


# **Resumen acerca de la primera previa**

**Jorge Miguel Coronado  
Kevin Mauricio Carmona**



# I. Paper sobre Perceptrón y Backpropagation

Principalmente se realizó el desarrollo de un paper basado en el perceptrón y el backpropagation, en cual los definimos, hablamos un poco sobre lo que son y su funcionamiento.





# Perceptron

## Ventajas

- Algoritmos de propósito general
- Gran éxito en la práctica
- Fácilmente implementables
- Fácilmente paralelizables

## Desventajas

- Son algoritmos aproximados, no exactos
- Son no determinísticos(probabilísticos)
- No siempre existe una base teórica establecida.



# BACKPROPAGATION


Para el backpropagation tenemos que es un método para el cálculo de gradiente utilizado en algoritmos de aprendizaje supervisado para redes neuronales artificiales.

Teníamos también que también es sencillo ya que se basa en alimentarlo con múltiples muestras de información, y ya después cada neurona mostraría sus pesos  $W$  y ahí ya se empezaría a trabajar..



## 2. Código en colab

En este punto seleccionamos un código de lógica difusa, para ser más exactos el programa 20 de control difuso.



**El cual consiste básicamente en solucionar el problema de un cliente que recibe un plato de comida con cierta calidad, teniendo también en cuenta la calidad del servicio.**

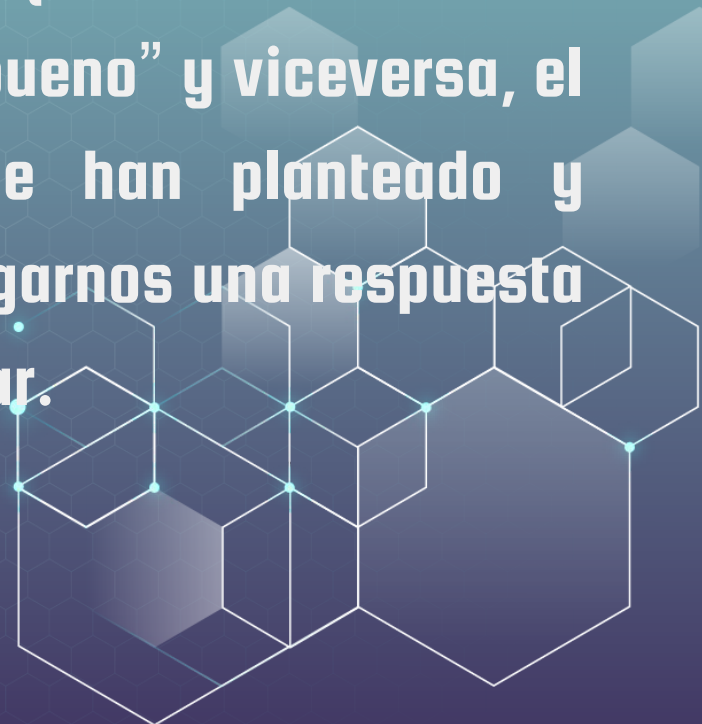
---



**Con estos dos factores antes mencionados Obtenemos la cantidad de propina que debemos de dar, de acuerdo a lo proporcionado por las gráficas tanto de servicio como de calidad de la comida**



cabe aclarar que no solo será buena o mala, utilizando estas dos premisas como base, “Que el servicio sea malo pero el plato de comida sea bueno” y viceversa, el programa con las reglas que le han planteado y mediante lógica difusa, podrá otorgarnos una respuesta óptima de acuerdo a la propina a dar.





Entonces procedemos a evaluar lo que es el código del programa

Como grupo de trabajo, estudiamos este, y logramos comprender su funcionalidad, entonces después de realizado esto procedimos a evaluar creando nuestros propios criterios, con una variable llamada análisis creamos la población de la función con un .automf(5) y trabajamos con las reglas que había ya planteadas, procedimos a mostrar su gráfica con el .view y ejecutamos y el resultado obtenido fue el siguiente:

```
# variables del universo y las funciones de membresía
calidad = ctrl.Antecedent(np.arange(0, 11, 1), 'calidad')
servicio = ctrl.Antecedent(np.arange(0, 11, 1), 'servicio')
propina = ctrl.Consequent(np.arange(0, 26, 1), 'propina')
analisis = ctrl.Antecedent(np.arange(0, 11, 1), 'prueba')

# La población de la función de membresía automática es posible con .automf (3, 5 o 7)
calidad.automf(3)
servicio.automf(3)
analisis.automf(5)

# Las funciones de membresía personalizadas se pueden construir interactivamente con la
# API Pythonic
propina['bajo'] = fuzz.trimf(propina.universe, [0, 0, 13])
propina['medio'] = fuzz.trimf(propina.universe, [0, 13, 25])
propina['alto'] = fuzz.trimf(propina.universe, [13, 25, 25])

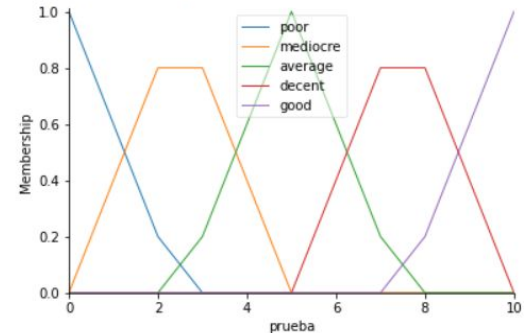
# Visualización con .view()
analisis.view()
calidad['average'].view()
servicio.view()
propina.view()

# Creación de las reglas
regla1 = ctrl.Rule(calidad['poor'] | servicio['poor'], propina['bajo'])
regla2 = ctrl.Rule(servicio['average'], propina['medio'])
regla3 = ctrl.Rule(servicio['good'] | calidad['good'], propina['alto'])

# Visualización de la regla 1
regla1.view()

# Generación del simulador
control_propina = ctrl.ControlSystem([regla1, regla2, regla3])
asignacion_propina = ctrl.ControlSystemSimulation(control_propina)
```

Valor de la propina:  
19.847607361963192





**GRACIAS!!!**