



Redes Neuronales (Parte II)

Clase sincrónica

Implementar ensambles de modelos en problemas complejos, ajustando diferentes factores para optimizar la predicción.

- **Unidad 1: Modelos de ensamble**
(Parte I)

(Parte II)

(Parte III)

- **Unidad 2: Redes neuronales**
(Parte I)

(Parte II)

- **Unidad 3: Procesamiento y Redes recurrentes**
(Parte I)

(Parte II)



Te encuentras aquí



¿Qué aprenderás en esta sesión?

*Entender los componentes de una red neuronal feed forward sencilla, con sus pasos forward y backward.
Comprender el uso de regularización Dropout.*

¿Cual es la diferencia
entre función de
activación y función de
pérdida?



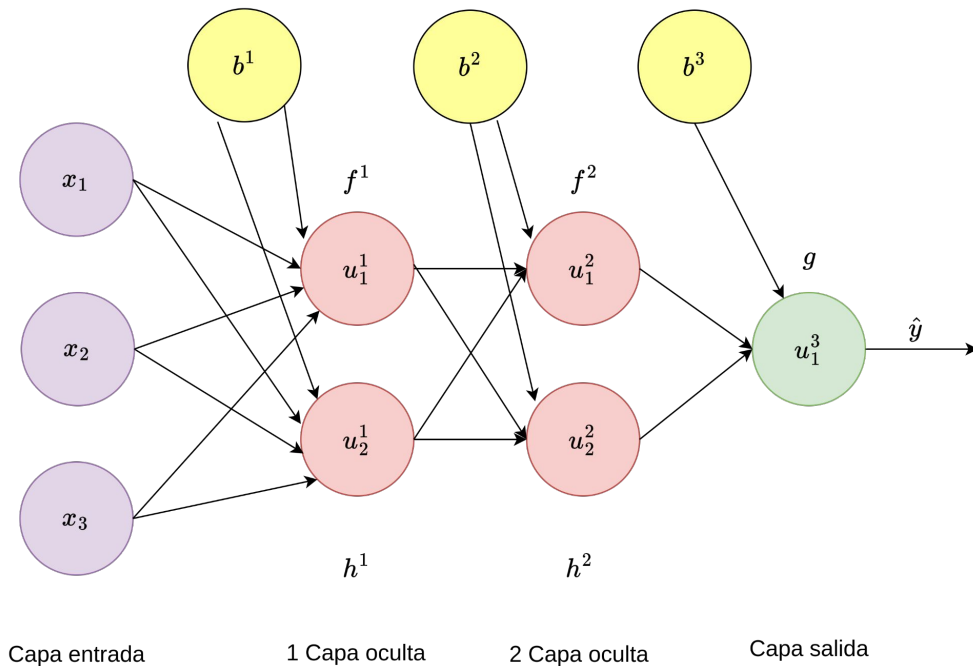
/* Redes Neuronales Multicapa*/

Red Neuronal Multicapa

Arquitectura

Cantidad neuronas por capa:

- Capa Entrada $d_0 = 3$
- 1 Capa oculta $d_1 = 2$
- Sesgo 1 capa oculta $b_1 = 2$
- 2 Capa oculta $d_2 = 2$
- Sesgo 2 capa oculta $b_2 = 2$
- Capa de salida $d_3 = 1$
- Sesgo capa de salida $b_3 = 1$



Red Neuronal Multicapa

Arquitectura

Cantidad neuronas por capa:

- Capa Entrada $d_0 = 3$
- 1 Capa oculta $d_1 = 2$
- Sesgo 1 capa oculta $b_1 = 2$
- 2 Capa oculta $d_2 = 2$
- Sesgo 2 capa oculta $b_2 = 2$
- Capa de salida $d_3 = 1$
- Sesgo capa de salida $b_3 = 1$

Cantidad de parámetros a estimar:

