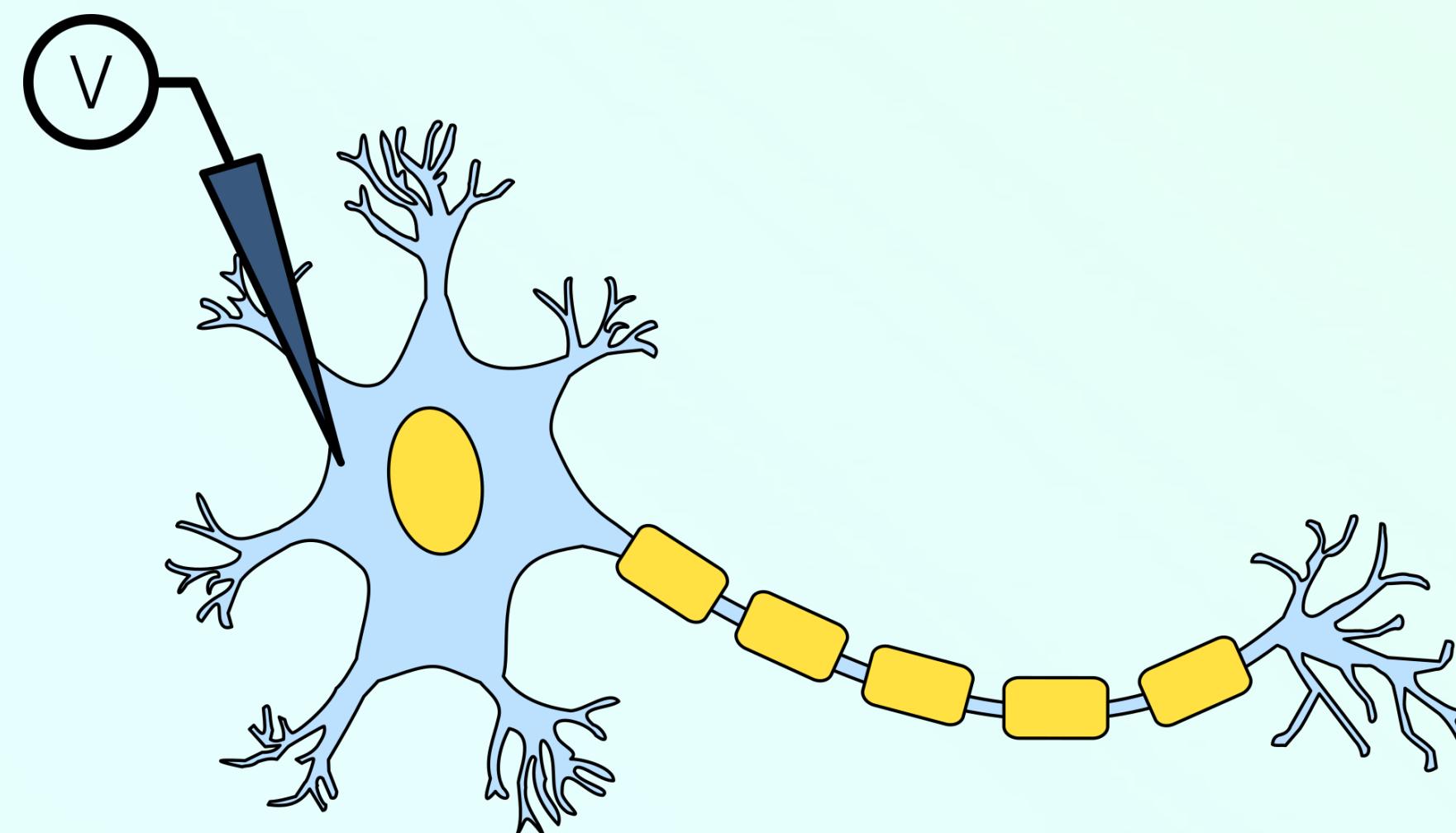


PERCEPTRÓN

Veamos la función de activación. ¿Qué estamos modelando con ella?

En nuestro modelo, no nos interesa modelar el impulso nervioso, ni siquiera cuando ocurre, sino modelar la tasa de impulsos.

Hagamos un experimento: Pinchamos una neurona e inyectemos voltaje, y midamos la tasa de impulsos nervioso versus el voltaje. Pero llega una etapa que el periodo refractario deja que aumente la tasa de disparo y se empiece a saturar

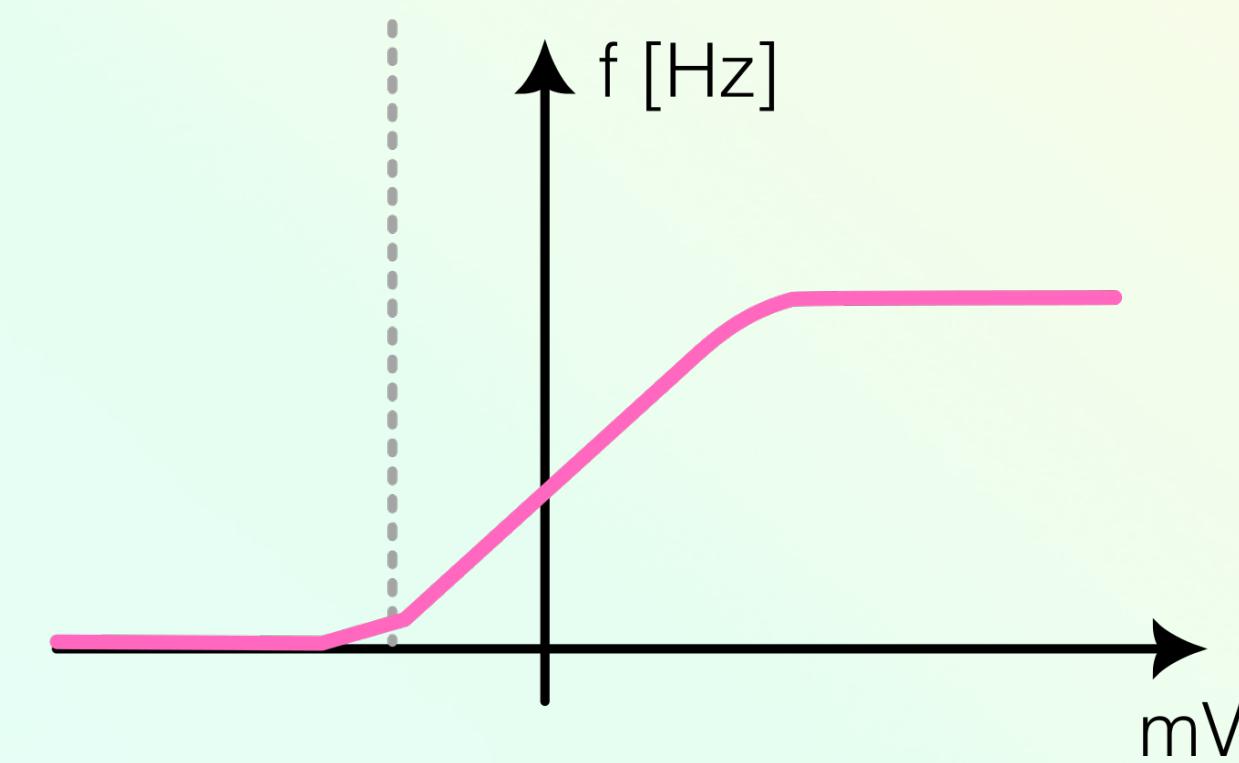
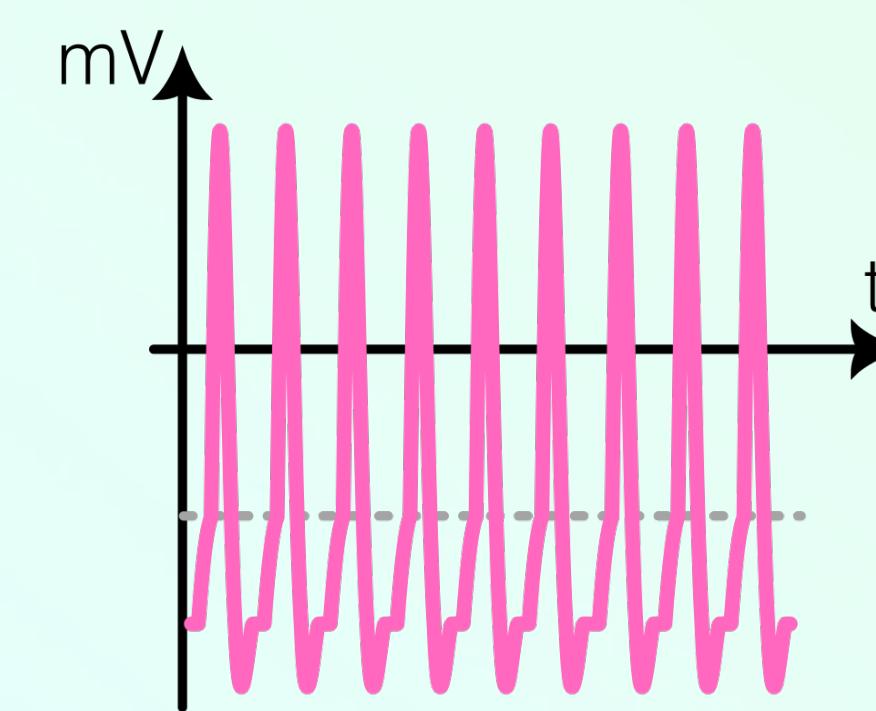
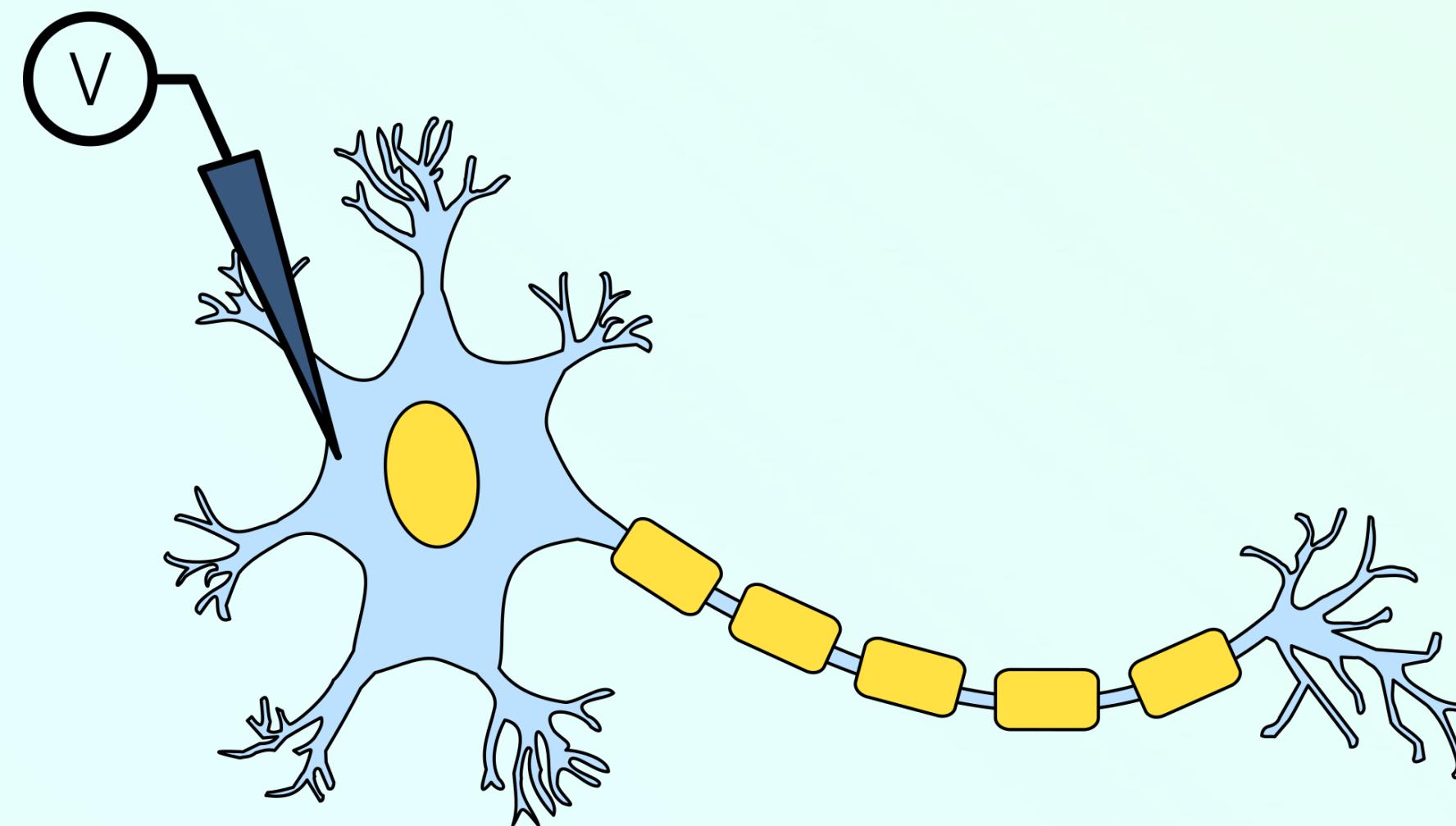


