4.1 Introducción

Los **sistemas conexionistas** o **Redes Neuronales Artificiales (RNA)** son modelos computacionales inspirados en el funcionamiento del cerebro humano. Representan una **dualidad científica y tecnológica** en la Inteligencia Artificial:

- **Bioinspiración**: Imitan la estructura y funcionamiento de las neuronas biológicas
- **Tecnoinspiración**: Crean soluciones tecnológicas prácticas basadas en estos principios

4.2 De la Neurona Biológica a la Artificial

4.2.1 La Neurona Biológica

Una neurona biológica es una célula especializada que: - Recibe señales eléctricas a través de **dendritas** - Procesa estas señales en el **soma** (cuerpo celular) - Si la suma de señales supera un umbral, genera un **potencial de acción** - Transmite la señal a otras neuronas a través del **axón**

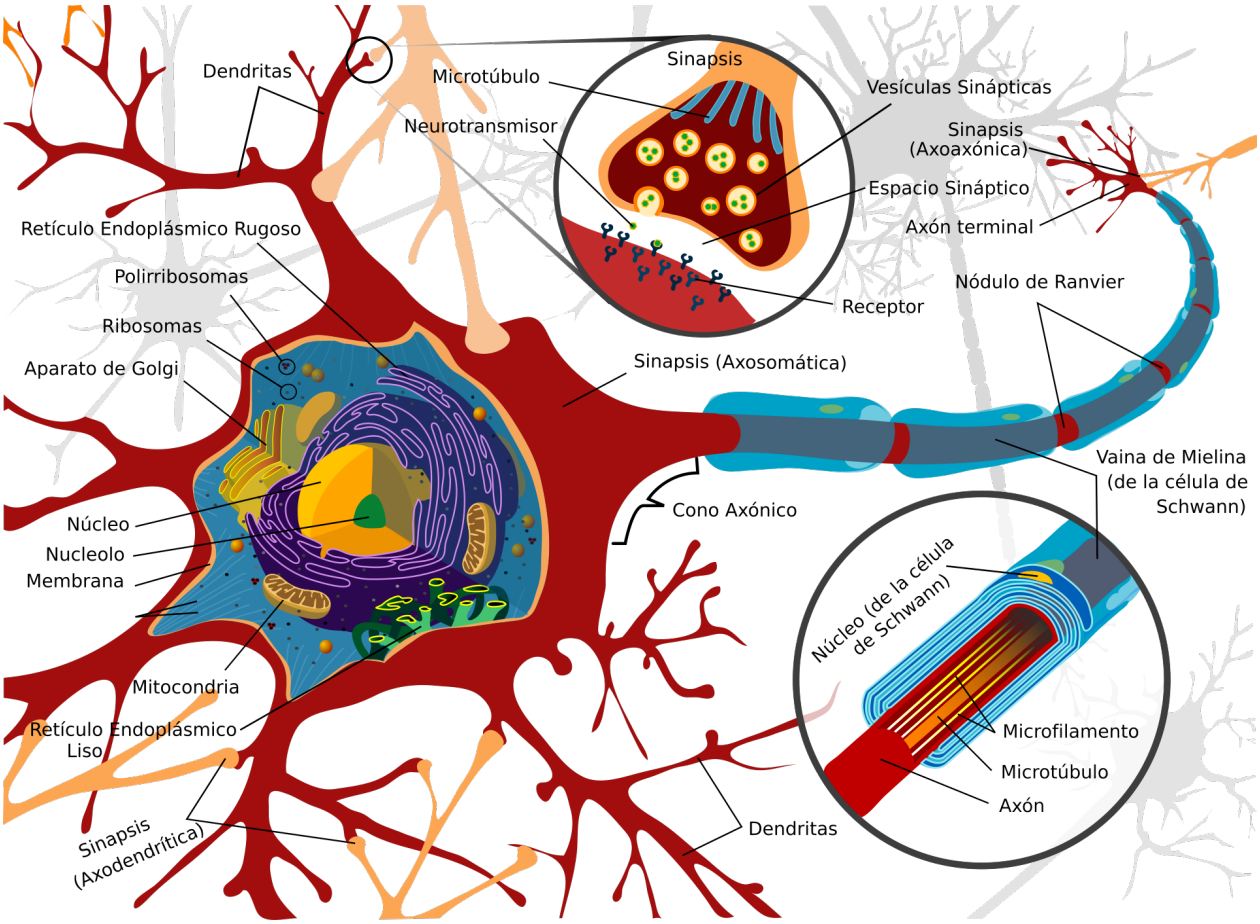


Figure 1: Pasted image 20251102174452.png

Concepto clave: El potencial de acción es una respuesta "todo o nada": o la neurona

se activa completamente, o no se activa.