$$a_0 = d_0 \times d_1 + b_1 \times d_1$$

$$a_1 = d_1 \times d_2 + b_2 \times d_2$$

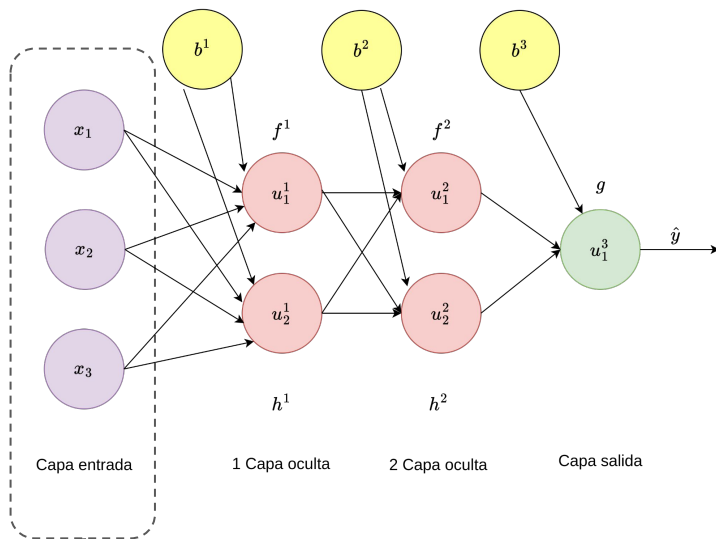
$$a_2 = d_2 \times 1 + b_3 \times 1$$

$$\text{Total} = a_0 + a_1 + a_2$$

$$= 17$$

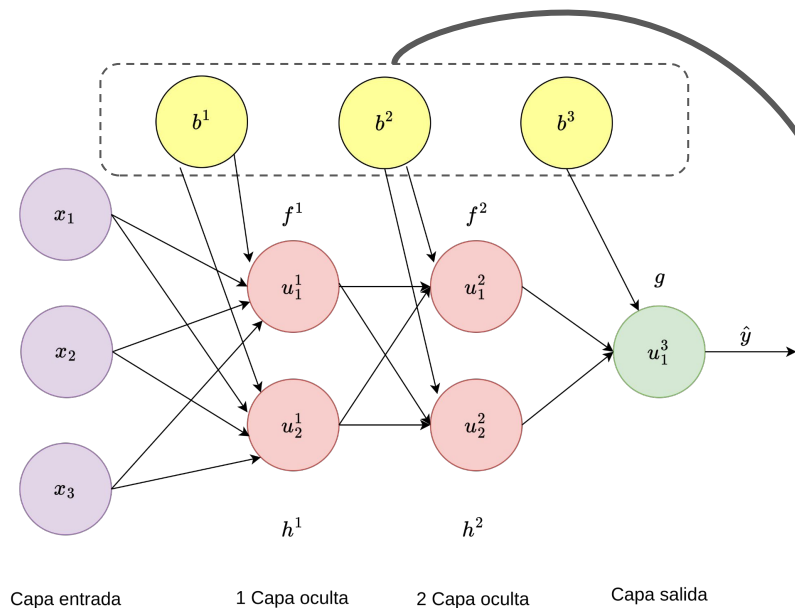
Red Neuronal Multicapa

Arquitectura



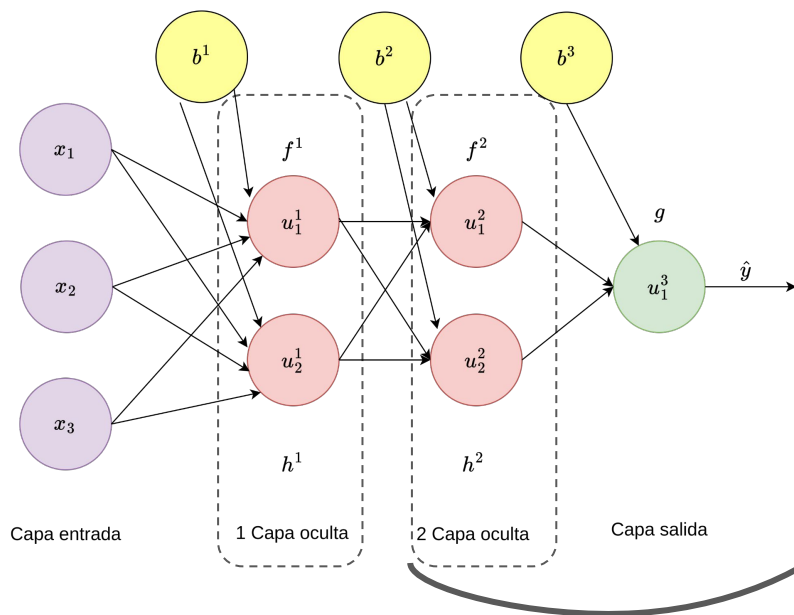
Capa de entrada: corresponde a los atributos de cada observación que pasaremos por la red.

Red Neuronal Multicapa



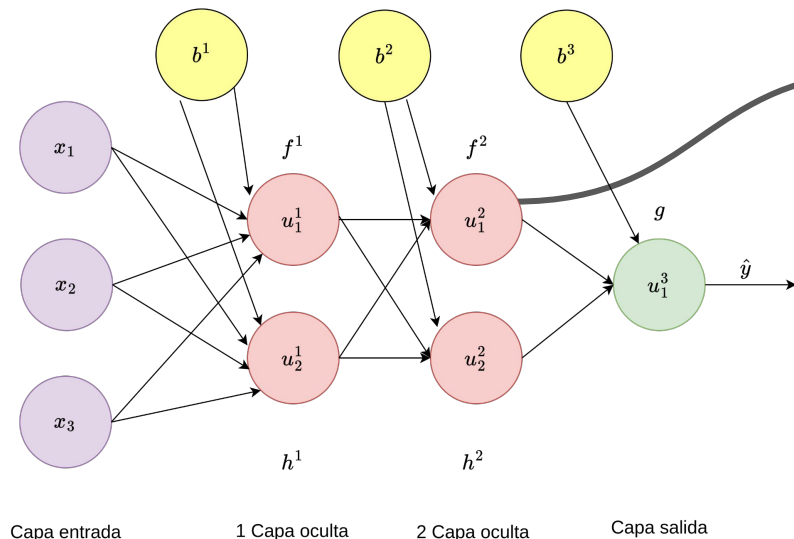
Bias: neurona que controla el sesgo en la suma ponderada de entradas y pesos en cada neurona de cada capa.

Red Neuronal Multicapa



Capas ocultas: estas capas pueden contener un número arbitrario de neuronas. Su objetivo es recibir conexiones de capas anteriores, procesar y generar salidas que serán entradas en la siguientes capa.