PERCEPTRÓN

Veamos la función de activación. ¿Qué estamos modelando con ella?

En nuestro modelo, no nos interesa modelar el impulso nervioso, ni siquiera cuando ocurre, sino modelar la tasa de impulsos.

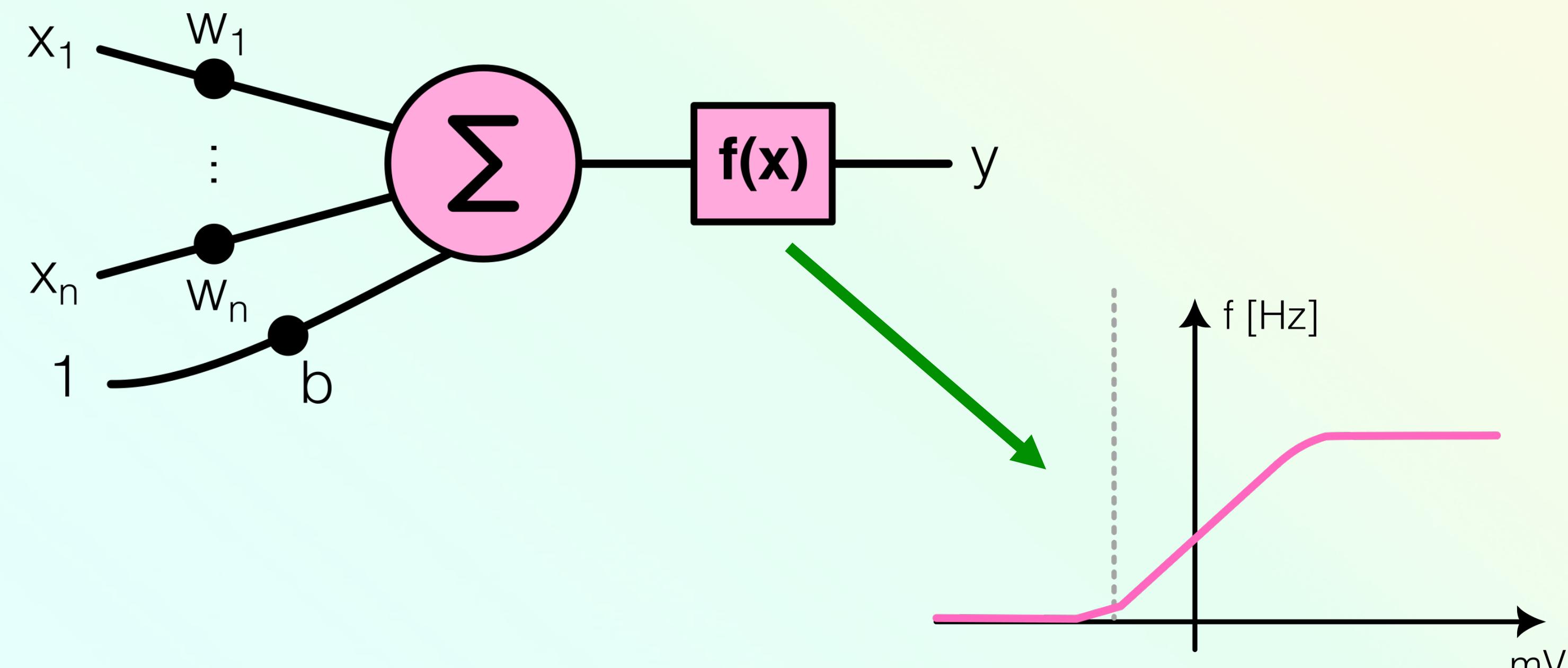
Hagamos un experimento: Pinchamos una neurona e inyectemos voltaje, y midamos la tasa de impulsos nervioso versus el voltaje. Cuando llegamos a un punto, no importa cuánto voltaje introduzcamos, a la neurona dispara a la máxima frecuencia.



