Arquitecturas clásicas de capas

Aquí tienes **arquitecturas completas** clásicas y modernas para distintos tipos de problemas en Machine Learning con redes neuronales. Te incluyo para cada caso: el problema, el tipo de datos, la arquitectura típica y un ejemplo de código (Keras/TensorFlow, por ser el más popular y didáctico).

---

1. Clasificación de Imágenes (Visión por Computador)

Problema: Clasificar imágenes en categorías (ej. MNIST, CIFAR-10, detección de perros/gatos).

Tipo de datos: Imágenes.

Arquitectura típica:

- Entrada: Imagen (ej. 32×32×3)
- [Conv2D + ReLU] → [MaxPooling2D]
- [Conv2D + ReLU] → [MaxPooling2D]
- [Flatten]
- [Dense + ReLU]
- [Dropout]
- [Dense + Softmax] (salida)

Ejemplo:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dens
e, Dropout

model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(32,32,3)),
    MaxPooling2D(2,2),
    Conv2D(64, (3,3), activation='relu'),
```

```
    MaxPooling2D(2,2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])
```

---

2. Procesamiento de Texto (NLP, Sentiment Analysis)

Problema: Clasificar sentimientos en texto.

Tipo de datos: Secuencias de palabras.

Arquitectura típica:

- Entrada: Secuencia de índices de palabras
- [Embedding]
- [LSTM o GRU] (puede ser bidireccional)
- [Dense + ReLU]
- [Dropout]
- [Dense + Softmax/Sigmoid] (salida)

Ejemplo:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

model = Sequential([
    Embedding(input_dim=10000, output_dim=128, input_length=200),
    LSTM(64),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')  # Para binaria. Usa 'softmax' y más unida
des para multiclase
])
```

---

3. Series Temporales (Predicción de valores futuros)

Problema: Predecir valores futuros en una serie temporal (ej. precio de acciones).

Tipo de datos: Series temporales (secuencias de números).

Arquitectura típica:

- Entrada: Secuencia de valores
- [LSTM o GRU]
- [Dense]
- [Dense (salida)] (a veces solo una neurona para regresión)

Ejemplo:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

model = Sequential([
    LSTM(64, input_shape=(30, 1)),  # 30 pasos, 1 variable por paso
    Dense(32, activation='relu'),
    Dense(1)  # Salida de regresión
])
```

---

4. Clasificación de Datos Tabulares

Problema: Predecir una categoría o valor a partir de variables numéricas/categóricas.

Tipo de datos: Tablas (csv, excel).

Arquitectura típica:

- Entrada: Vector de características
- [Dense + ReLU] x n
- [BatchNormalization/Dropout] (opcional)
- [Dense + Softmax/Sigmoid] (salida)

Ejemplo:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization, Dropout

model = Sequential([
    Dense(64, activation='relu', input_shape=(20,)),
    BatchNormalization(),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

---

5. Traducción Automática (NLP, Encoder-Decoder)

Problema: Traducir de un idioma a otro (machine translation).

Tipo de datos: Secuencias de palabras.

Arquitectura típica:

- **Encoder:** [Embedding] → [LSTM/GRU]
- **Decoder:** [Embedding] → [LSTM/GRU] → [Dense + Softmax] (por paso)
- **Opcional:** Capas de atención (Attention)

Ejemplo (simplificado):

```
from tensorflow.keras.layers import Input, Embedding, LSTM, Dense
from tensorflow.keras.models import Model

encoder_inputs = Input(shape=(None,))
x = Embedding(10000, 256)(encoder_inputs)
encoder_outputs, state_h, state_c = LSTM(256, return_state=True)(x)

decoder_inputs = Input(shape=(None,))
x = Embedding(10000, 256)(decoder_inputs)
decoder_lstm = LSTM(256, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(x, initial_state=[state_h, state_c])
decoder_dense = Dense(10000, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)
```

```
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

En producción, se suele usar Attention y arquitecturas más sofisticadas (Transformers).

---

6. Modelos tipo Transformer (BERT, GPT, etc.)

Problema: Traducción, clasificación, generación de texto, etc.

Tipo de datos: Texto.

Arquitectura típica:

- [Embedding + Positional Encoding]
- [Stack de capas de Atención Multi-cabeza + Feedforward]
- [Dense] (salida)

Ejemplo:

No es trivial programar a mano un transformer completo, pero con Keras puedes usar:

```
from tensorflow.keras.layers import MultiHeadAttention
# Implementar transformer requiere construir varias subcapas,
# pero puedes usar modelos pre-entrenados de HuggingFace fácilmente.
```