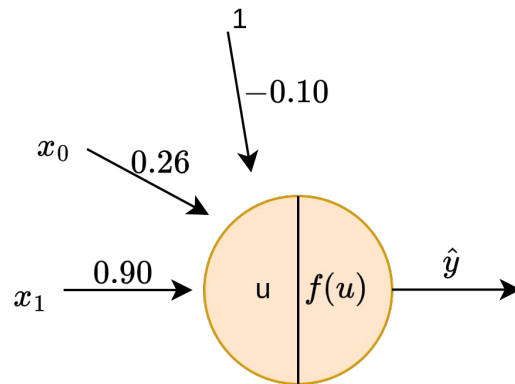
Red Neuronal Multicapa



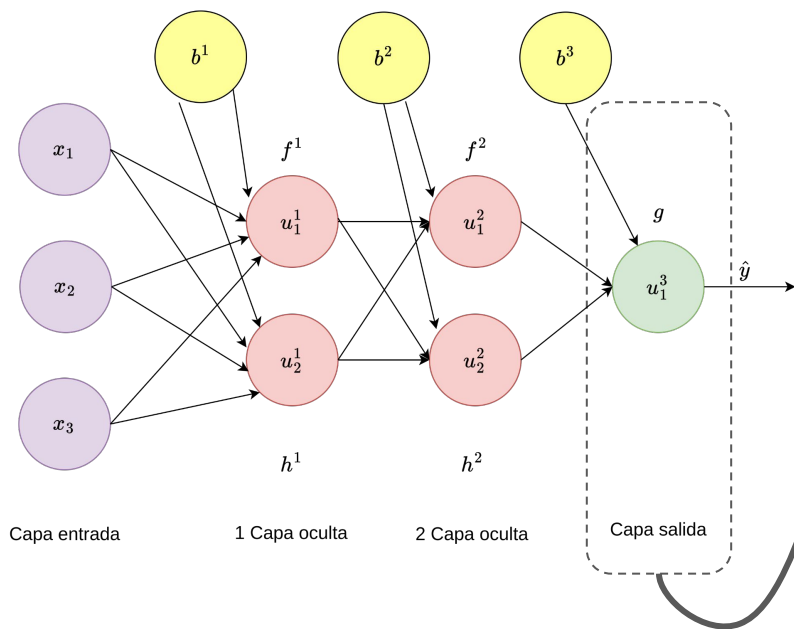
Perceptrón (Neurona): Calcula la suma ponderada:

$$u_1^{(2)} = h_1^{(1)} W_1^{(2)} + b_1^{(2)}$$

y la función de activación: $f^{(2)}(u_1^{(2)})$



Red Neuronal Multicapa



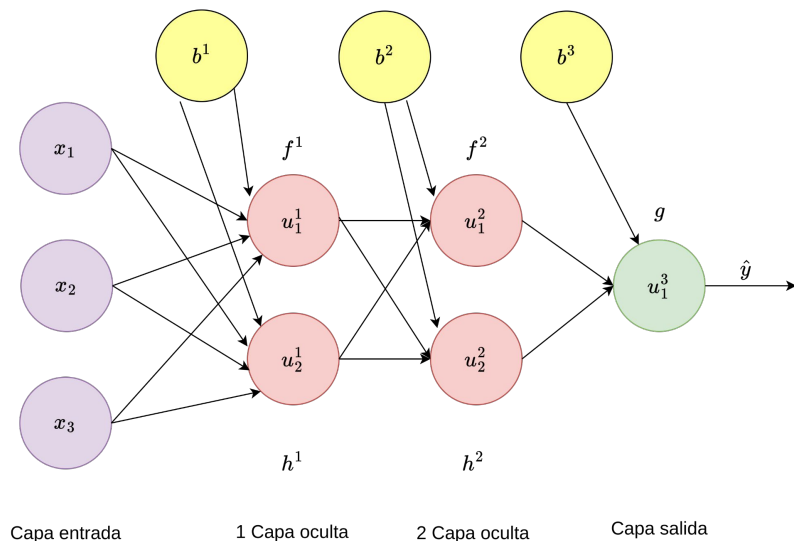
Capa Salida: última capa de la red, que modela el tipo de problema a resolver

- Para regresión o clasificación binaria se utiliza una neurona
- Para clasificación de más de dos clases se utilizan tantas neuronas como clases.

En esta capa aplicaremos una función de activación (g) que depende del problema:

- Para regresión usamos **activación lineal**
- Para clasificación binaria, **sigmoidal**
- Para más de dos clases, **softmax**.

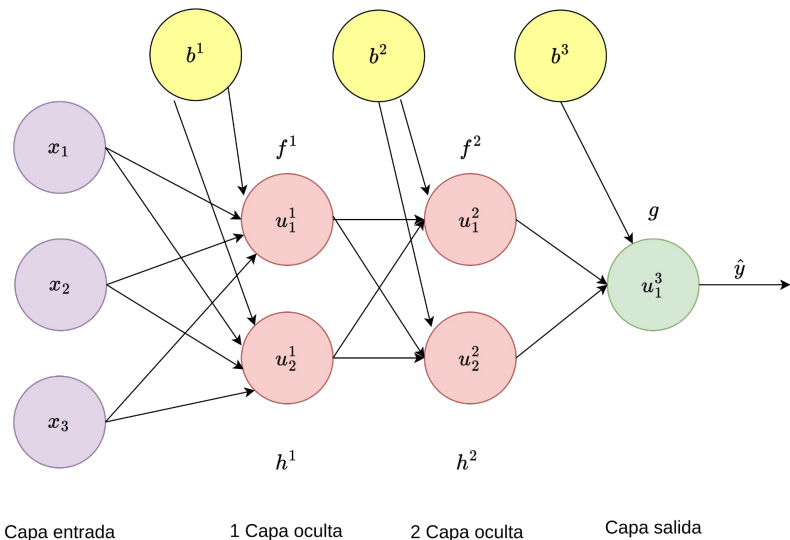
Red Neuronal Multicapa



Función de pérdida: Corresponde a la función que mide el ajuste actual de la red con respecto a la predicción esperada.

Para problemas de regresión usamos el **error cuadrático medio**. Para clasificación, **entropía cruzada**.