PERCEPTRÓN

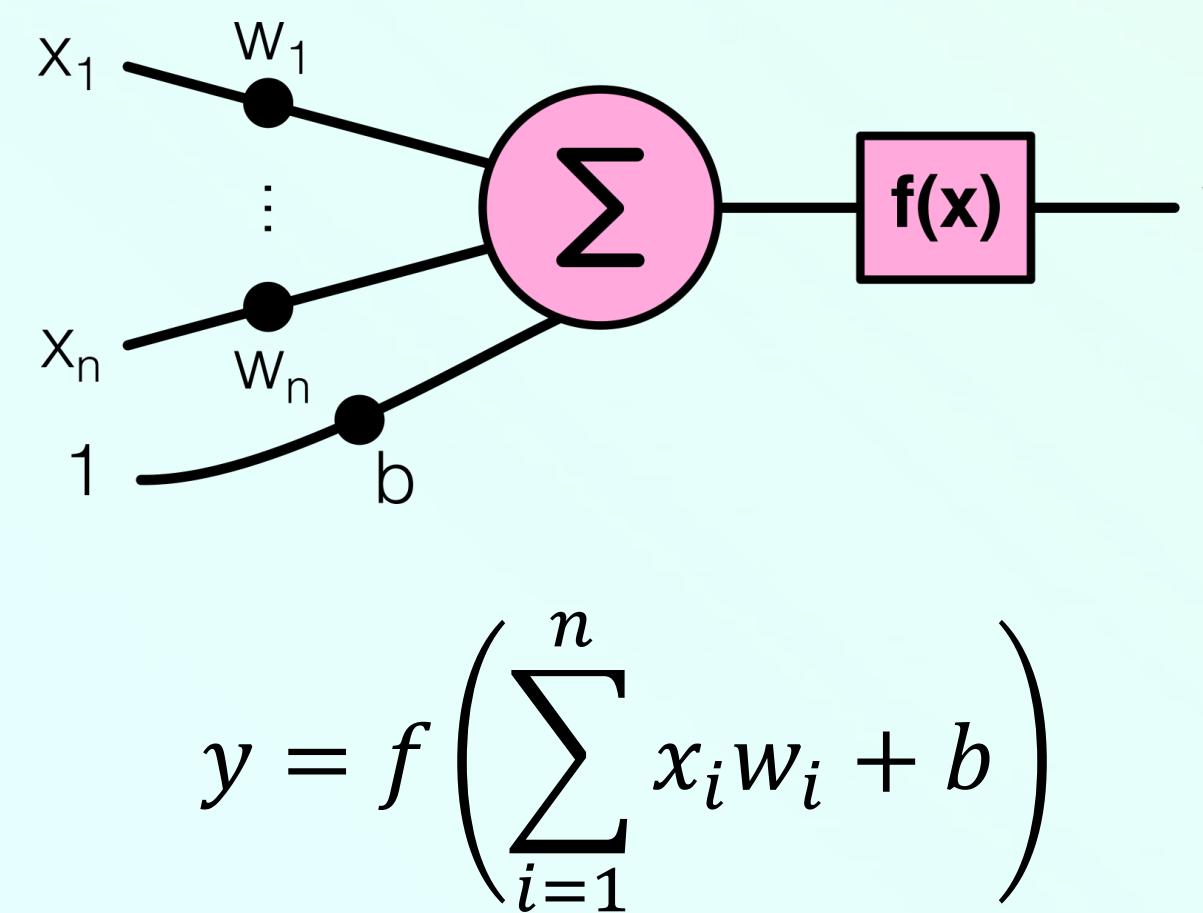
Por lo que la función de activación es una función no lineal que modela la variación de la tasa de disparo de la neurona.

Y con esto último, tenemos la expresión mínima de una neurona, el cual es una **unidad de cálculo no lineal**.



PERCEPTRÓN

La función de activación que se usa en redes neuronales es una decisión de diseño:



Función escalón

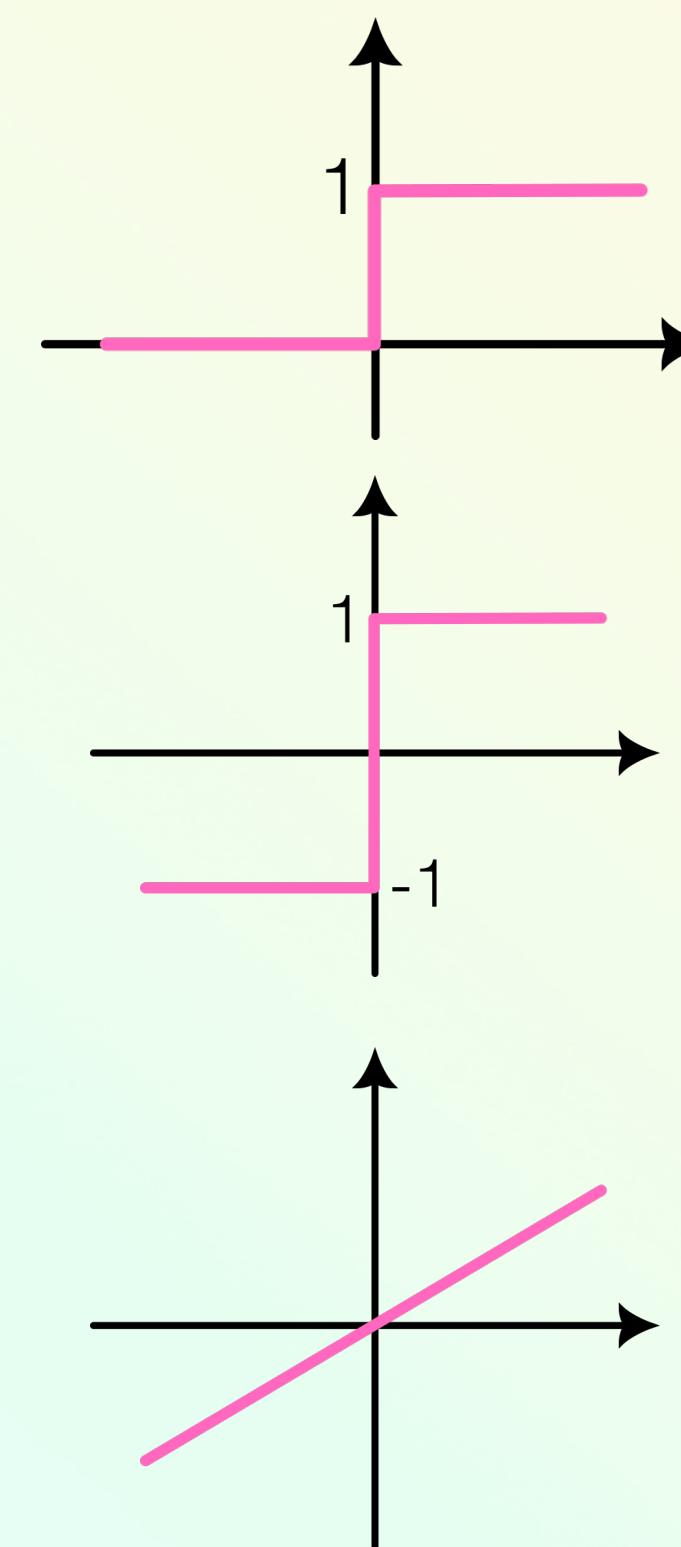
Función signo

Función lineal

$$f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

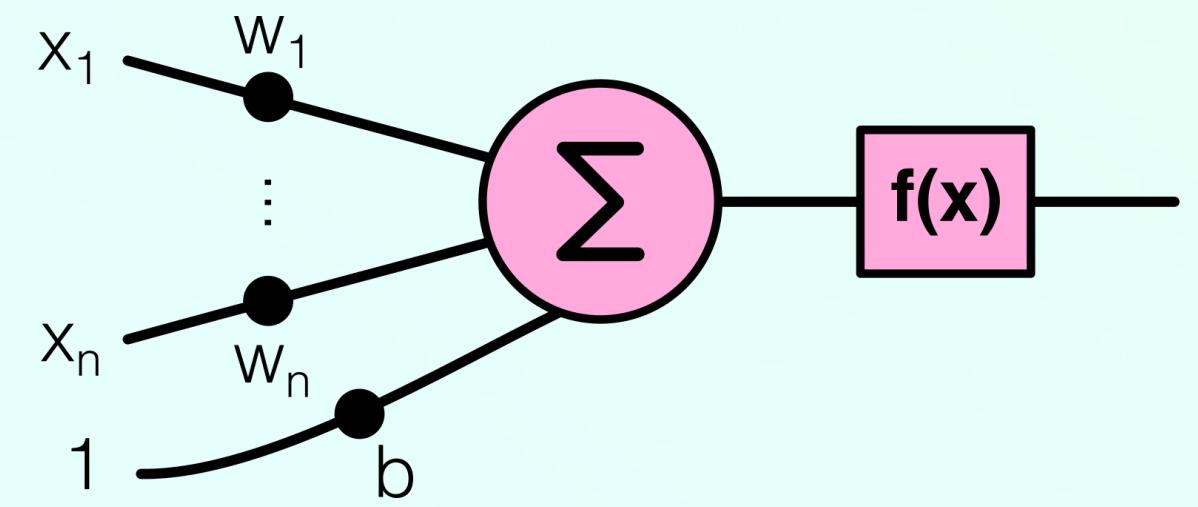
$$f(x) = \begin{cases} -1 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