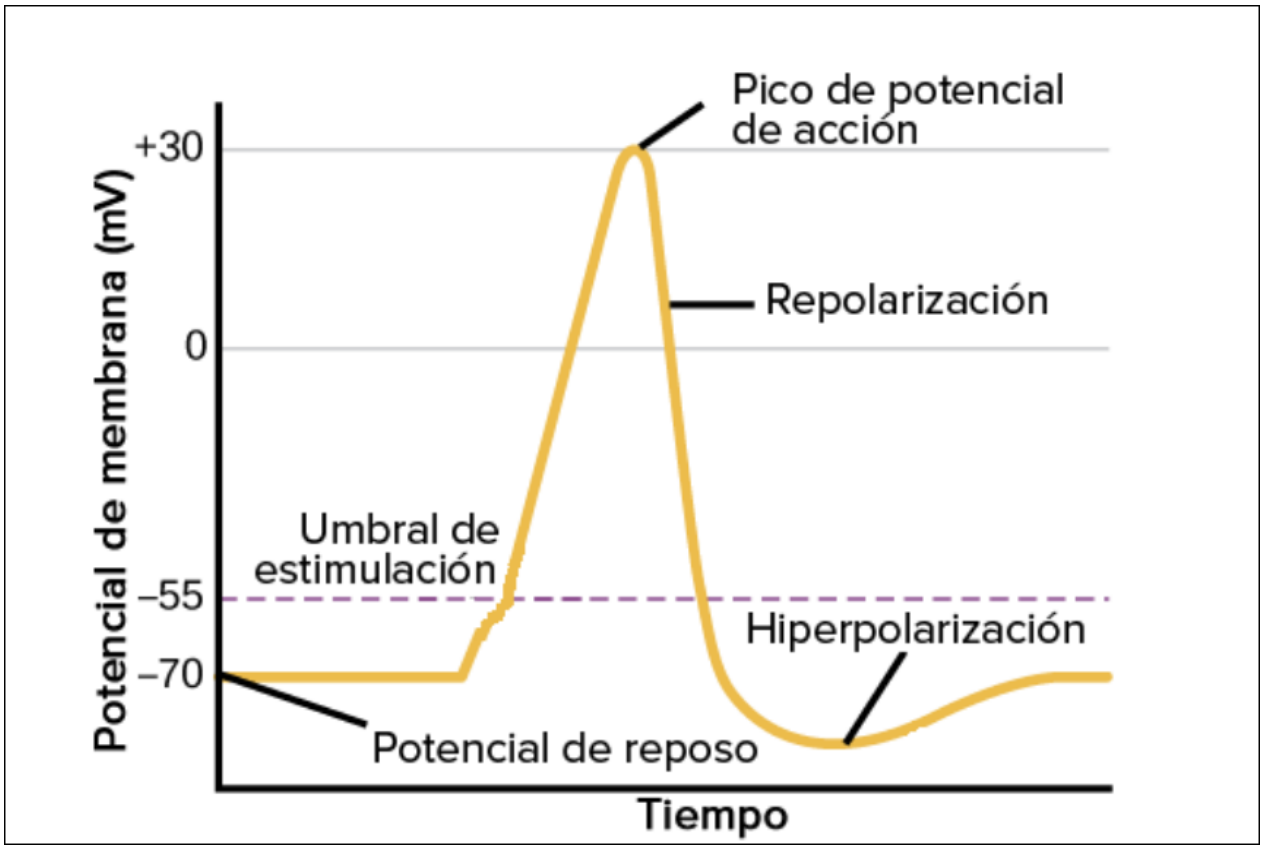


Figure 2: Pasted image 20251102174623.png

2.2 La Neurona Artificial: Modelo Simple

La neurona artificial es una abstracción matemática que replica este comportamiento:

Componentes básicos: 1. **Entradas (x_1, x_2, …, x_n)**: Señales que recibe la neurona 2. **Pesos (w_1, w_2, …, w_n)**: Importancia de cada entrada (equivalente a las sinapsis) 3. **Suma ponderada**: $\Sigma(w_i \times x_i)$ 4. **Función de activación**: Determina la salida según el resultado de la suma 5. **Salida (y)**: Resultado final de la neurona

Fórmula básica:

```
y = f(Σ(w  × x ) + bias)
```

4.3 Tipos de Neuronas Artificiales

4.3.1 Neurona Binaria con Umbral

Funcionamiento: - Si $\Sigma(w_i \times x_i) \geq$ umbral → salida = 1 - Si $\Sigma(w_i \times x_i) <$ umbral → salida = 0

Ejemplo práctico:

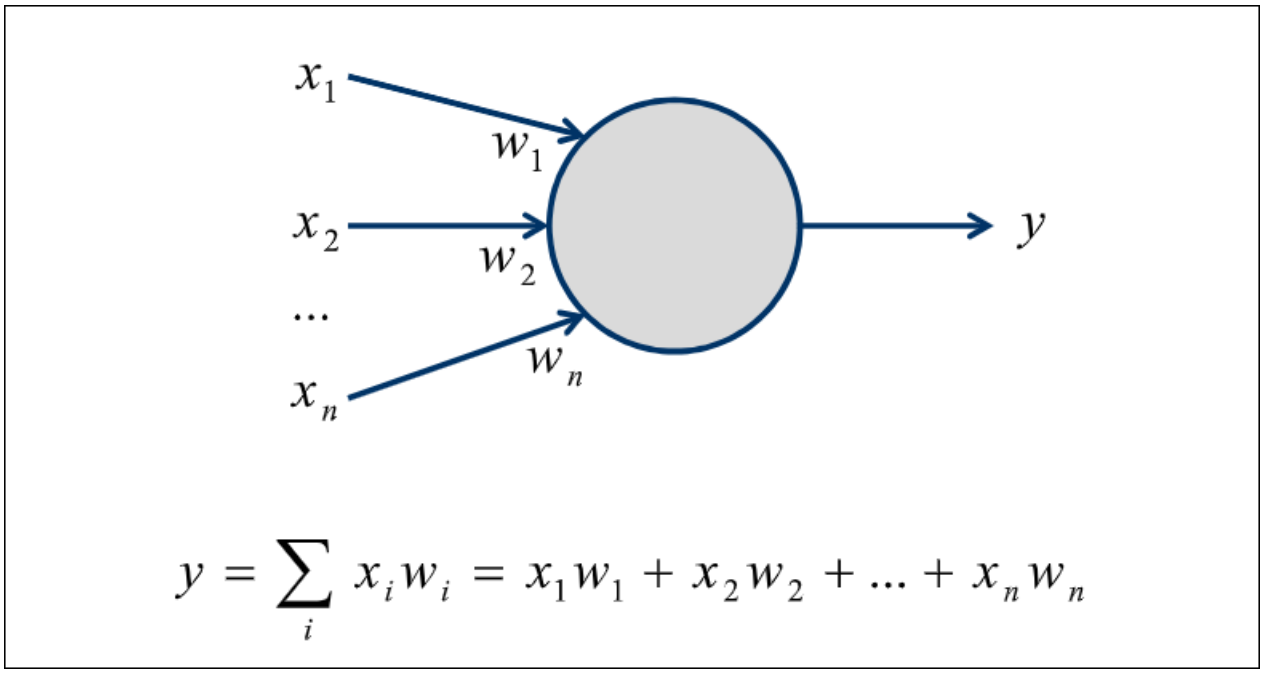


Figure 3: Pasted image 20251102174752.png

```
Entradas: x =1, x =0, x =1
Pesos: w =0.5, w =0.3, w =0.4
Umbral = 0.7

Cálculo: (1×0.5) + (0×0.3) + (1×0.4) = 0.9
Como 0.9   0.7 → salida = 1
```

4.3.2 Neurona Lineal Rectificada (ReLU)

Función de activación:

```
f(x) = max(0, x)
```

Características: - Si entrada < 0 → salida = 0 - Si entrada ≥ 0 → salida = entrada - Muy utilizada en redes profundas modernas por su simplicidad computacional

4.3.3 Neurona Sigmoidal

Función de activación:

```
f(x) = 1 / (1 + e^(-x))
```

Características: - Salida entre 0 y 1 (continua) - Forma de “S” - Útil para probabilidades y funciones diferenciables - Permite gradientes para aprendizaje

4.4 Representación con Bias

El **bias** es un término adicional que permite ajustar el umbral de activación:

Sin bias integrado:

```
y = f(Σ(w  × x ) + b)
```

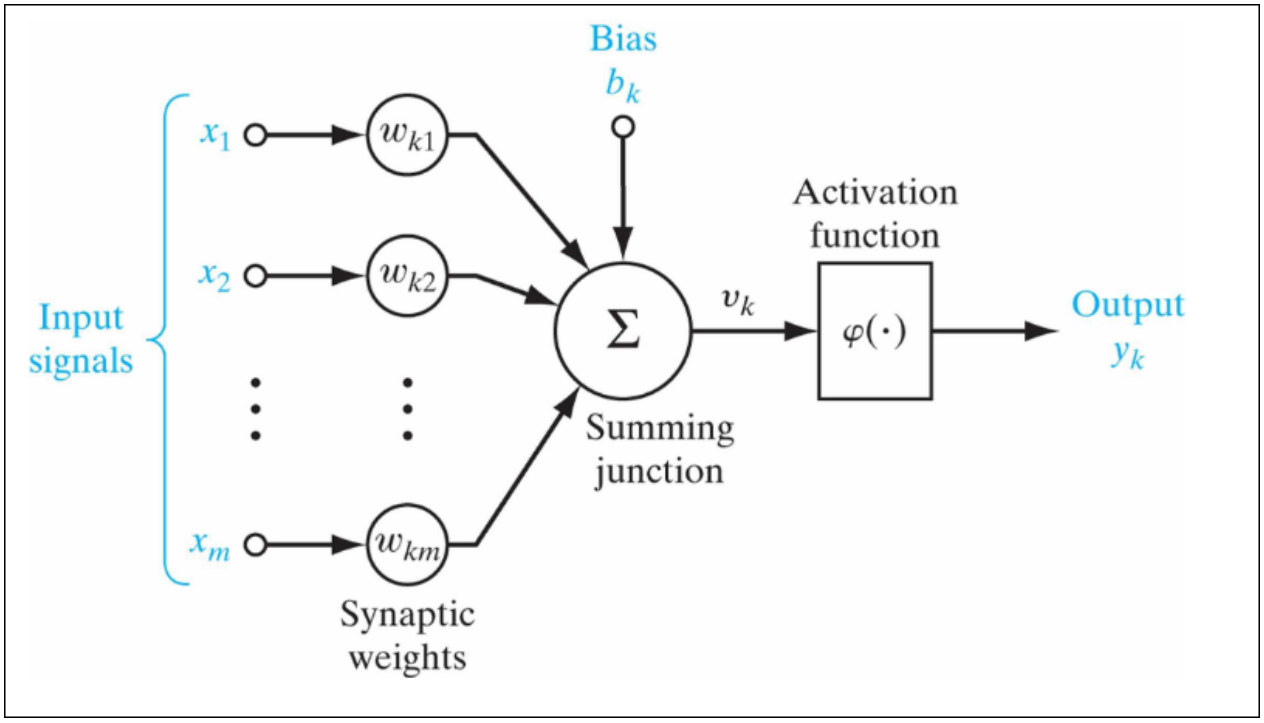


Figure 4: Pasted image 20251102175036.png

Con bias integrado (técnica común): - Se añade una entrada adicional $x_0 = 1$ - Su peso asociado w_0 actúa como bias - Simplifica la notación: $y = f(\Sigma(w_i \times x_i))$ donde i va de 0 a n

4.5 Arquitecturas de Redes Neuronales

4.5.1 Perceptrón Simple

Estructura: - Una capa de entrada - Una capa de salida (una o más neuronas) - Sin capas ocultas

Limitación fundamental: Solo puede resolver problemas **linealmente separables**

4.5.2 Red Multicapa Feedforward (hacia adelante)

Estructura: - Capa de entrada - Una o más **capas ocultas** - Capa de salida - Conexiones solo hacia adelante (sin ciclos)

Ventaja: Puede resolver problemas **no linealmente separables**

Algoritmo de entrenamiento: Backpropagation (1986)