Red Neuronal Multicapa



Forward

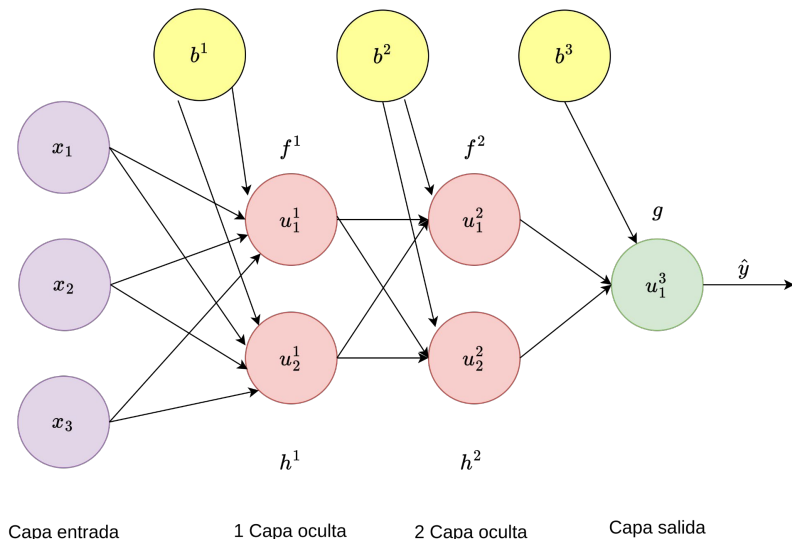
{desafío} Backward
latam_

Entrenamiento Red Neuronal Feed Forward

1. **Forward:** pasamos los ejemplos por la red desde la capa de entrada hasta la capa de salida, realizando todos los cálculos necesarios.
2. **Backward:** corresponde al **aprendizaje** y se realiza en dirección contraria a Forward, para ajustar los parámetros.

Se llama **época** a una iteración que se cuenta cuando hemos pasado todos los ejemplos del conjunto de entrenamiento por la red y realizado ajustes de parámetros.

Ejemplo FFNN



Forward

Funciones de activación:

$$f^{(1)} = f^{(2)} = \text{Relu}$$

$$g = \text{Linear}(x)$$

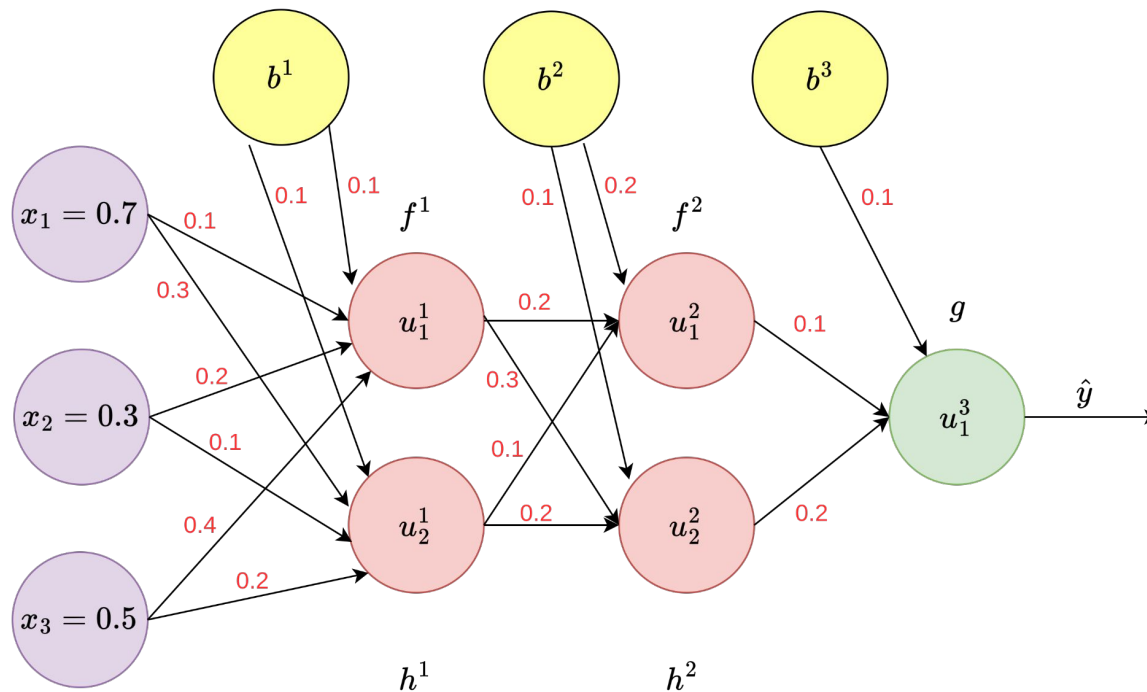
Función de pérdida:

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$



Forward

Ejemplo FFNN



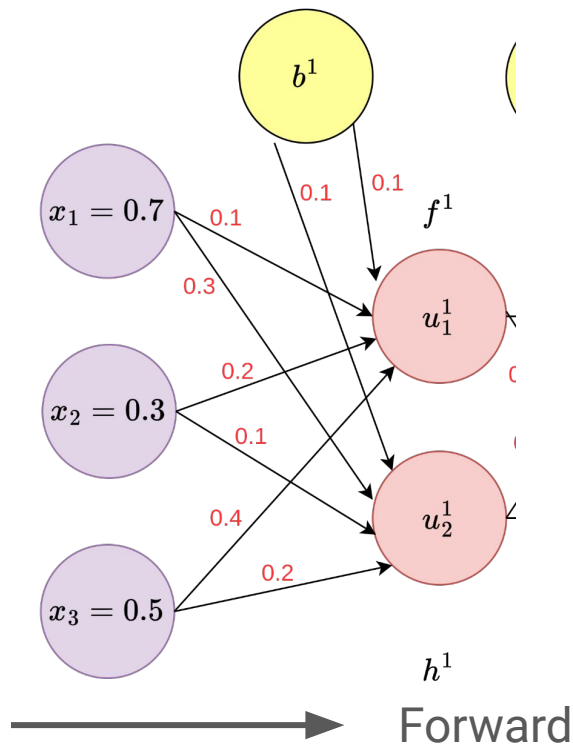
Capa entrada

1 Capa oculta

2 Capa oculta

Capa salida

Ejemplo FFNN



{desafío}
latam_

Forward

1 capa oculta

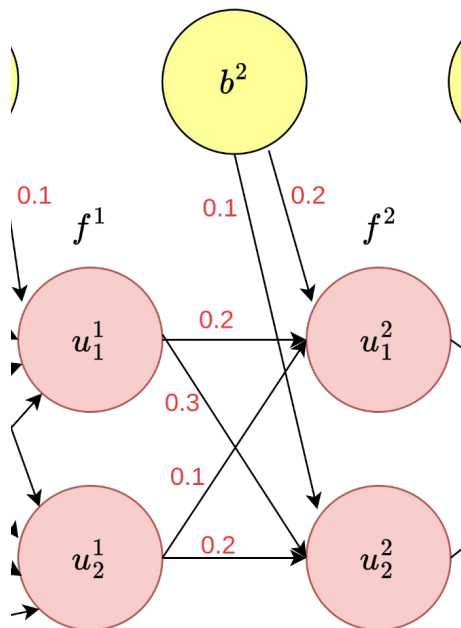
$$u_1^{(1)} = 0.7 \times 0.1 + 0.3 \times 0.2 + 0.5 \times 0.4 + 0.1$$
$$= 0.43$$

$$h_1^{(1)} = \text{Relu}(u_1^{(1)})$$
$$= 0.43$$

$$u_2^{(1)} = 0.7 \times 0.3 + 0.3 \times 0.1 + 0.5 \times 0.2 + 0.1$$
$$= 0.44$$

$$h_2^{(1)} = \text{Relu}(u_2^{(1)})$$
$$= 0.44$$

Ejemplo FFNN



Forward

{desafío}
latam_

Forward

2 capa oculta

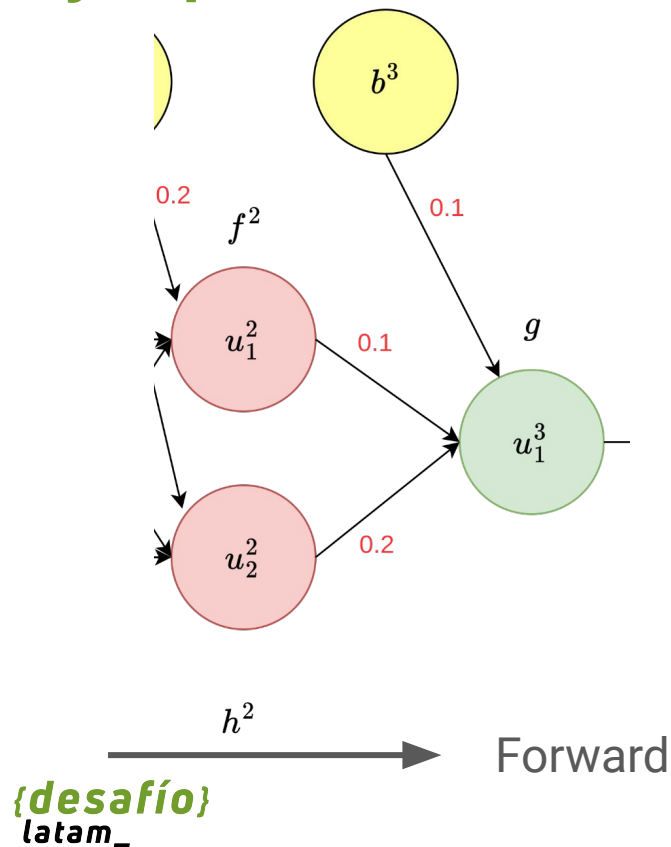
$$u_1^{(2)} = 0.43 \times 0.2 + 0.44 \times 0.1 + 0.2 \\ = 0.33$$

$$h_1^{(2)} = \text{Relu}(u_1^{(2)}) \\ = 0.33$$

$$u_2^{(2)} = 0.43 \times 0.3 + 0.44 \times 0.2 + 0.1 \\ = 1.217$$

$$h_2^{(2)} = \text{Relu}(u_2^{(2)}) \\ = 1.217$$

Ejemplo FFNN



Forward

Capa de salida

$$u_1^{(3)} = 0.33 \times 0.1 + 1.217 \times 0.2 + 0.1 \\ = 0.3764$$

$$g = \text{Linear}(u_1^{(3)}) \\ = 0.3764$$

/*Backward*/

Backward

- Corresponde a la etapa que realiza el aprendizaje de la red, por medio del ajuste de los parámetros.
- El ajuste de parámetros requiere medir el impacto que tiene cambiar en una pequeña cantidad el valor actual de un parámetro, en la función de pérdida de la red. Esto se hace calculando el gradiente de la función de pérdida con respecto a cada parámetro.
- La actualización de parámetro se realiza utilizando la fórmula:

$$\theta_{t+1} = \theta_t - \alpha \left(\frac{1}{n} \sum_{i=1}^n \nabla_{\theta} \mathcal{L}(x^{(i)}, y^{(i)}, \theta_t) \right)$$

Backward

- Los parámetros tienen diferentes niveles de dependencia entre ellos, de acuerdo con las capas que afectan a la función de pérdida.
- Para calcular el gradiente de una red se usa la **regla de la cadena**

$$\begin{aligned} f &= u(w) \\ w &= g(x) \end{aligned} \qquad \frac{\partial f}{\partial x} = \frac{\partial u}{\partial w} \frac{\partial w}{\partial x}$$