$$f(x) = x$$



PERCEPTRÓN

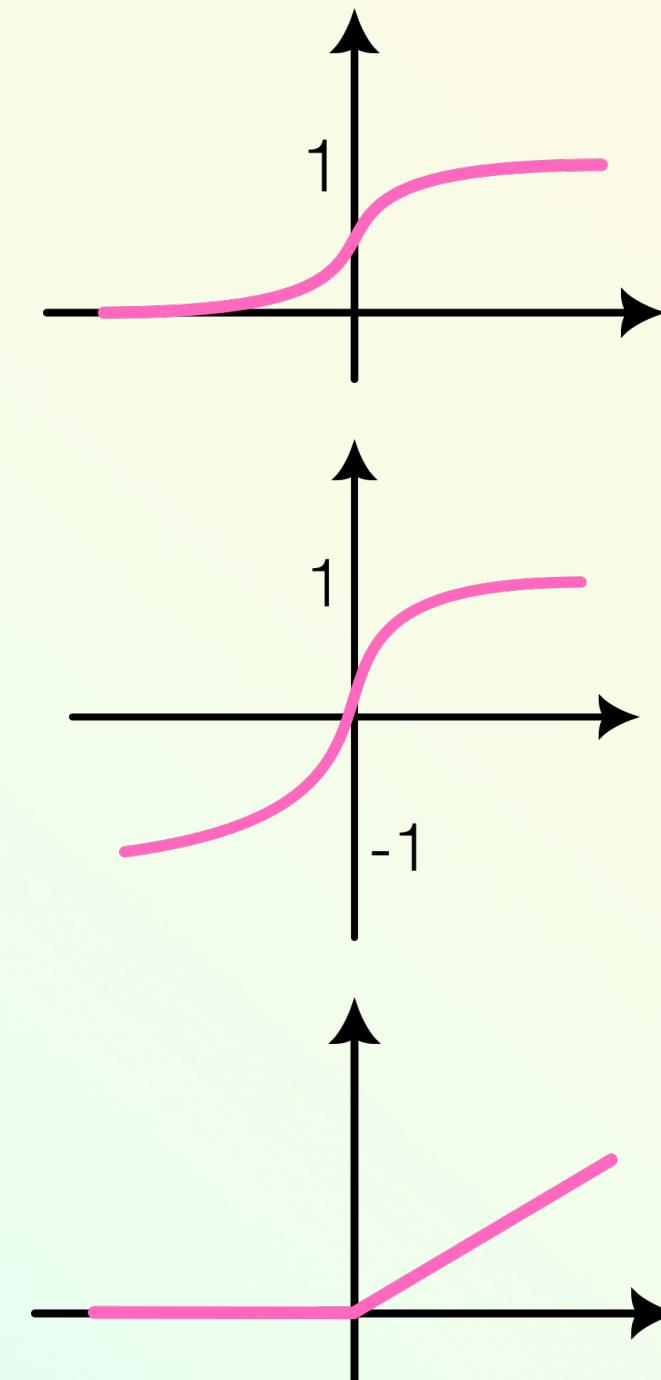
La función de activación que se usa en redes neuronales es una decisión de diseño:



$$y = f\left(\sum_{i=1}^n x_i w_i + b\right)$$

Funciones sigmoideas

$$f(x) = \frac{e^x}{1 + e^x}$$



Función ReLu

$$f(x) = \tanh(x)$$

$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

PERCEPTRÓN

Con un perceptrón o neurona o red de una sola capa, se toma un vector de variables:

$$X = (x_1, x_2, \dots, x_n)$$

Donde, una neurona entrenada, puede predecir un **label** o variable a predecir,

$$y = f\left(\sum_{i=1}^n x_i w_i + b\right)$$

Si usamos la **función de activación lineal**... podemos ver una regresión lineal

Si usamos la **función sigmoidea**... podemos ver una regresión logística. Al usar funciones de activación con salida continua, tenemos una métrica no solo de que clase es, sino un valor probabilístico de que tan seguro es la predicción.

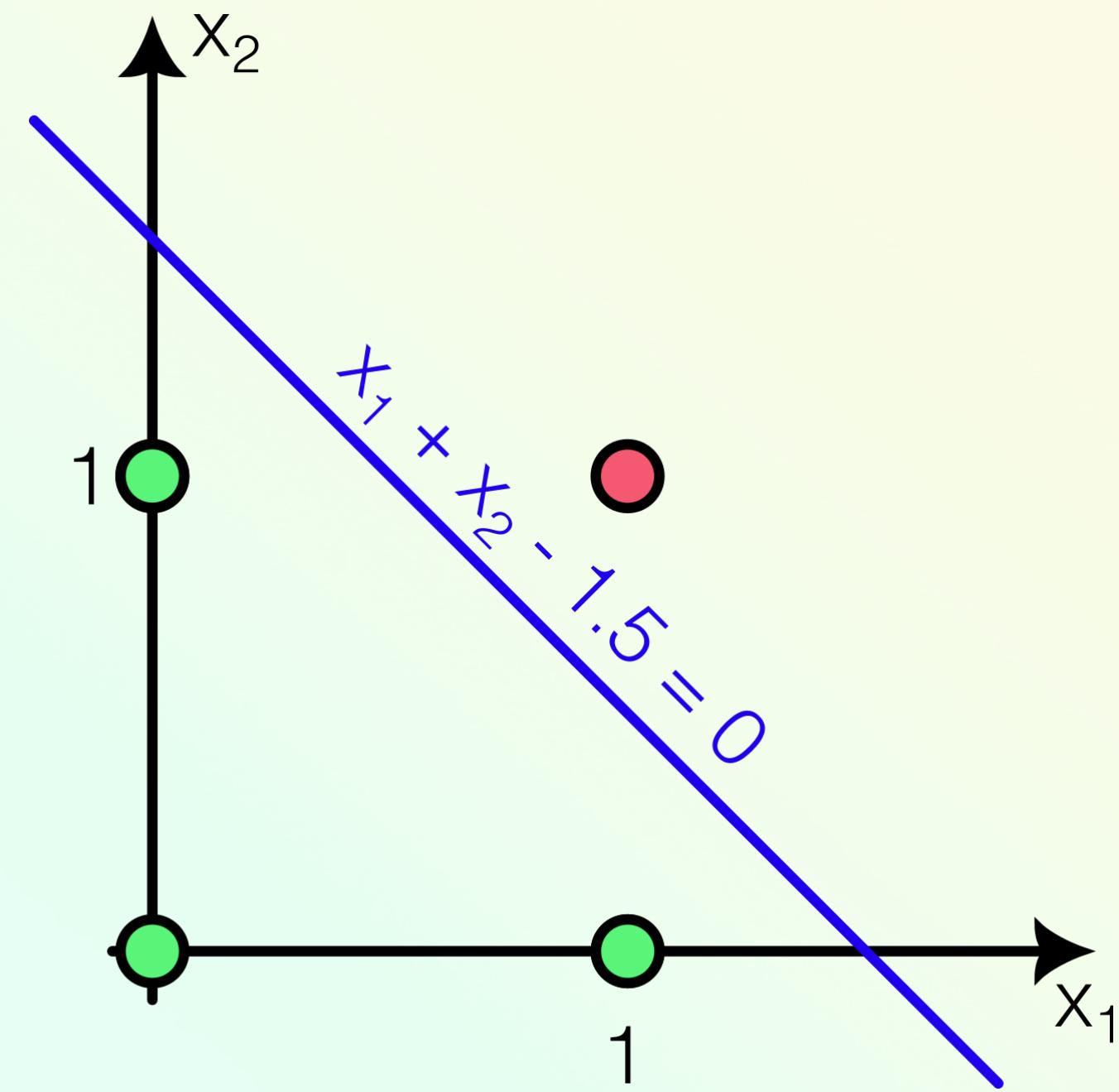
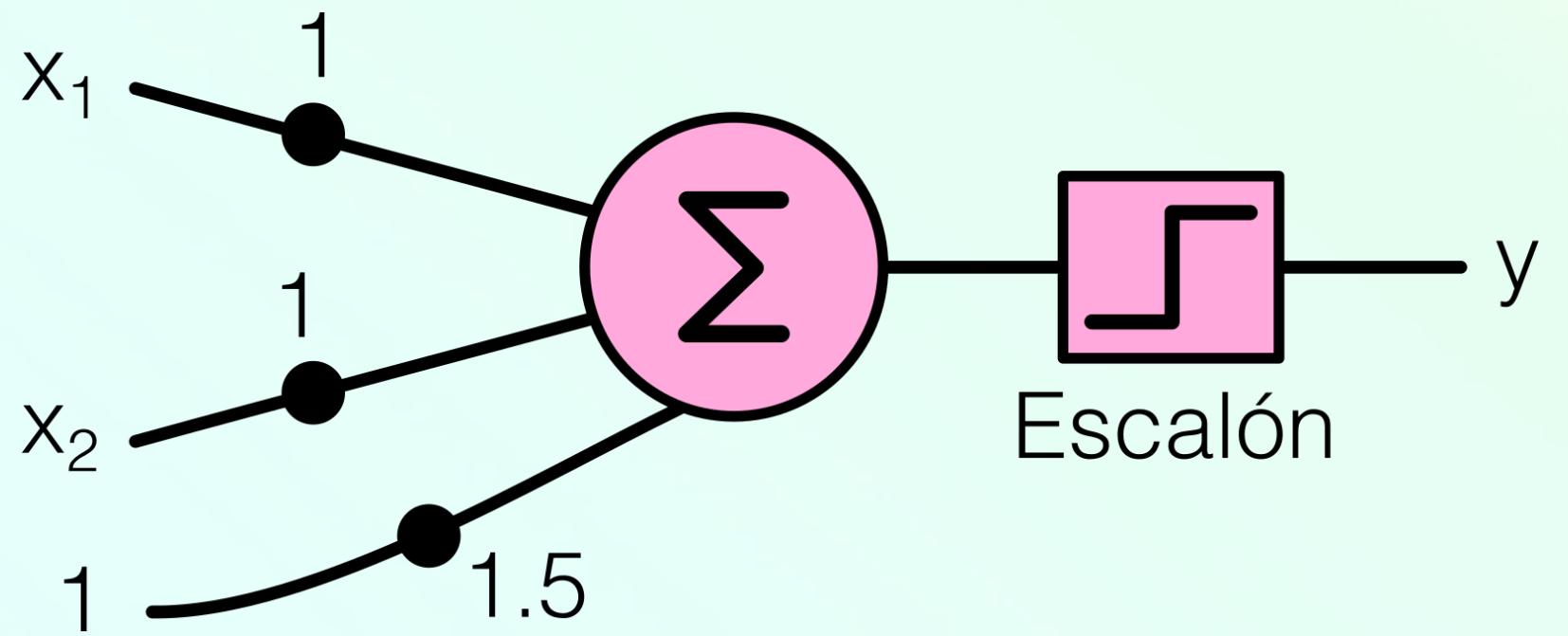
Como podemos ver una neurona puede hacer clasificaciones o regresiones. Una neurona en un espacio n-dimensional pone un hiperplano para separar clases o realizar una regresión.