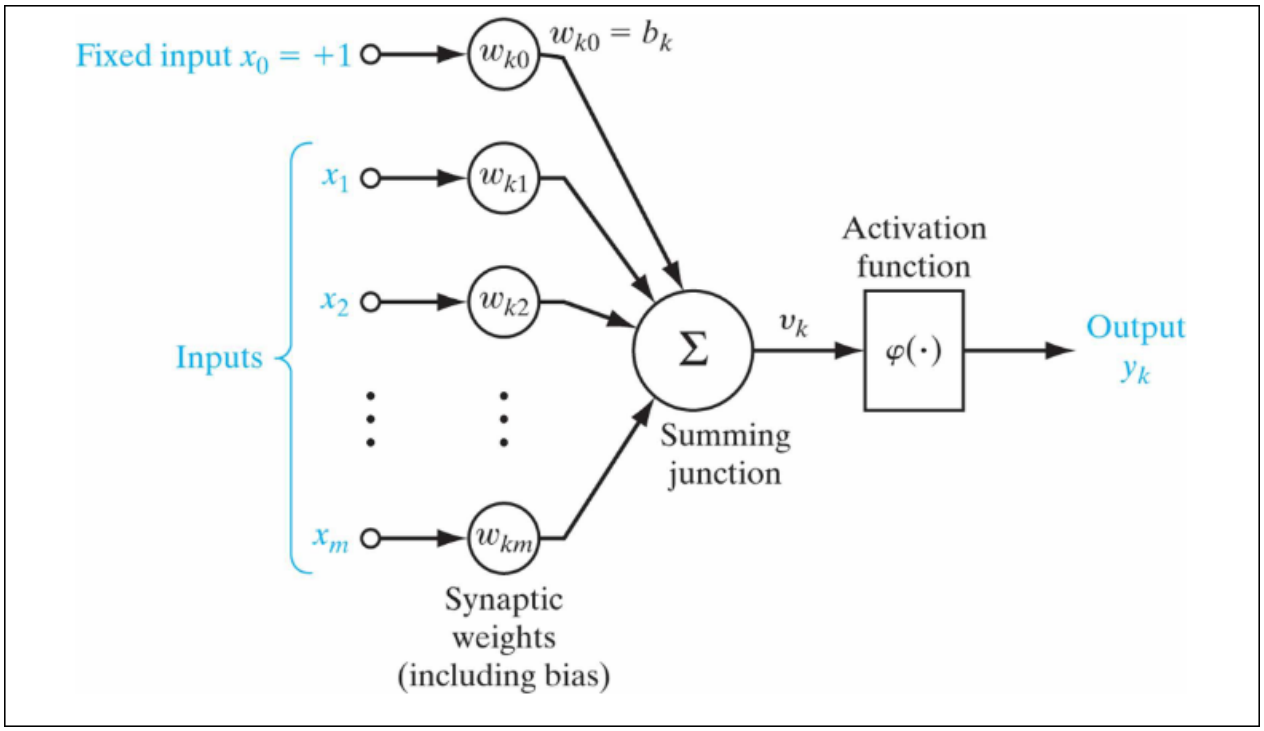


Figure 5: Pasted image 20251102175126.png

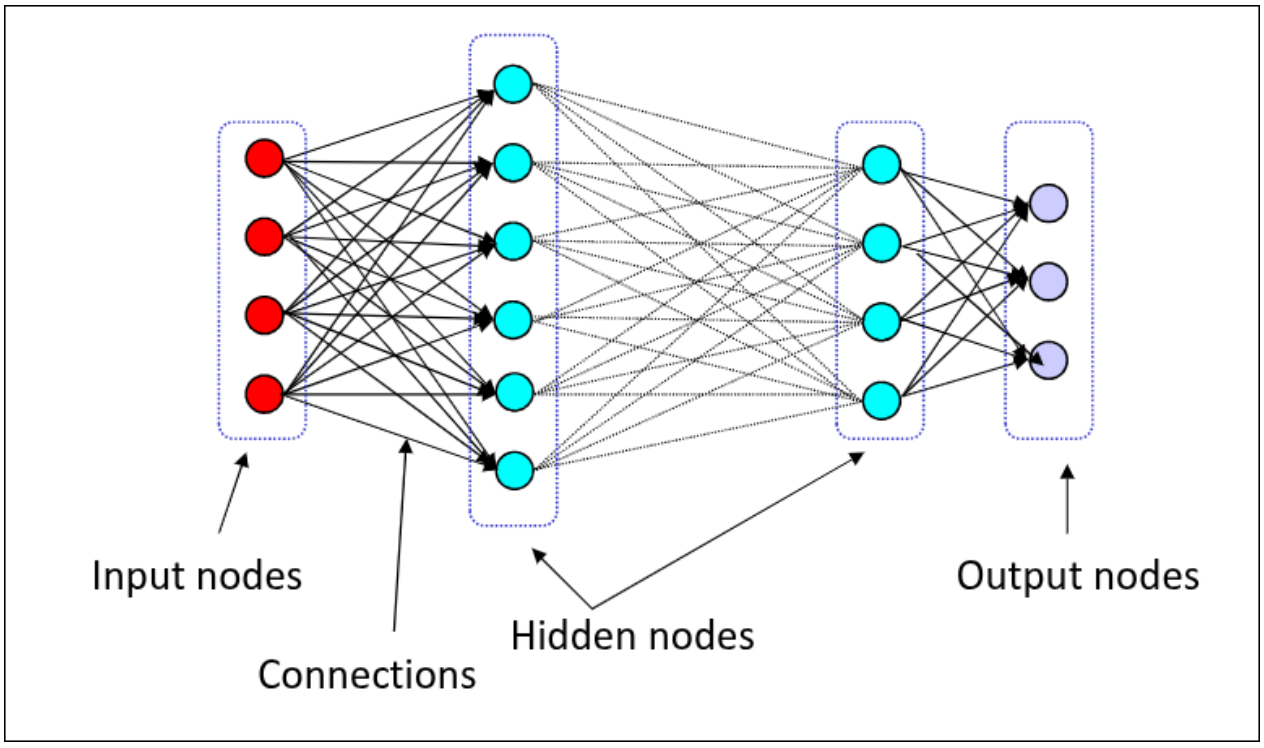
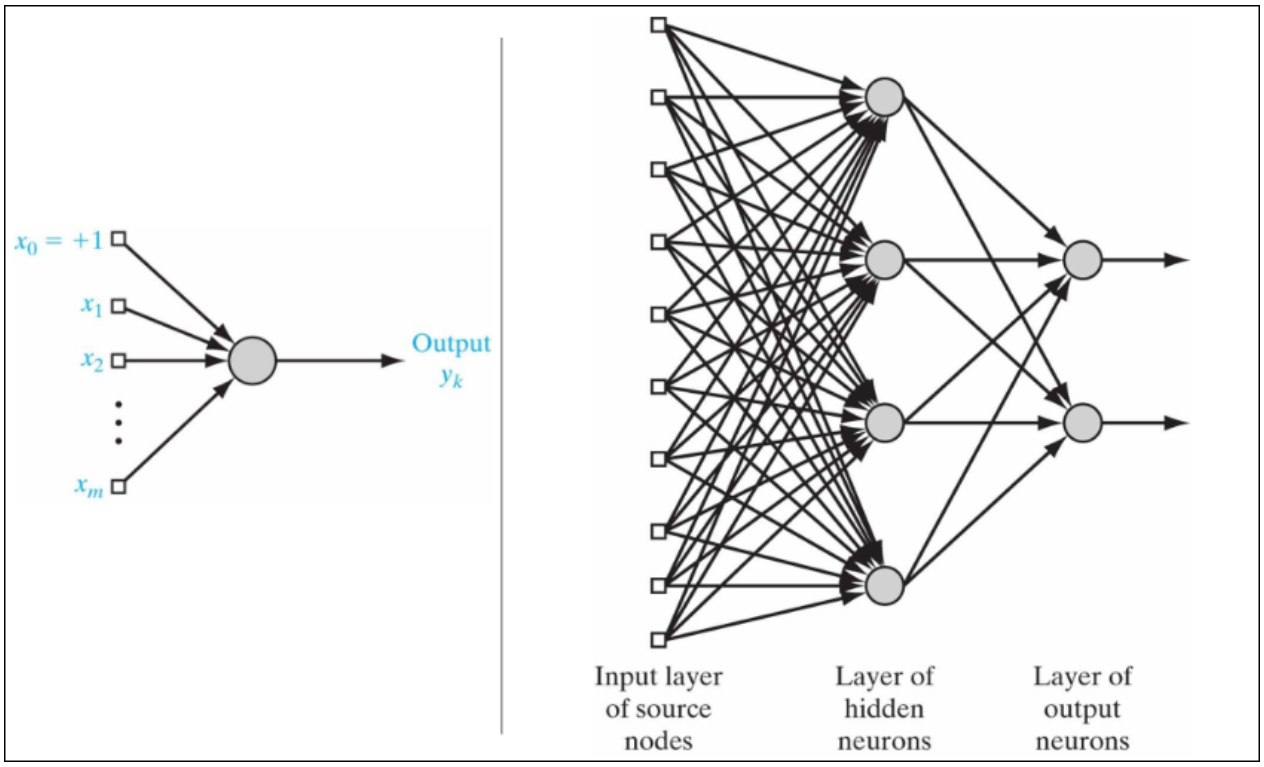


Figure 6: Pasted image 20251102175342.png



4.5.3 Redes Recurrentes **Características**: - Contienen conexiones hacia atrás (ciclos) - Tienen "memoria" de estados anteriores - Útiles para secuencias y series temporales

Ejemplo: Redes de Hopfield (1982) - funcionan como memoria asociativa

4.6 Características de las RNA

1. **Estructura paralela y distribuida**:
   - Múltiples neuronas procesan información simultáneamente
   - El conocimiento se distribuye en los pesos de las conexiones
2. **Aprendizaje desde datos**:
   - No se programan explícitamente
   - Aprenden patrones a partir de ejemplos
3. **Modificación de pesos sinápticos**:
   - El aprendizaje ocurre ajustando los pesos w_i
   - Similar a la plasticidad sináptica cerebral

