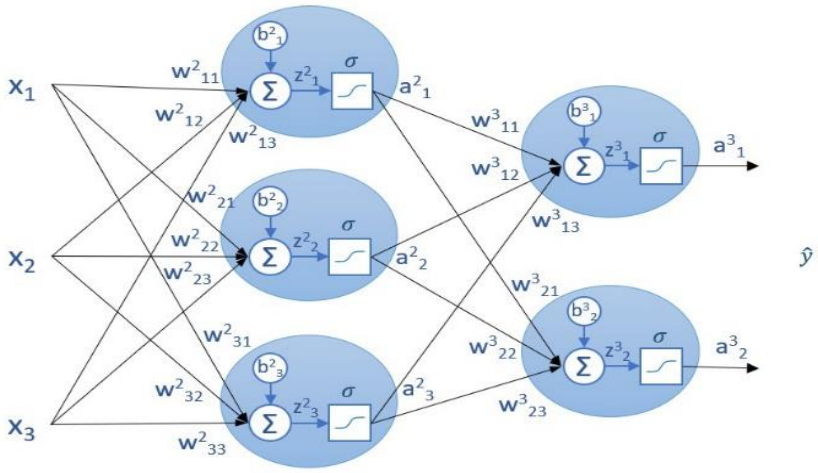
1. Redes neuronales:

Las redes neuronales son modelos de aprendizaje automático inspirados en el funcionamiento del cerebro humano.

Nosotros establecemos la estructura de la red neuronal:

- **Número de capas**: Determina la profundidad de la red.
- **Número de neuronas**: Cada capa tiene un conjunto de neuronas que procesan información.
- **Función de activación**: Introduce no linealidad, permitiendo que la red capture patrones complejos (ReLU, Sigmoid, Softmax…).
- **Número de épocas (*epochs*)**: Ciclos completos de entrenamiento donde se recorren todos los datos.
- **Tamaño del lote (*batch_size*)**: Cantidad de muestras usadas para calcular los gradientes en cada iteración.
- **Tasa de aprendizaje (*learning rate*)**: Determina el tamaño de los pasos en la optimización.
- **Algoritmo de optimización (*optimazer*)**: Ajusta los pesos para minimizar la función de pérdida (adam, RMSProp…).
- **Función de pérdida (*loss*)**: Mide el error entre las predicciones y los valores reales (BinaryCrossentropy para clasificación binaria, categorical_crossentropy para multiclase…).
- **Métrica**: Evalúa el rendimiento del modelo.



Entrenamiento general

1. Los datos pasan por todas las capas.
2. Se calcula la **función de pérdida**.
3. **Retropropagación**: Se ajustan los pesos con base en los gradientes (nos indican la dirección donde se minimiza la función de pérdida).

Antes de entrenar el modelo:

- **Normalizar los datos** para evitar gradientes inestables.
- **Codificar variables** (binomializar la variable objetivo en clasificación multiclase).
- **División de datos**: Train/Test/Validación.
- **OPCIONAL: Optimiza el entrenamiento (dataset de tensorflow),** por ejemplo:
  - *PREFETCH*: Carga y procesa los datos de manera simultánea.
  - *SHUFFLE*: Mezclar aleatoriamente los datos.
  - *BATCH*: Dividir los datos en lotes.
- **OPCIONAL: Data Augmentation:** Aumentar la variedad del dataset con transformaciones (principalmente en problemas de visión por computadora).

---

2. Tipos de Redes Neuronales:

Redes Neuronales Densas (Fully Connected)

- **Capa de entrada**: Establecemos la dimensión de los datos.
- **Capas densas:** Totalmente conectadas (Fully Connected).
- **Capa de salida**:
  - *Sigmoid*: Clasificación binaria.
  - *Softmax*: Clasificación multiclase.
  - *Linear*: Regresión.

Ejemplo: Predicción de precios de viviendas a partir de características numéricas.