PERCEPTRÓN

Veamos el ejemplo de funciones lógicas:

AND (2 entradas)

X ₁	X ₂	Y
0	0	0
0	1	0
1	0	0
1	1	1

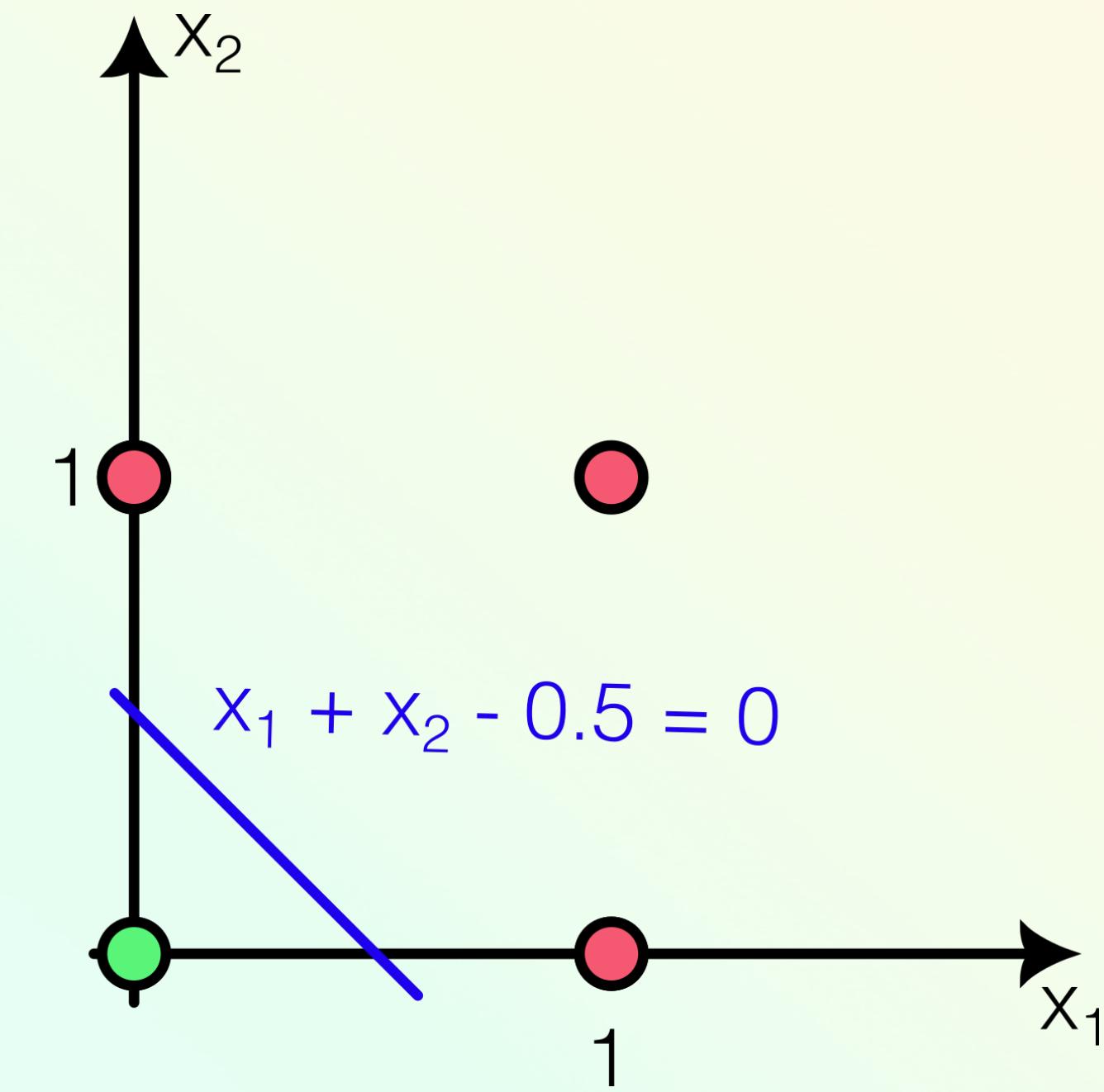
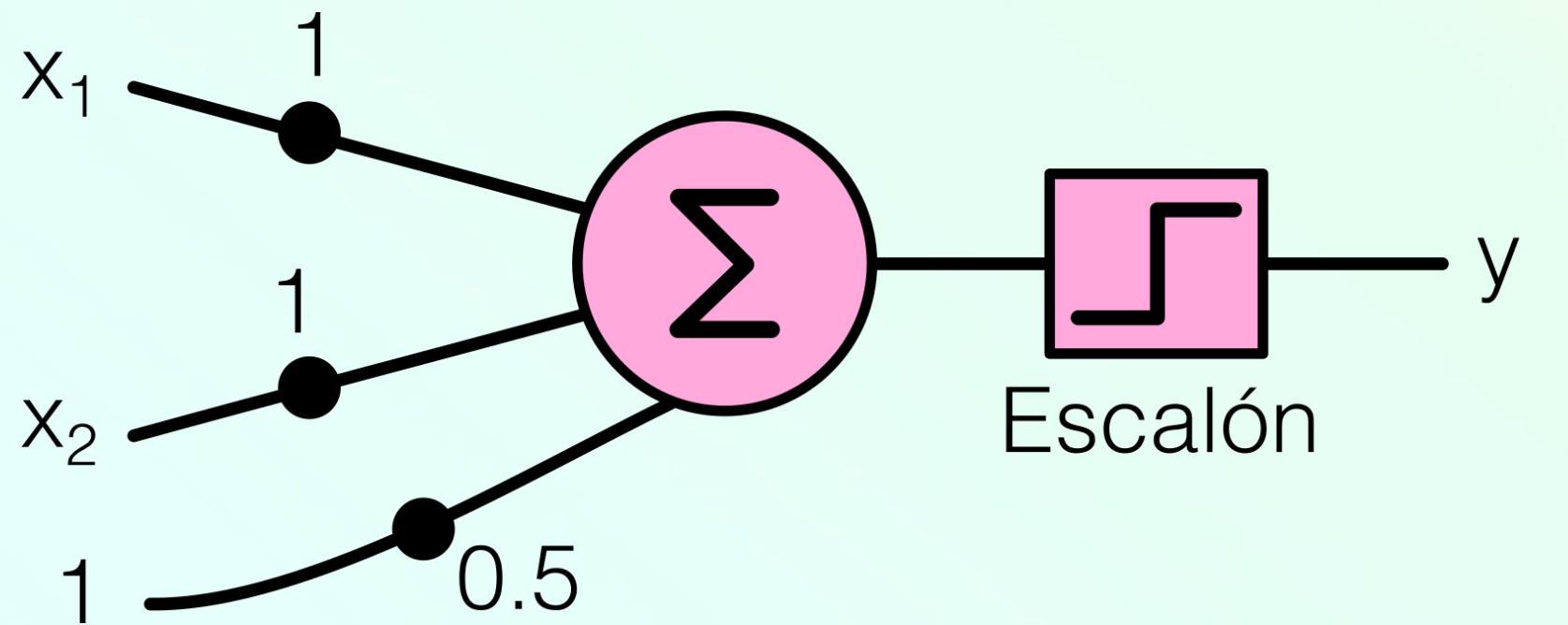


PERCEPTRÓN

Veamos el ejemplo de funciones lógicas:

OR (2 entradas)