4.7 Aprendizaje en Redes Neuronales

4.7.1 Aprendizaje Supervisado

Proceso: 1. Se proporciona un **conjunto de entrenamiento**: pares (entrada, salida_deseada) 2. La red procesa cada entrada y produce una salida 3. Se calcula el **error**: diferencia entre salida_real y salida_deseada 4. Se ajustan los **pesos** para minimizar el error

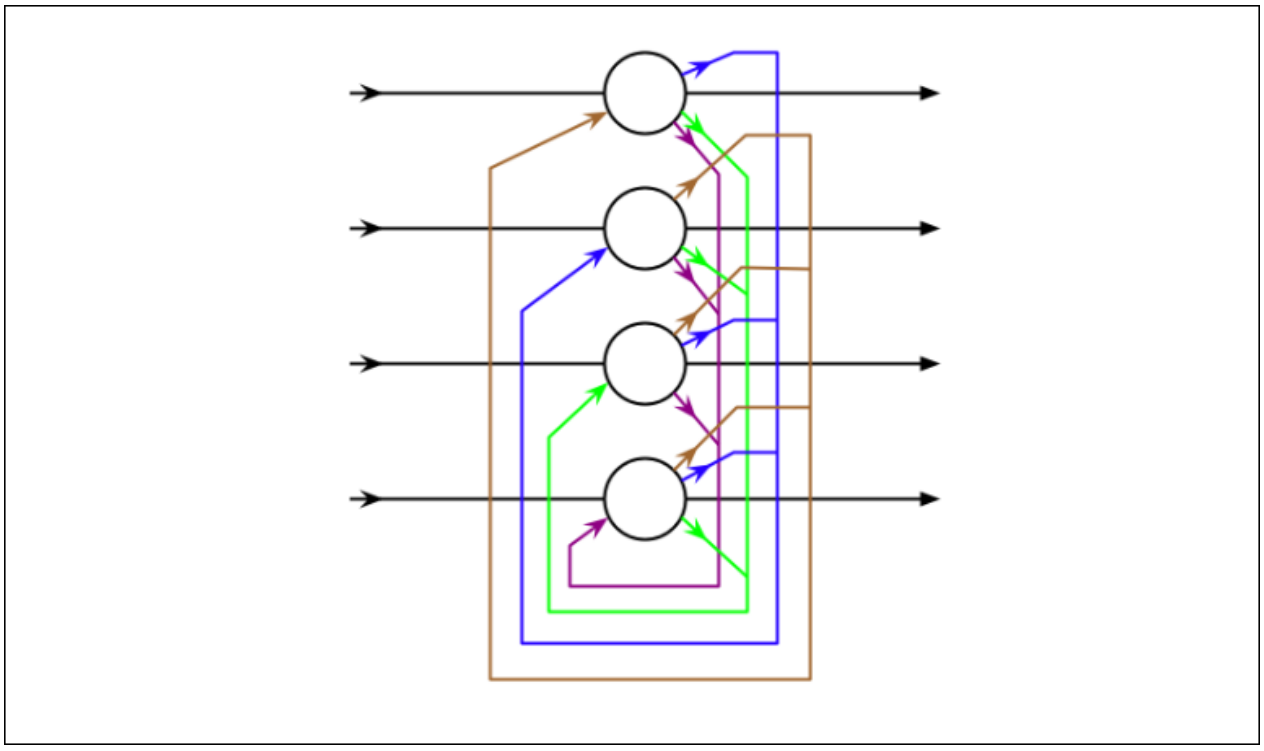


Figure 7: Pasted image 20251102175412.png

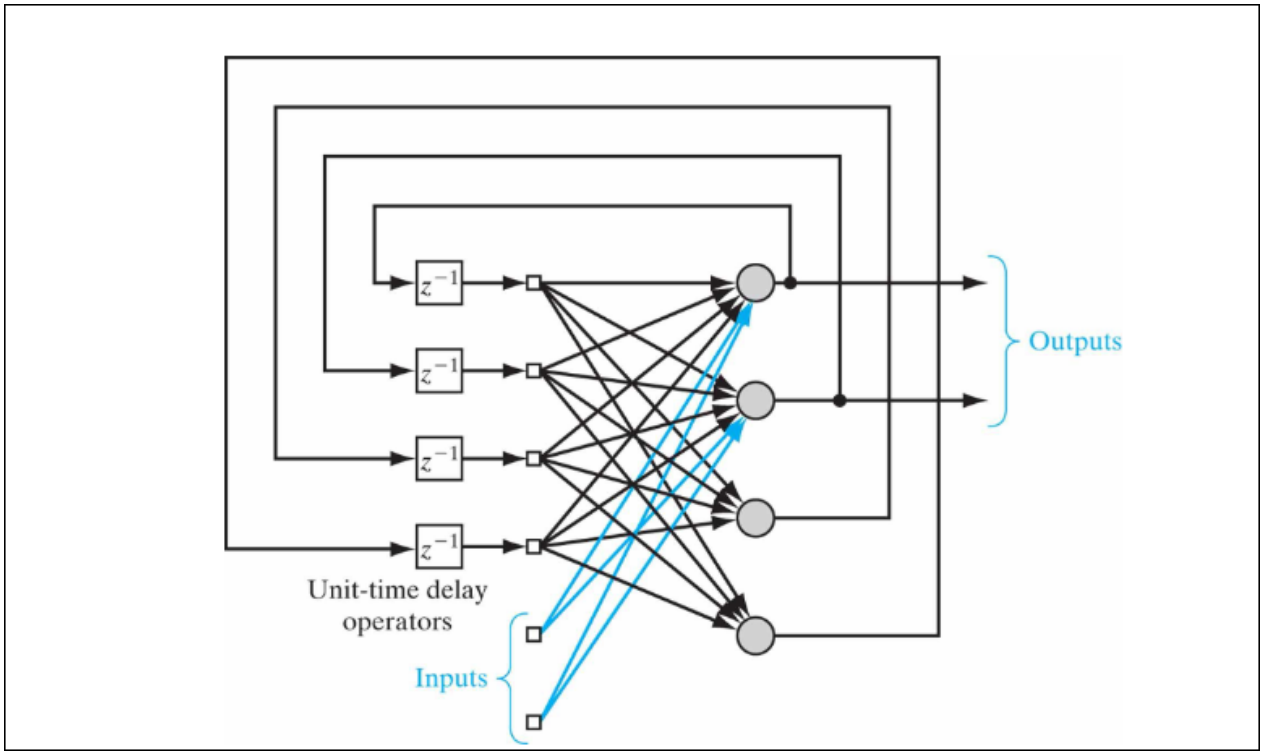


Figure 8: Pasted image 20251102180310.png

Ejemplo:

```
Problema: Clasificar emails como spam (1) o no spam (0)

Conjunto de entrenamiento:
- ("oferta increíble", "compra ahora") → 1 (spam)
- ("reunión mañana a las 10") → 0 (no spam)
- ("gana dinero fácil") → 1 (spam)
```

4.7.2 Algoritmo de Convergencia del Perceptrón

Pasos básicos:

```
1. Inicializar pesos aleatoriamente
2. Para cada ejemplo (x, d) del conjunto de entrenamiento:
   a. Calcular salida: y = f(Σ(w  × x ))
   b. Calcular error: e = d − y
   c. Actualizar pesos: w (nuevo) = w (antiguo) +   × e × x
   (donde   es la tasa de aprendizaje)
3. Repetir hasta que no haya errores o se alcance un máximo de iteraciones
```

Propiedades: - **Garantiza convergencia** si existe solución (problema linealmente separable) - **No converge** si el problema no es linealmente separable

4.8 Separabilidad Lineal

4.8.1 Clases Linealmente Separables

Definición: Dos clases son linealmente separables si puedes dibujar una **línea recta** (en 2D) o un **hiperplano** (en dimensiones superiores) que las separe completamente.