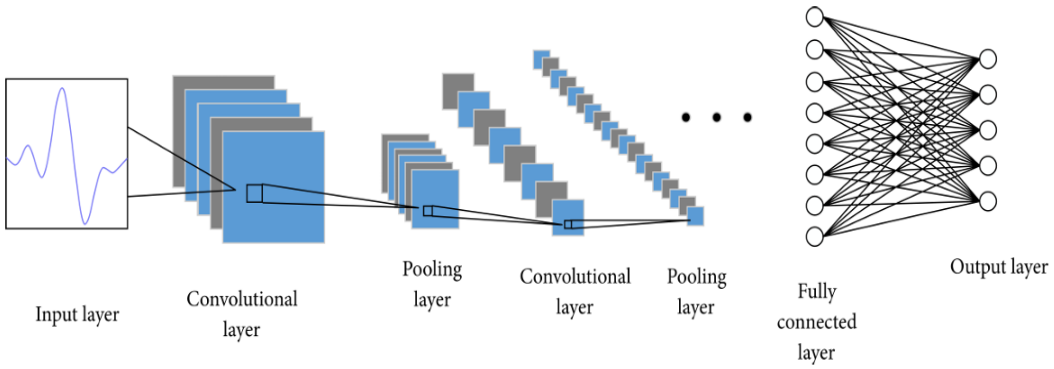
Redes Neuronales Convolucionales (CNN)

- **Capa de entrada**.
- **Capas convolucionales**: Extraen características locales (patrones).
- **Capas de agrupamiento**: Reducen la dimensionalidad, manteniendo la información relevante (*MaxPooling*…).

  (estas dos se van alternando)
- **Capa de aplanamiento (Flatten)**: Convierte la salida de las capas convolucionales en un vector. A veces, en lugar de usar *Flatten*, se utilizan otras capas de agrupamiento global, como *GlobalAveragePooling*.
- **Capas densas (Fully Connected)**: Combinan las características extraídas para realizar la predicción final.
- **Capa de salida**.

Son efectivas en la extracción de características locales y patrones dentro de una secuencia o imagen, realizando esta extracción de manera jerárquica.

Ejemplo: Clasificación de imágenes (gatos vs perros). Detección de neumonía en radiografías de tórax.



Redes Neuronales Recurrentes (RNN)

- **Capa de entrada**: Establece la dimensión de los datos secuenciales.
- **Capa recurrente:** Procesa la secuencia paso a paso, capturando dependencias temporales entre los pasos.
- **Capa densa (Fully Connected):** Combina las características extraídas por las capas recurrentes y realiza la predicción final.
- **Capa de salida.**

Útiles para **datos secuenciales**, donde la información de pasos anteriores influye en las predicciones actuales. Usadas por ejemplo para **series temporales** o **análisis de texto**.

Limitación: Capturan solo **dependencias secuenciales a corto plazo**.

Ejemplo: Predicción de un fallo en una máquina basándose en secuencias de eventos previos. Predicción de la siguiente palabra o carácter en una secuencia.

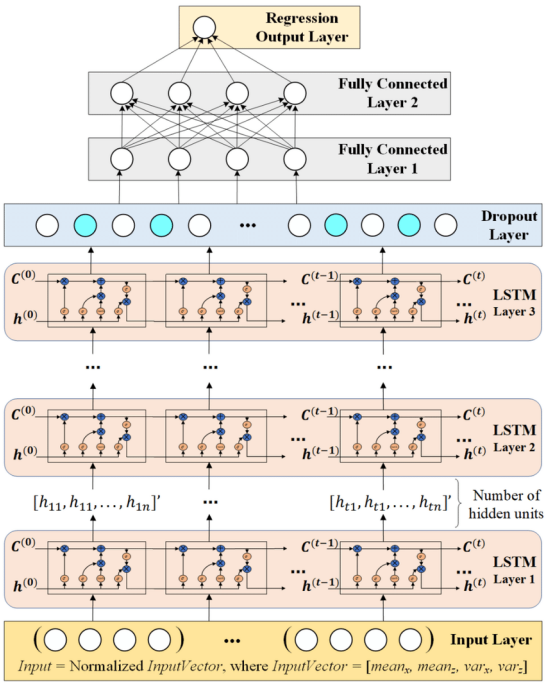
LSTM (Long Short-Term Memory)