X₁	X₂	Y
0	0	0
0	1	1
1	0	1
1	1	1

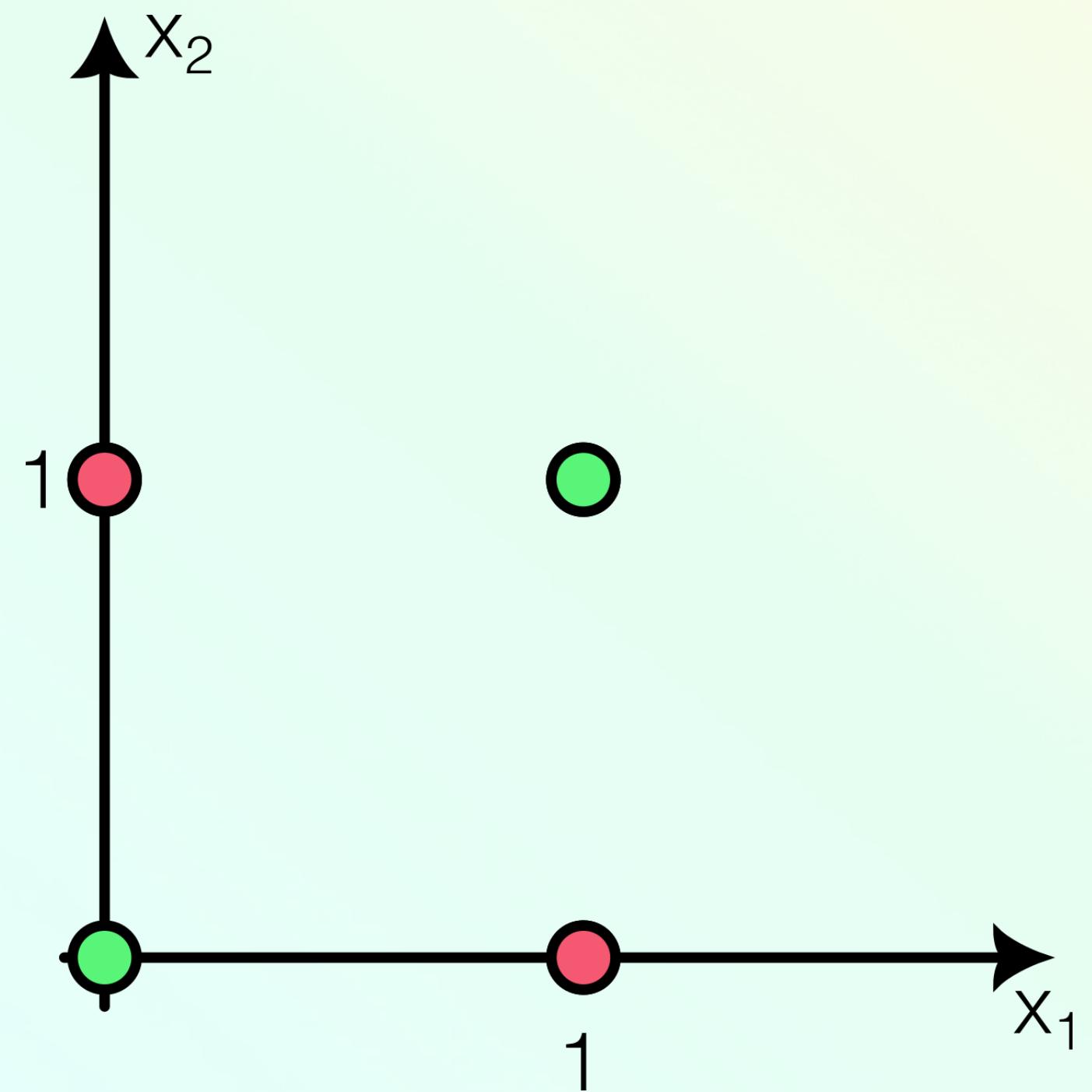


PERCEPTRÓN

Veamos el ejemplo de funciones lógicas:

XOR (2 entradas)

X₁	X₂	Y
0	0	0
0	1	1
1	0	1
1	1	0



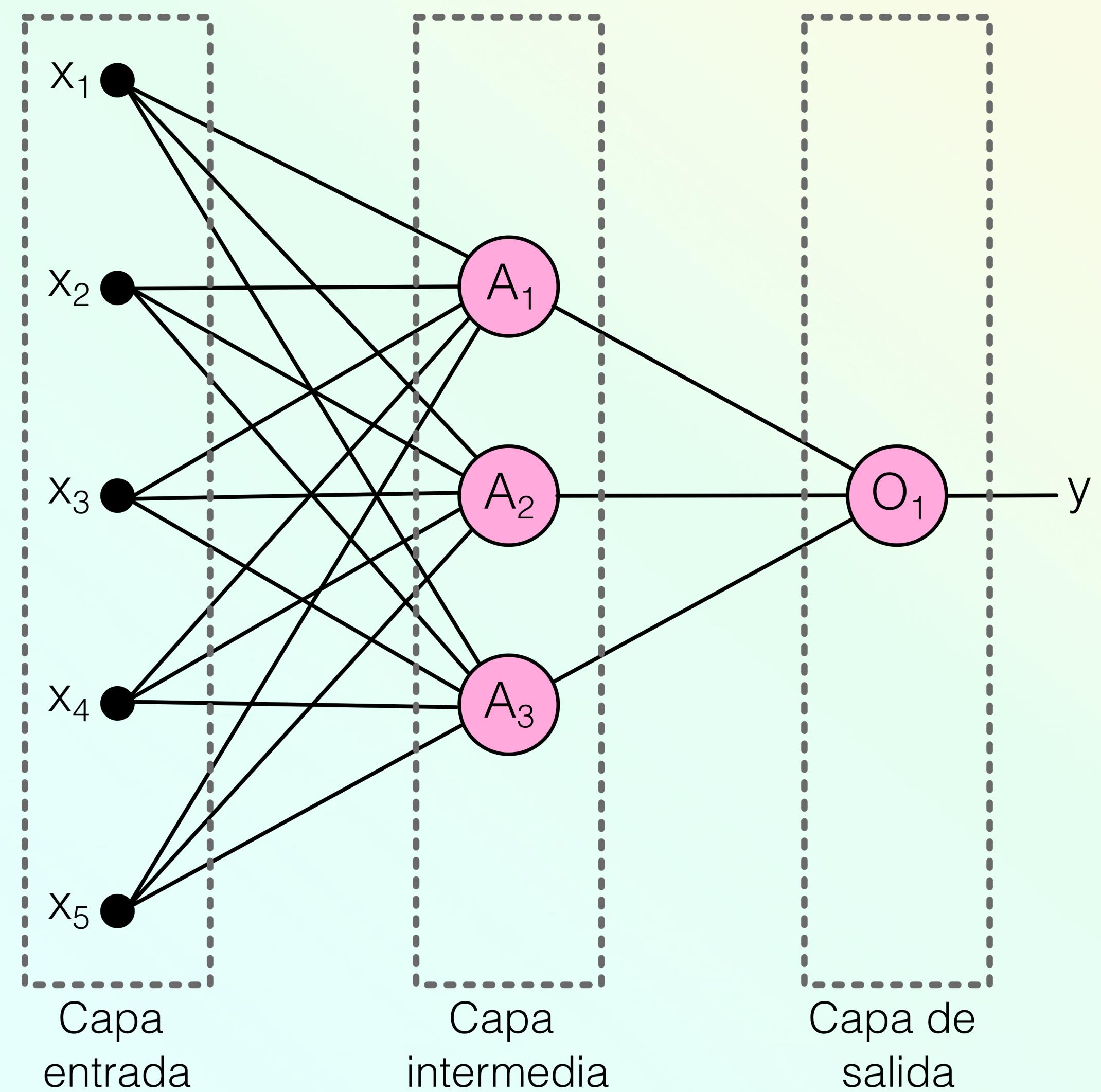
REDES FEED-FORWARD

REDES FEED-FORWARD

La forma de tener fronteras de decisión más complejas que la lineales o regresiones más complejas que la lineal, una forma de resolverlo es armando una red formada por neuronas en capas, en donde cada capa se conecta con la siguiente pero nunca hay retroalimentación.

Además, este tipo de redes, también llamadas redes densas, una neurona de una capa se conecta con todas las neuronas de la siguiente capa.

REDES FEED-FORWARD



REDES FEED-FORWARD

Este tipo de redes, la capa intermedia transforma el espacio de entrada en diferentes espacios generalmente no lineales.

Si todas las neuronas tienen función de activación lineal, la red colapsa a una sola neurona, por lo que una red feed-forward se justifica en casos no lineales.

REDES FEED-FORWARD

Cada neurona de la capa intermedia tiene la misma función de activación y su salida es:

$$A_k = g \left(\sum_{i=1}^n x_i w_{ki} + b_k \right)$$

Esta transformación producida por la capa intermedia, luego ingresan como entrada a la capa final:

$$y = f \left(\sum_{k=1}^K A_k \omega_k + \beta \right)$$

BREVE INTRODUCCIÓN DE ENTRENAMIENTO DE REDES

ENTRENAMIENTO

El entrenamiento de redes es algo complejo, y lo verán en más detalles en otra asignatura. Entrenar una red neuronal, es ajustar los pesos sinápticos para que los resultados coincidan con los valores a predecir del set de entrenamiento. Es decir, las redes neuronales feed-forward es de **aprendizaje supervisado**.