Ejemplo visual (2D):

```
Clase A:
                          ← Línea separadora
Clase B:
```

Funciones lógicas linealmente separables: - AND: - OR: - NOT:

4.8.2 Clases No Linealmente Separables

Definición: No existe ninguna línea recta que pueda separar las clases.

Ejemplo: Función XOR

```
Tabla de verdad:
x  | x  | XOR
0  | 0  | 0
0  | 1  | 1
1  | 0  | 1
1  | 1  | 0
```

```
Representación gráfica:
  x
  1  1( )    0( )

  0  0( )    1( )
           x
    0          1
```

Problema: No hay una línea recta que separe los 1s de los 0s.

Solución: - Un perceptrón simple **NO puede** resolver XOR - Se necesita una **red multicapa** con al menos una capa oculta

4.9 Backpropagation: Entrenando Redes Multicapa

Backpropagation (retropropagación) es el algoritmo que permite entrenar redes con capas ocultas.

Proceso: 1. **Forward pass** (hacia adelante): - Los datos fluyen desde la entrada hasta la salida - Se calcula la salida de cada neurona capa por capa

2. **Cálculo del error**:
   - Se compara la salida obtenida con la salida deseada
3. **Backward pass** (hacia atrás):
   - El error se propaga hacia atrás
   - Se calcula la contribución de cada peso al error total
   - Se usan derivadas (gradientes) para determinar cómo ajustar cada peso
4. **Actualización de pesos**:
   - Cada peso se ajusta en dirección opuesta al gradiente (descenso de gradiente)

Por qué es importante: Sin backpropagation, no podríamos entrenar redes profundas modernas.

4.10 Resumen de Conceptos Clave

Concepto	Explicación Simple
Neurona Artificial	Unidad básica que suma entradas ponderadas y aplica una función de activación
Pesos	Parámetros que determinan la importancia de cada entrada
Bias	Término que permite ajustar el umbral de activación
Función de activación	Transforma la suma ponderada en la salida final
Perceptrón	Red de una sola capa, solo resuelve problemas linealmente separables

Concepto	Explicación Simple
Red Multicapa	Red con capas ocultas, puede resolver problemas más complejos
Backpropagation	Algoritmo para entrenar redes multicapa mediante gradientes
Separabilidad lineal	Propiedad que determina si un perceptrón simple puede resolver el problema

Los sistemas conexionistas evolucionaron desde modelos simples (perceptrón) hasta arquitecturas complejas (deep learning). Los conceptos fundamentales son:

1. Imitación de neuronas biológicas mediante matemáticas
2. Aprendizaje mediante ajuste de pesos
3. Limitaciones del perceptrón simple (solo problemas lineales)
4. Poder de las redes multicapa (problemas no lineales)
5. Backpropagation como motor del aprendizaje profundo

Espero que esta reestructuración te ayude a comprender mejor los sistemas conexionistas. Si necesitas profundizar en algún concepto específico o más ejemplos, no dudes en preguntar.

Ejemplo Clave para Entender esta maraña de Conceptos

¿Qué es una Red neuronal

Descripción del problema

Detectar números dibujados para decir a que carácter se corresponden. Nuestro cerebro nada más ve un 3 dibujado sabe que es un 3, pero a un ordenador esto le puede costar un poco más. Por ello partimos de una cuadrícula de píxeles con diferentes valores de iluminación.

Cómo aborda la Red Neuronal el Problema

Empleando la estructura multicapa

Para ellos tenemos las neuronas de entrada, que se corresponden a cada uno de los píxeles anteriores. Estas neuronas toman un valor entre 0 y 1 y son encargadas de transmitir su valor a las neuronas de cada capa intermedia. Para ello podemos pensar el problema subdividiendo lo qué es un número para identificar patrones. Por ejemplo un 8 sería un círculo arriba un círculo abajo:

Y podemos también subdividir este subpatrón en otro más pequeño, de forma que las neuronas de la segunda capa lo detecten.

Cada una de las líneas que unen cada nodo son los pesos y toman ciertos valores para que se activen las neuronas. Nuestro objetivo es que en cada capa se activen determinadas neuronas, la función que determina si se activa o no no es más que la suma de cada termino de entrada por su correspondiente peso.