- **Capa de entrada**: Establece la dimensión de los datos secuenciales.
- **Capa LSTM:** Variante mejorada de las RNN, con celdas de memoria que controlan el flujo de información y **capturan dependencias a corto y largo plazo**.
- **Capa densa (Fully Connected):** **Combina las características extraídas** para realizar la predicción final.
- **Capa de salida.**

Si se usa return_sequences=True en la capa LSTM, la salida será una secuencia completa. En este caso, se utiliza la capa **Flatten antes de la capa densa para convertir esa secuencia en un vector plano.*

Ejemplo:

1. **Traducción automática de textos:** En este caso, el modelo necesita procesar toda la secuencia de palabras de entrada para generar la traducción palabra por palabra.
2. **Predicción de valores futuros en series temporales, como temperaturas o precios de acciones:** Solo se necesita la última salida de la secuencia para predecir el valor futuro.



3. Procesamiento de Lenguaje Natural (NLP):

Paquetes comunes: NLTK, STANZA.

Pasos clave en el PREPROCESAMIENTO (¡¡depende del texto!!):

- **Tokenización**: Dividir el texto en unidades básicas de análisis llamadas *tokens*.
- **Conversión a minúsculas**.
- **Eliminación de stopwords**: palabras comunes en un idioma que no aportan mucho significado o información relevante.
- **Eliminación de puntuación y caracteres especiales**.
- **Lematización/Stemming**: Reducir palabras a su forma base.

Opciones avanzadas para mejorar el análisis:

- Identificación de entidades (NER).
- Etiquetado de partes del discurso (POS).
- Análisis de sentimientos.
- Modelado de temas (LDA).
- Sinónimos.

Modelos NLP:

- **Modelos simples/pocos datos:** TF-IDF.

  Técnica estadística que mide la importancia de cada token en un corpus, sin capturar relaciones semánticas.

- **Modelos complejos/textos largos:** Tokenizer + Embedding.

  El *tokenizer* divide el texto en tokens, y la *capa embedding* convierte cada token en un vector denso que captura significado y contexto.

---

4. Posibles pasos para mejorar un modelo inicial:

- Comienza con un modelo sencillo y ve añadiendo más capas y neuronas.
- Grafica la **función de pérdida** en entrenamiento y validación. Si oscila mucho, prueba a disminuir la tasa de aprendizaje.

- Ajusta el número de épocas y el tamaño de los lotes (se suelen usar potencias de dos).
- Introduce **callbacks.** Por ejemplo:
    - *EarlyStopping*: Detiene el entrenamiento si no mejora la (métrica) en el conjunto de validación después de un número de épocas.
    - *ReduceLROnPlateau*: Reduce la tasa de aprendizaje si no mejora la (métrica) en el conjunto de validación después de un número de épocas.
- Añadir **BatchNormalization** para estabilizar las activaciones durante el entrenamiento (útil en redes densas y convolucionales, menos recomendable en secuencias temporales; *LayerNormalization*).
- Para evitar **OVERFITTING**:
    - Agregar **dropout**: desactiva un porcentaje de las neuronas de la red de manera aleatoria durante el entrenamiento.
    - Aplicar **regularización** (L1, L2).
- Uso de modelos **preentrenados** (Transfer Learning).

---

5. Evaluación del modelo final:

- Evalúa el modelo con el conjunto de test.
- Métricas a utilizar:
    - **Clasificación**: accuracy, precision, recall, F1-score.
    - **Regresión**: MSE, RMSE, R2 (Recomendación: grafica o saca estadísticas de tu variable objetivo que te ayuden a interpretar los resultados).
- Grafica tus resultados.