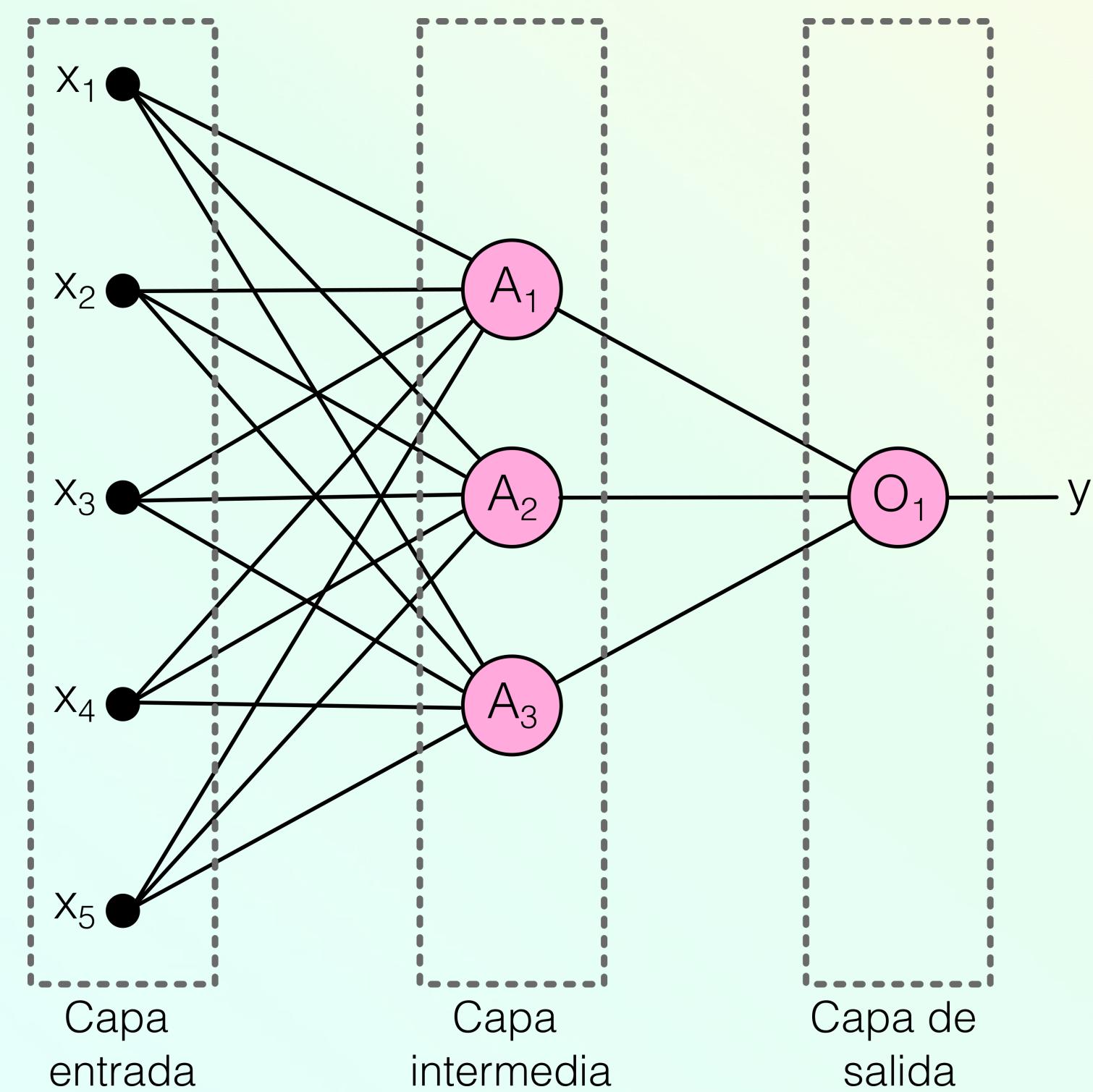
Para entrenar nuestro modelo, como vimos otros, es minimizar una función de perdida que sea acotada, de tal forma que podemos llegar, mediante alguna técnica de optimización al valor mínimo.

Para este caso, veamos en un caso de regresión, el error cuadrático medio:

$$\underset{w_{kj}, b_k, \omega_k, \beta}{\text{minimizar}} \frac{1}{2} \sum_{i=1}^n (y_i - f(X_i))^2$$

ENTRENAMIENTO

Para ahora vamos a basarnos en esta red:



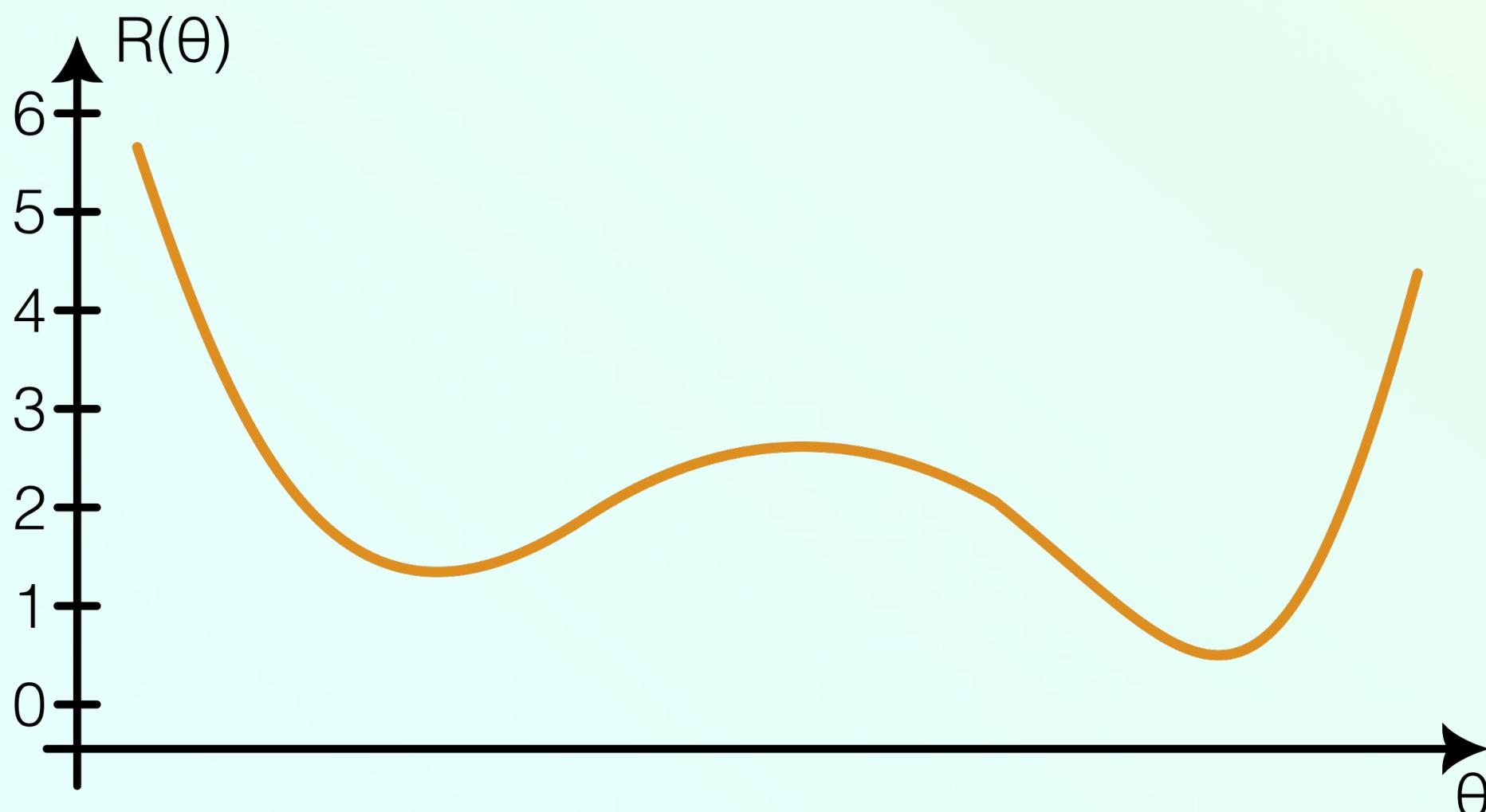
La función de activación g de la capa oculta es genérica, pero la de salida es lineal.

ENTRENAMIENTO

$$\underset{w_{kj}, b_k, \omega_k, \beta}{\text{minimizar}} \frac{1}{2} \sum_{i=1}^n (y_i - f(X_i))^2$$

$$\text{Donde: } f(X_i) = \beta + \sum_{k=1}^K \omega_k g\left(\sum_{i=1}^n x_i w_{ki} + b_k\right)$$

Esta función de perdida, que tiene tantas dimensiones como pesos sinápticos tiene, no es convexa y, por lo tanto, existen múltiples soluciones. Veamos a un ejemplo a que nos referimos a esto de un caso de optimización con un solo parámetro:



ENTRENAMIENTO

Para lograr evitar caer en mínimos locales y también de sobreajustes lo entrenamos lentamente usando un algoritmo de gradiente descendiente y el proceso se corta cuando se detecta sobreajuste usando set de validación.