Ejemplo FFNN

Backward

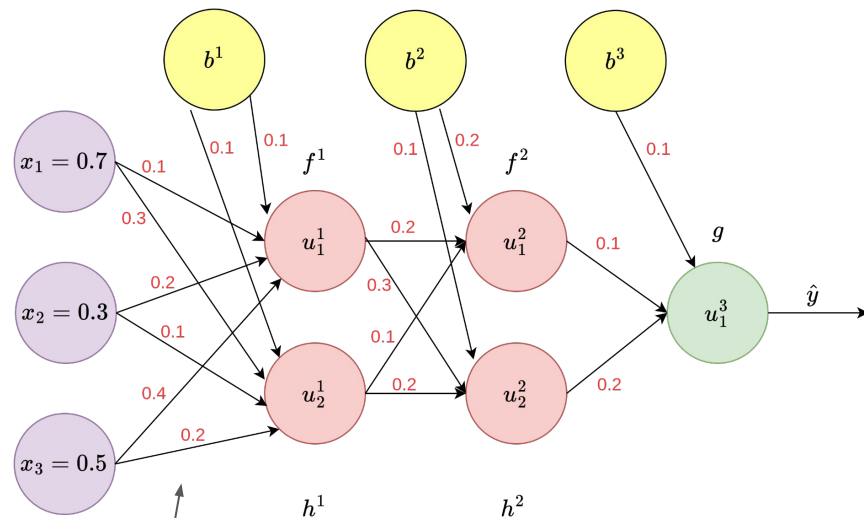
Proceso de actualización de pesos, que comienza una vez concluye la etapa forward.

- Para actualizar los pesos debemos calcular el gradiente de cada parámetro.
- Los gradientes se calculan usando la regla de la cadena
- Una vez que se calculan los gradientes, se actualizan sus valores usando la regla de aprendizaje (descenso del gradiente).

$$\hat{y} \rightarrow \epsilon_{total} = \mathcal{L}(y_i, \hat{y}) \quad \theta_{t+1} = \theta_t - \alpha \left(\frac{1}{n} \sum_{i=1}^n \nabla_{\theta} \mathcal{L}(x^{(i)}, y^{(i)}, \theta_t) \right)$$

Ejemplo FFNN

Backward



$$\frac{\partial \mathcal{L}}{\partial \mathcal{L}} = 1$$

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}} \frac{\partial \mathcal{L}}{\partial \hat{y}}$$

$$\frac{\partial \mathcal{L}}{\partial u^{(3)}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial u^{(3)}}$$

$$\frac{\partial \mathcal{L}}{\partial w^{(1)}} = \frac{\partial \mathcal{L}}{\partial u^{(1)}} \frac{\partial u^{(1)}}{\partial w^{(1)}}$$

1 Capa oculta

$$\frac{\partial \mathcal{L}}{\partial h^{(1)}} = \frac{\partial \mathcal{L}}{\partial u^{(2)}} \frac{\partial u^{(2)}}{\partial h^{(1)}}$$

$$\frac{\partial \mathcal{L}}{\partial u^{(1)}} = \frac{\partial \mathcal{L}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial u^{(1)}}$$

2 Capa oculta

Capa salida

$$\frac{\partial \mathcal{L}}{\partial h^{(2)}} = \frac{\partial \mathcal{L}}{\partial u^{(3)}} \frac{\partial u^{(3)}}{\partial h^{(2)}}$$

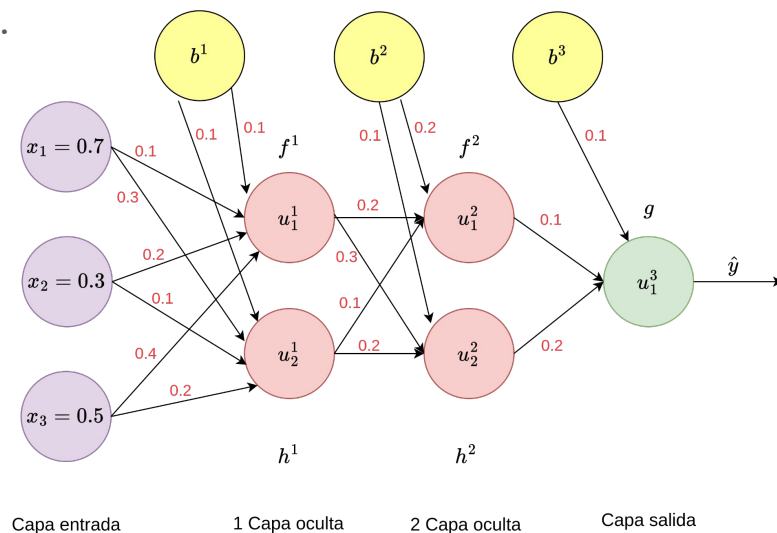
$$\frac{\partial \mathcal{L}}{\partial u^{(2)}} = \frac{\partial \mathcal{L}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial u^{(2)}}$$

/*Backpropagation*/

Ejemplo FFNN

Backpropagation

Es un algoritmo que se utiliza para calcular derivadas en forma eficiente, utilizando programación dinámica.



Demostración - Simulación de redes neuronales multicapa



Simulación red neuronal multicapa

Problema no linealmente separable



Revisaremos una simulación de red neuronal con distintas arquitecturas usando un simulador online. Para ello, visitaremos la página <https://playground.tensorflow.org/>





Simulación red neuronal multicapa

DATA

Which dataset do you want to use?