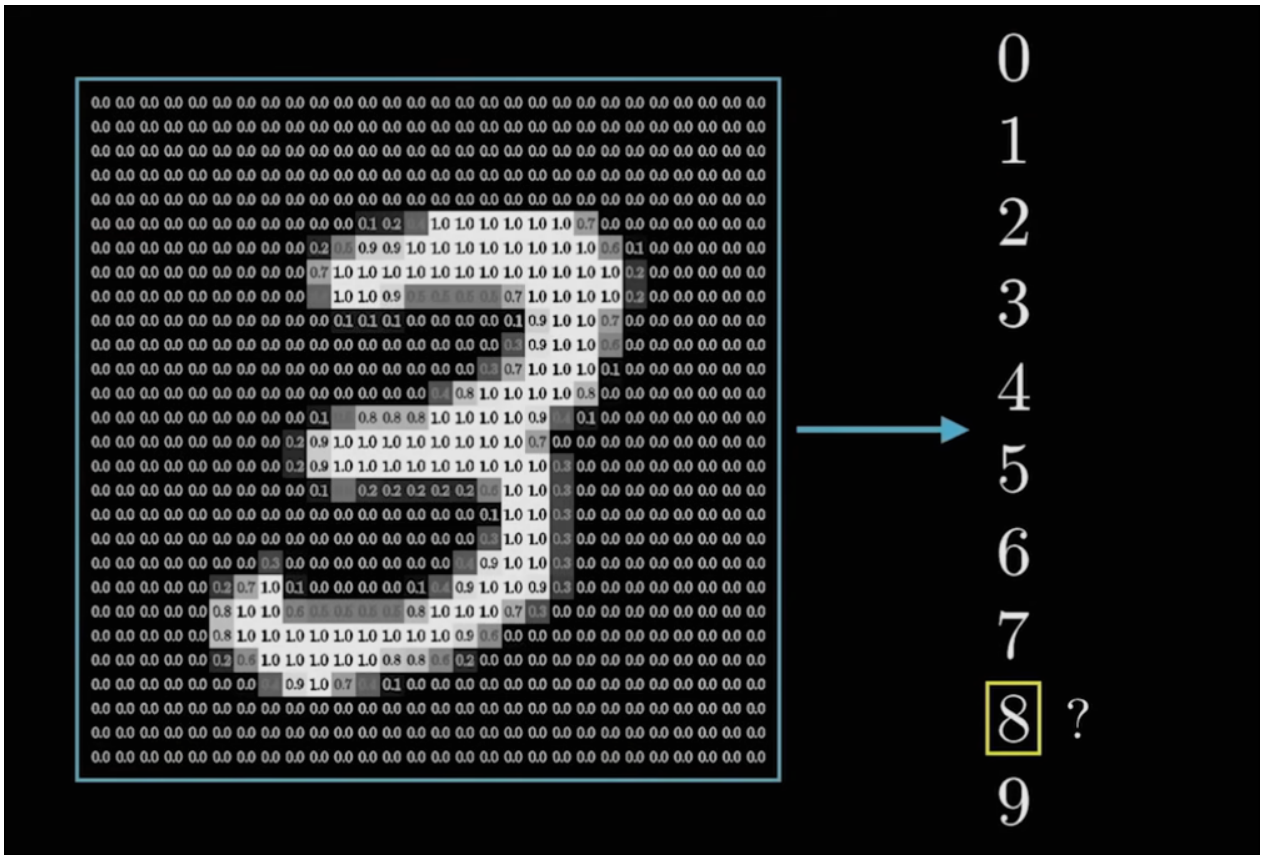


Figure 9: Pasted image 20251103104552.png

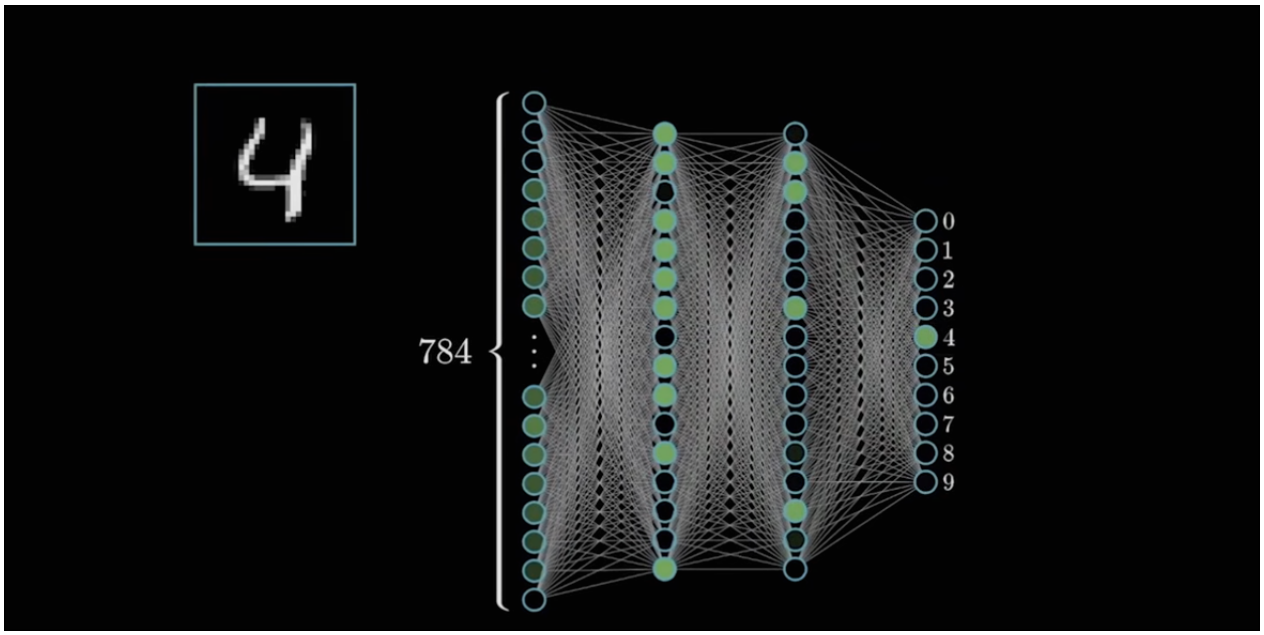


Figure 10: Pasted image 20251103104724.png

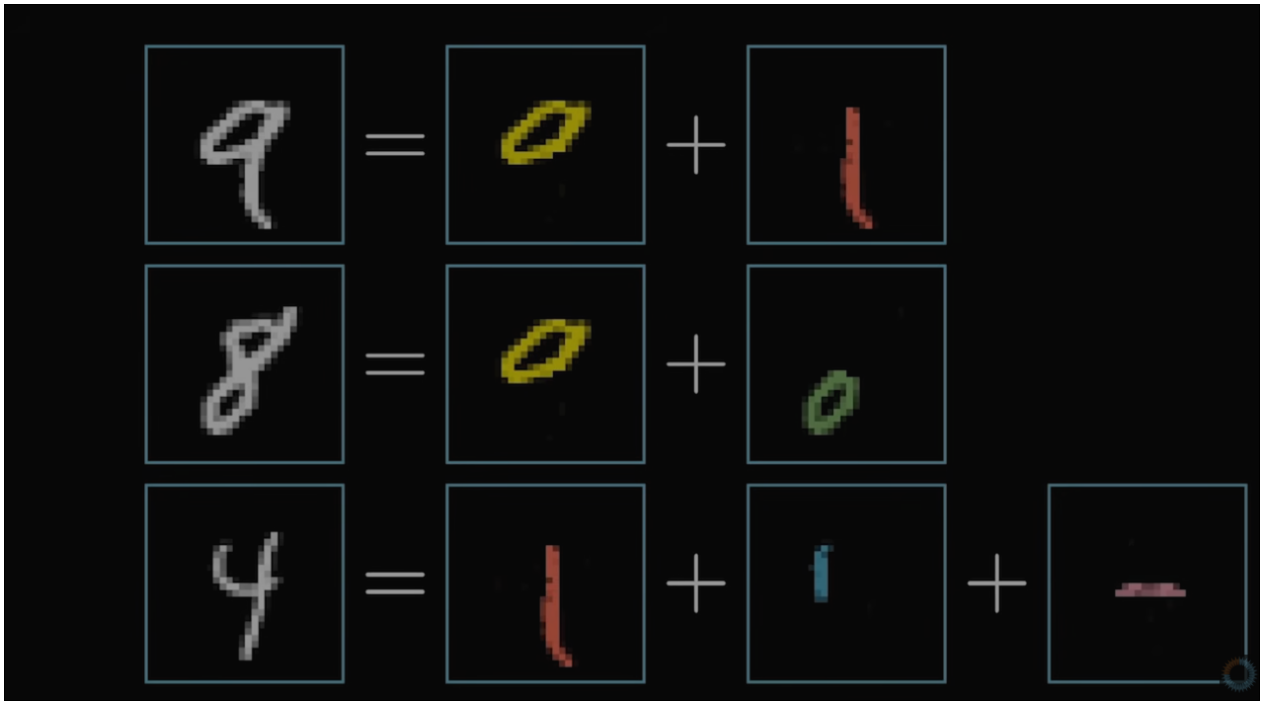


Figure 11: Pasted image 20251103105102.png

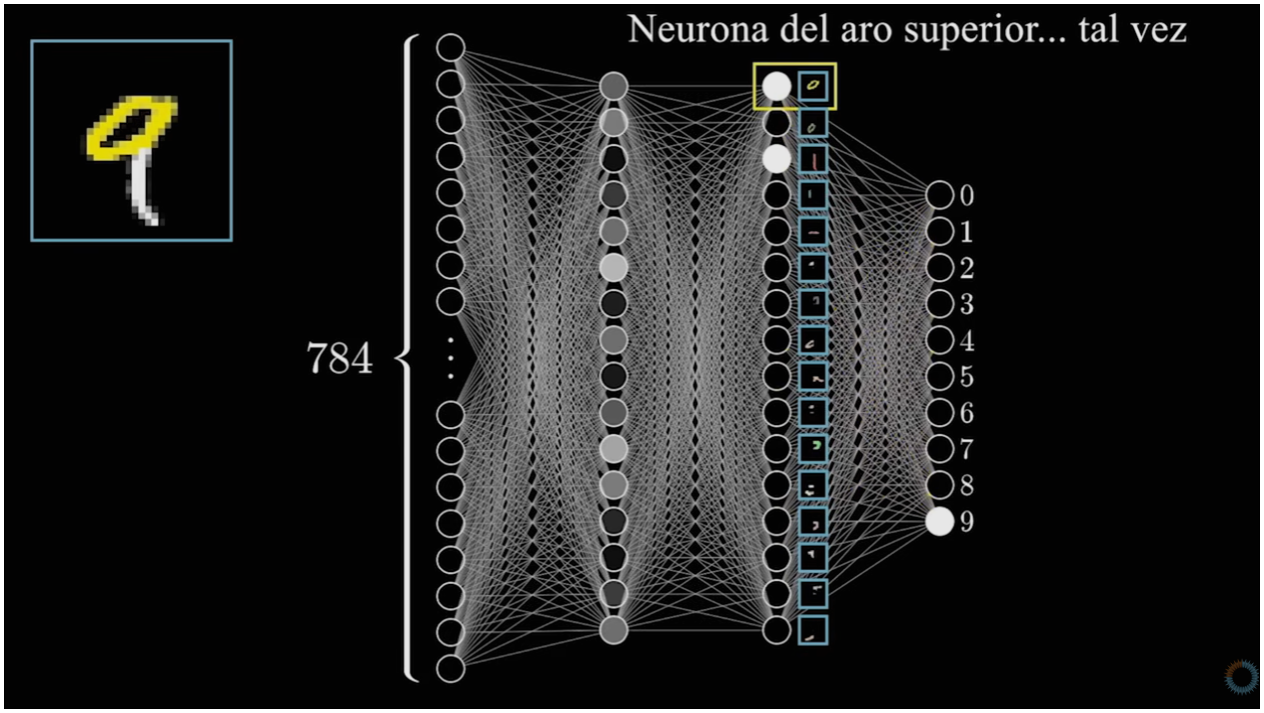


Figure 12: Pasted image 20251103105122.png

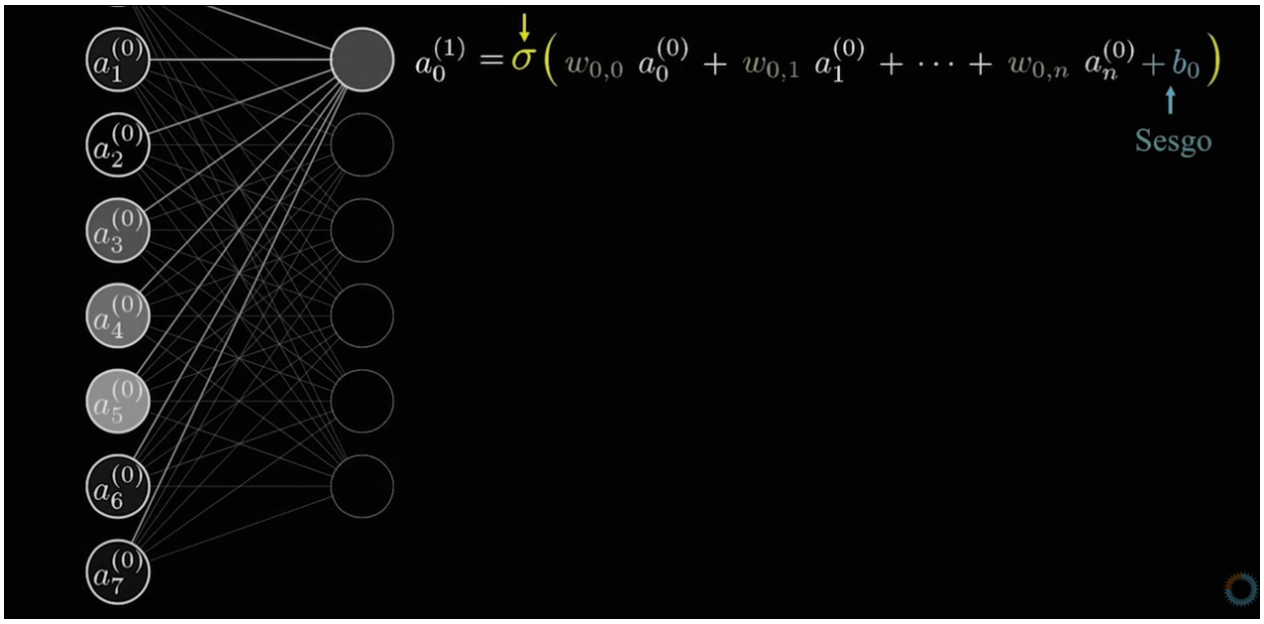


Figure 13: Pasted image 20251103105925.png

Y para calcular esos pesos, debemos entrenar a la red neuronal, usando el método del descenso gradiente. Esto nos permitirá ir modificando los pesos para que de una solución válida mientras que la red siga fallando (por ello tenemos un conjunto de datos de entrenamiento y uno de prueba). La forma de optimizar estos pesos se verá en el siguiente tema.