Para ver cómo funciona gradiente descendiente, supongamos que ponemos a todos los pesos sinápticos como un gran vector θ , por lo que podemos reescribir a nuestro objetivo de minimización como:

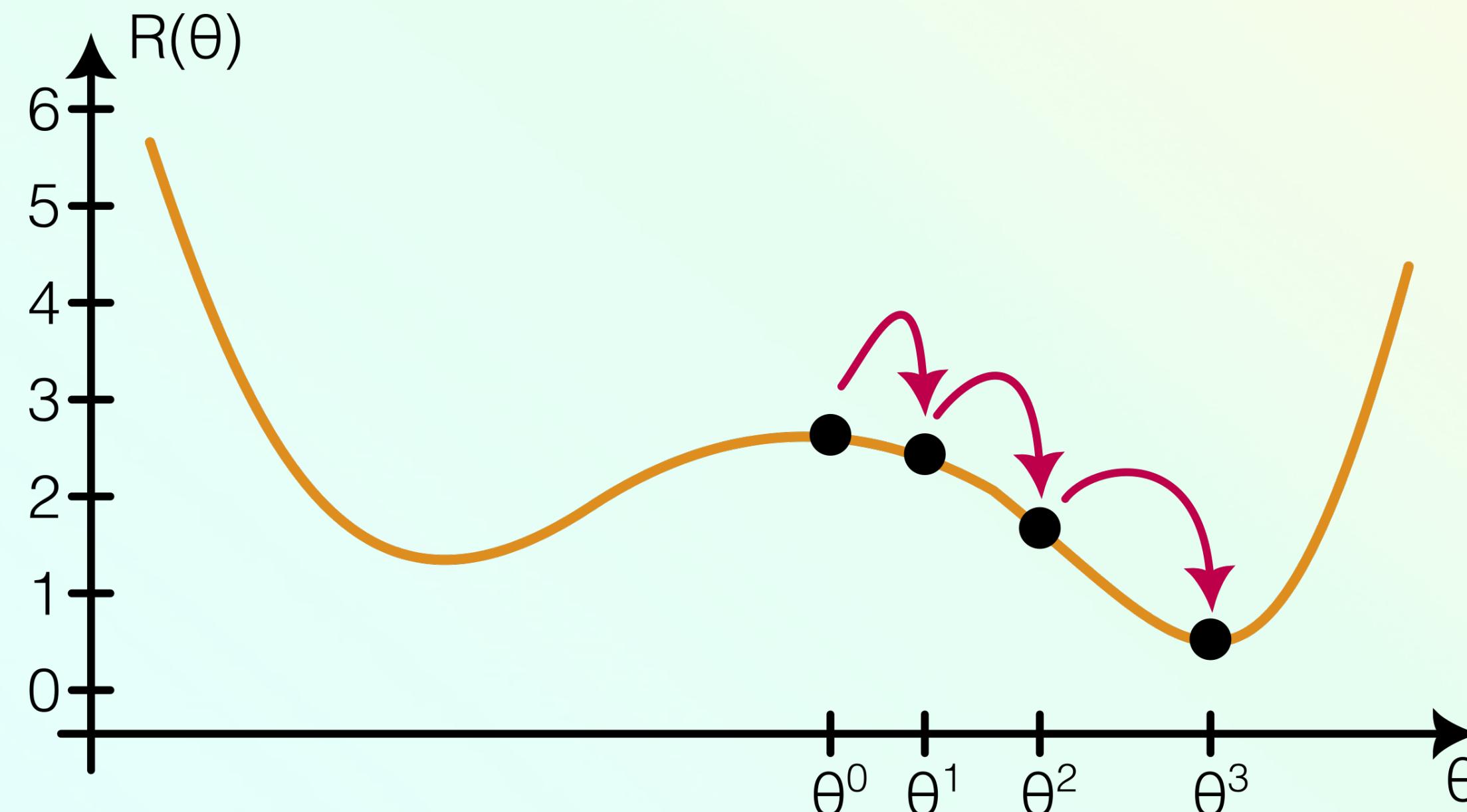
$$R(\theta) = \frac{1}{2} \sum_{i=1}^n (y_i - f(X_i))^2$$

ENTRENAMIENTO

$$R(\theta) = \frac{1}{2} \sum_{i=1}^n (y_i - f_\theta(X_i))^2$$

La idea de gradiente descendiente es:

1. Comenzar con un valor de θ al azar θ^0 en $t=0$
2. Iterar hasta que $R(\theta) = \frac{1}{2} \sum_{i=1}^n (y_i - f_\theta(X_i))^2$ deja de crecer o llega a un valor objetivo:
 1. Buscar un vector δ que refleje un pequeño cambio en θ , de tal forma que $\theta^{t+1} = \theta^t + \delta$ reduzca a $R(\theta)$.
 2. Cambiar t a $t+1$, e iterar.



BACKPROPAGATION

Bien, tenemos la idea, ¿ahora como sabemos en qué dirección mover a θ para decrecer a $R(\theta)$?

Eso lo podemos ver usando el gradiente de $R(\theta)$, evaluada en un punto θ^m , que es el vector de las derivadas parciales:

$$\nabla R(\theta^m) = \frac{\delta R(\theta)}{\delta \theta} \Big|_{\theta=\theta_m}$$

El gradiente nos da la dirección en el espacio θ en donde $R(\theta)$ incrementa más cuando estamos parado en θ^m . La idea en gradiente descendiente es movernos un pasito en la dirección contraria:

$$\theta^{m+1} = \theta^m - \rho \nabla R(\theta^m)$$

BACKPROPAGATION

ρ es la constante de aprendizaje, que es un valor que elegimos para determinar qué tan rápido queremos que sea el entrenamiento.

Obtengamos el gradiente para nuestra red, volvamos a R:

$$R(\theta) = \frac{1}{2} \sum_{i=1}^n (y_i - f(X_i))^2$$

Esto es una suma de n observaciones, por lo tanto, el gradiente es una suma de n. Cada término de la suma de R la podemos escribir como:

$$R_i(\theta) = \frac{1}{2} \left(y_i - \beta - \sum_{k=1}^K \omega_k g \left(\sum_{i=1}^n x_i w_{ki} + b_k \right) \right)^2$$

BACKPROPAGATION

ρ es la constante de aprendizaje, que es un valor que elegimos para determinar qué tan rápido queremos que sea el entrenamiento.

Obtengamos el gradiente para nuestra red, volvamos a R:

$$R(\theta) = \frac{1}{2} \sum_{i=1}^n (y_i - f(X_i))^2$$

Esto es una suma de n observaciones, por lo tanto, el gradiente es una suma de n. Cada término de la suma de R la podemos escribir como:

$$R_i(\theta) = \frac{1}{2} \left(y_i - \beta - \sum_{k=1}^K \omega_k g \left(\sum_{i=1}^n x_i w_{ki} + b_k \right) \right)^2$$

Para simplificar creamos la siguiente variable intermedia

$$z_{ik} = \sum_{i=1}^n x_i w_{ki} + b_k$$

BACKPROPAGATION

Realizamos la derivada parcial de cada peso sináptico, empezamos por los pesos de la última capa:

$$\frac{\delta R_i(\theta)}{\delta \beta_k} = \frac{\delta R_i(\theta)}{\delta f_\theta(x_i)} \frac{\delta f_\theta(x_i)}{\delta \beta_k} = -(y_i - f_\theta(x_i))g(z_{ik})$$

Y los pesos de la capa oculta:

$$\frac{\delta R_i(\theta)}{\delta w_{kj}} = \frac{\delta R_i(\theta)}{\delta f_\theta(x_i)} \frac{\delta f_\theta(x_i)}{\delta g(z_{ik})} \frac{\delta g(z_{ik})}{\delta z_{ik}} \frac{\delta z_{ik}}{\delta w_{kj}} = -(y_i - f_\theta(x_i))\beta_k g'(z_{ik})x_{ij}$$

Notamos que el residuo aparece en todas las derivadas. Este traslado del residuo desde la salida para encontrar los valores es lo que llamamos **backpropagation**. Ya que los pesos de la red se modifican paso a paso trasladando desde la última capa hacia atrás.

BACKPROPAGATION

Entonces el algoritmo de gradiente descendiente mediante **backpropagation** lo podemos simplificar como:

1. Comenzar con un valor de θ al azar θ^0
2. Bajáramos al azar las observaciones del set de entrenamiento. Elegimos la primera observación.
 1. Calculamos el valor del gradiente con las fórmulas que vimos y la observación elegida.
 2. Modificamos los pesos haciendo $\theta^{m+1} = \theta^m - \rho \nabla R(\theta^m)$
 3. Si llegamos a un valor umbral terminamos.
 4. Si no, elegimos la siguiente observación y vamos al paso 1 de este sub-ítem. Si es la última observación, volvemos a barajar el set de observaciones y comenzamos de nuevo.

BACKPROPAGATION

Este algoritmo tiene un problema el cual es muy lento para set de entrenamientos muy largos, ya que debe pasar por todos los valores y evaluar a R cada vez.

Una forma que podemos acelerar esto es submuestreando en batch (**minibatch**) de m observaciones,

Para cada minibatch, se calculan los gradientes de las m observaciones y los sumamos. Luego evaluamos R.

De esta forma se acelera el proceso porque no estamos calculando R cada paso, y esto acelera el proceso, pero aumentamos la probabilidad de no llegar a un resultado más optimo (pasamos a usar gradiente descendiente estocástico).

PYTORCH

PYTORCH

PyTorch es una librería de Deep Learning. Es mantenida por Meta y es la principal competencia de otra famosa librería (tensorflow de Google).

Tiene un API para Python, como también para C++. PyTorch nos da toda una infraestructura de:

- Computación de tensores. Es similar a los arrays de Numpy pero con posibilidad de usarlos en GPU.
- Redes neuronales profundas.

Aquí vamos a usar lo básico, esta librería lo verán en más profundidad en otras asignaturas.

VAMOS A PRÁCTICAR UN POCO...