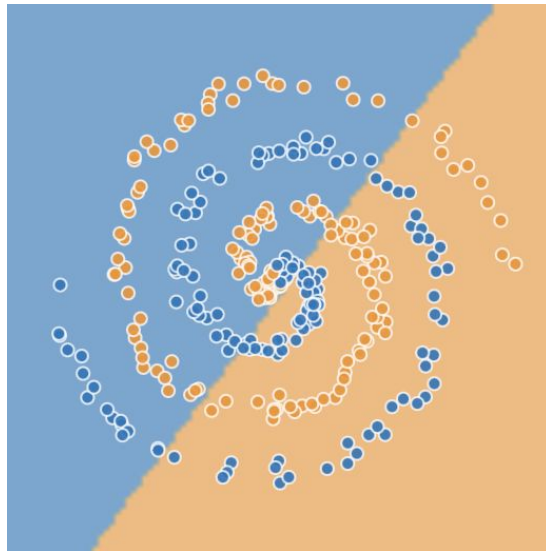
☐  ☐ 

☐  ☒ 

Ratio of training to test data: 60%

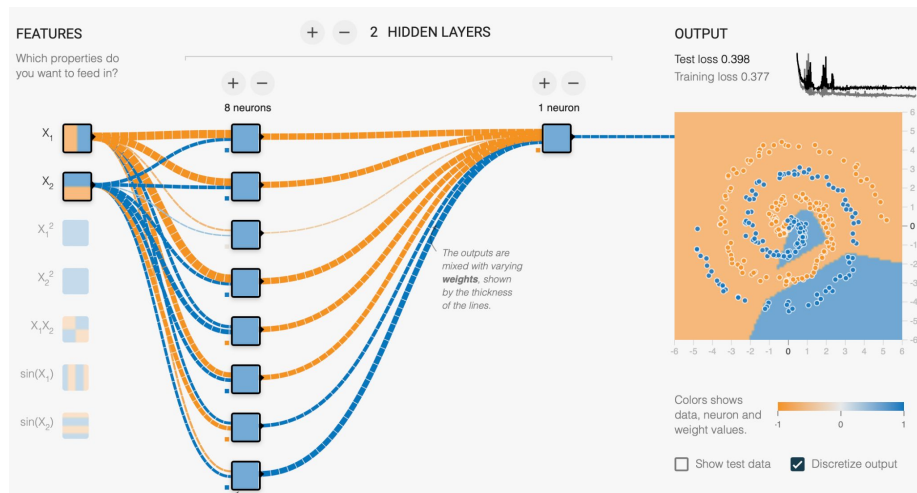
Noise: 25

Batch size: 10



Simulación red neuronal multicapa

Aproximar usando solo una capa oculta



Learning rate

0.1

Activation

ReLU

Regularization

None

Regularization rate

0

Problem type

Classification

Simulación red neuronal multicapa

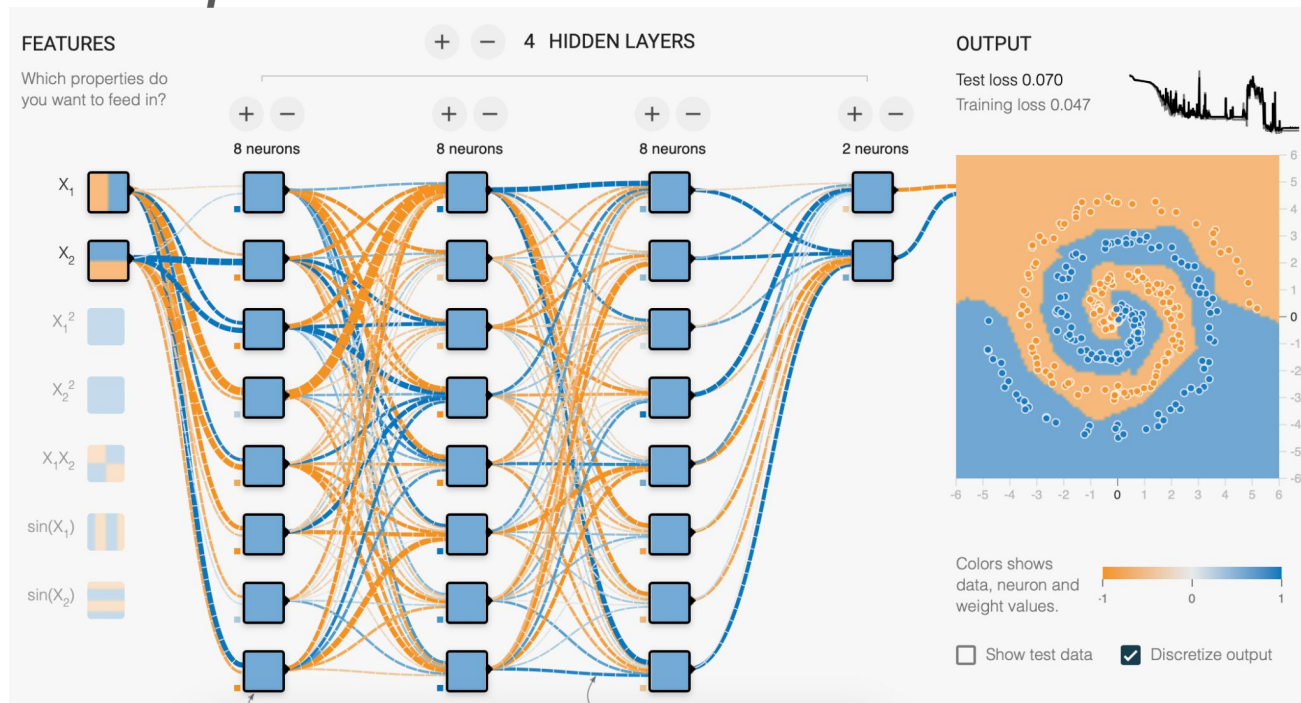
Aproximar usando solo una capa oculta

- No logramos mejorar una pérdida en el conjunto de test de 0.4
- La cantidad de neuronas se hace insuficiente debido a la complejidad de la separación de las clases.



Simulación red neuronal multicapa

Añadiendo más capas



/* Regularización para Redes Neuronales */

Regularización

¿Por qué es necesaria?

Las redes neuronales “profundas” son propensas a generar modelos con problemas de sobre entrenamiento. Para evitarlo mediante regularización contamos con:

- Regularización paramétrica
- DROPOUT

Regularización paramétrica

Normas

Norma L1 - Lasso

Esta norma es agresiva en cuanto a que permitirá apagar pesos, lo que significa un descarte o que dejará sin efecto al parámetro en la función de pérdida.

$$w^{(1)} + \frac{\lambda}{2m} \sum_w^m \|w\|$$

Norma L2 - Ridge

Esta norma penalizará los pesos, pero sin llegar a descartarlos

$$w^{(1)} + \frac{\lambda}{2m} \sum_w^m w^2$$

Regularización Dropout

Características

- **Aleatoriedad durante el entrenamiento:** aleatoriamente se “apagan” (poner un valor cero a los pesos de entrada de una neurona temporalmente) un porcentaje de neuronas durante cada paso del entrenamiento. Se evita así que las neuronas dependan en exceso unas de otras.
- **Reducción del sobreajuste:** al forzar a la red a aprender con neuronas que pueden ser desactivadas en cualquier momento, se evita que la red dependa demasiado de un conjunto particular de neuronas, evitando el sobreajuste.
- **Mejora del rendimiento y robustez:** el rendimiento mejora, ya que hace que el modelo sea más robusto y se potencie la capacidad de generalización.

Regularización Dropout

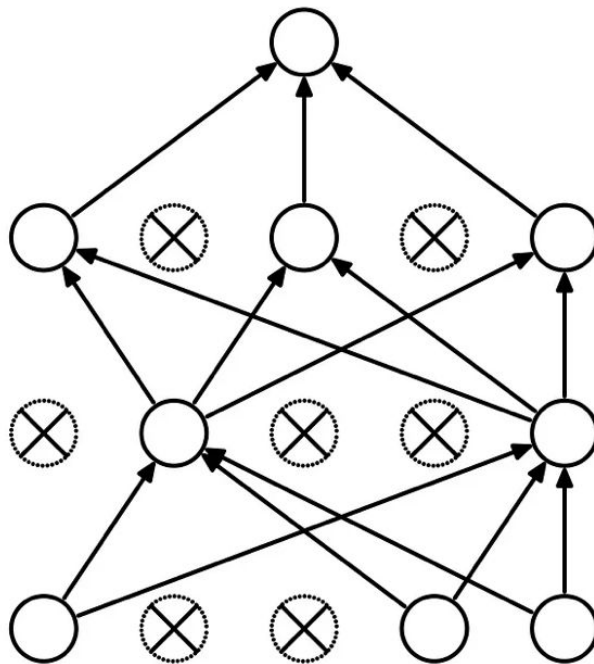
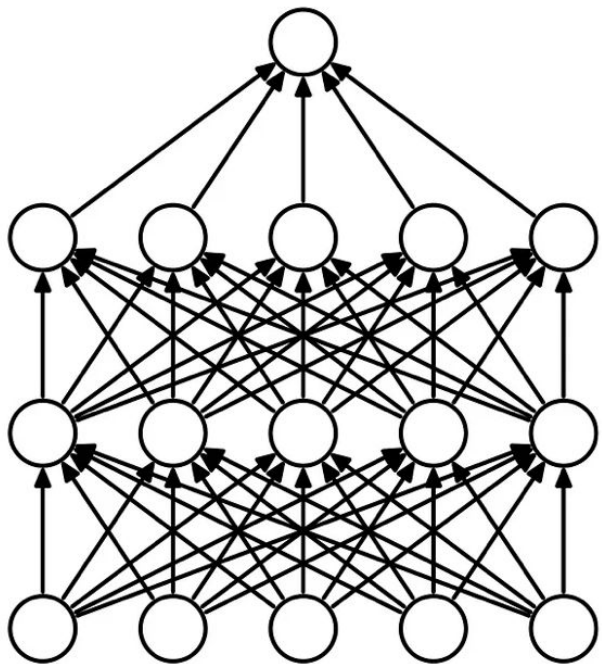
Descripción del proceso

En el entrenamiento se elige aleatoriamente un porcentaje de neuronas para “apagarlas” temporalmente.

1. Se realiza un paso Forward con neuronas que no fueron “apagadas”
2. Hacemos un paso Backward actualizando los parámetros asociados a las neuronas activas.
3. Se repiten estos pasos para cada época

Regularización Dropout

Proceso Dropout



¡Manos a la obra! - Identificación de dígitos



¡Manos a la obra!

Identificación de dígitos

Veremos ahora la forma de implementar esto en Python, para identificar dígitos escritos a mano en una base de imágenes. Para ello, abre el archivo indicado y sigue los pasos que te indicará tu profesor.



Desafío - Predicción de cancelación de reserva



Desafío

"Predicción de cancelación de reserva"

- Descarga el archivo "Desafío".
- Tiempo de desarrollo asincrónico: desde 2 horas.
- Tipo de desafío: individual.

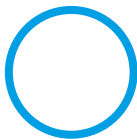
¡AHORA TE TOCA A TI! 💪



Ideas fuerza



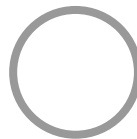
Las **redes neuronales feedforward multicapa** se entrenan con **propagación hacia adelante y hacia atrás**, para ajustar los pesos de la red.



La **inicialización adecuada de los pesos** y la elección de **funciones de pérdida y activación** son claves en el rendimiento del modelo.



La **regularización de los datos** evita el **sobreajuste**. Incluye la aplicación de **normas Lasso, Ridge y Dropout**.



Se debe **optimizar el modelo** por medio **de la búsqueda de hiperparámetros adecuados**

¿En qué áreas te parece que puede ser crucial el uso de lo que has aprendido en esta sesión?





Próxima sesión...

Abordaremos las problemáticas relacionadas con el procesamiento de imágenes utilizando redes neuronales convolucionales.

{desafío}
latam_

*Academia de
talentos digitales*

