Red neuronal con input

experimento optimizadores: Adam,RMSprop,SGD,Adagrad

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from imblearn.over_sampling import SMOTE
from sklearn.metrics import classification_report, confusion_matrix, accuracy_sc
import re
import matplotlib.pyplot as plt
import seaborn as sns

from tensorflow.keras import Input
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam, RMSprop, SGD, Adagrad
```

In [2]:
```
# Cargar dataset
df = pd.read_excel("datasetv2.xlsx")
```

In [3]:
```
def limpiar_texto(texto):
    # Minúsculas
    texto = texto.lower()
    # Eliminar caracteres especiales pero mantener números y letras
    texto = re.sub(r"[^a-z0-9áéíóúñ ]", " ", texto)
    # Eliminar espacios múltiples
    texto = re.sub(r"\s+", " ", texto).strip()
    return texto
```

In [4]:
```
df["message"] = df["message"].astype(str).apply(limpiar_texto)
```

In [5]:
```
# Vectorización de texto con TF-IDF
vectorizer = TfidfVectorizer(max_features=5000)  # limita vocabulario
X = vectorizer.fit_transform(df["message"]).toarray().astype("float32")
#y = df["target"].values
encoder = LabelEncoder()
y = encoder.fit_transform(df["target"])  # spam/legit → 0/1
```

In [6]:
```
# Split train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [7]:
```
# Balanceo con SMOTE
sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)
```

In [8]:
```
X_train_res = np.asarray(X_train_res).astype("float32")
X_test = np.asarray(X_test).astype("float32")
y_train_res = np.asarray(y_train_res).astype("int32")
y_test = np.asarray(y_test).astype("int32")
```

In [9]:

```python
# ==========================
# 2. Función para crear modelo
# ==========================
def create_model(optimizer):
    model = Sequential([
        Input(shape=(X_train_res.shape[1],)),
        Dense(128, activation='relu'),
        Dropout(0.3),
        Dense(64, activation='relu'),
        Dropout(0.3),
        Dense(1, activation='sigmoid')
    ])
    model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['acc
    return model
```

In [10]:

```python
# ==========================
# Lista de optimizadores
# ==========================
optimizers = {
    "Adam": Adam(learning_rate=0.001),
    "RMSprop": RMSprop(learning_rate=0.001),
    "SGD": SGD(learning_rate=0.01, momentum=0.9),
    "Adagrad": Adagrad(learning_rate=0.01)
}
```

In [11]:

```python
histories = {}
results = {}

for name, opt in optimizers.items():
    print(f"\n 🔹 Entrenando con {name}...")
    model = create_model(opt)
    history = model.fit(
        X_train_res, y_train_res,
        epochs=10, batch_size=32,
        validation_data=(X_test, y_test),
        verbose=0
    )

    histories[name] = history  # guardamos historial

    # Evaluación
    y_pred = (model.predict(X_test) > 0.5).astype("int32")
    report = classification_report(y_test, y_pred, target_names=["legit", "spam"
    results[name] = report
    print(classification_report(y_test, y_pred, target_names=["legit", "spam"]))
```

```
🔹 Entrenando con Adam...
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
              precision    recall  f1-score   support

       legit       0.97      0.95      0.96      1583
        spam       0.90      0.93      0.92       719

    accuracy                           0.95      2302
   macro avg       0.94      0.94      0.94      2302
weighted avg       0.95      0.95      0.95      2302


🔹 Entrenando con RMSprop...
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
              precision    recall  f1-score   support

       legit       0.97      0.95      0.96      1583
        spam       0.90      0.94      0.92       719

    accuracy                           0.95      2302
   macro avg       0.94      0.95      0.94      2302
weighted avg       0.95      0.95      0.95      2302


🔹 Entrenando con SGD...
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
              precision    recall  f1-score   support

       legit       0.97      0.97      0.97      1583
        spam       0.93      0.93      0.93       719

    accuracy                           0.96      2302
   macro avg       0.95      0.95      0.95      2302
weighted avg       0.96      0.96      0.96      2302


🔹 Entrenando con Adagrad...
72/72 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
              precision    recall  f1-score   support

       legit       0.97      0.96      0.97      1583
        spam       0.91      0.94      0.93       719

    accuracy                           0.95      2302
   macro avg       0.94      0.95      0.95      2302
weighted avg       0.95      0.95      0.95      2302
```

In [12]:

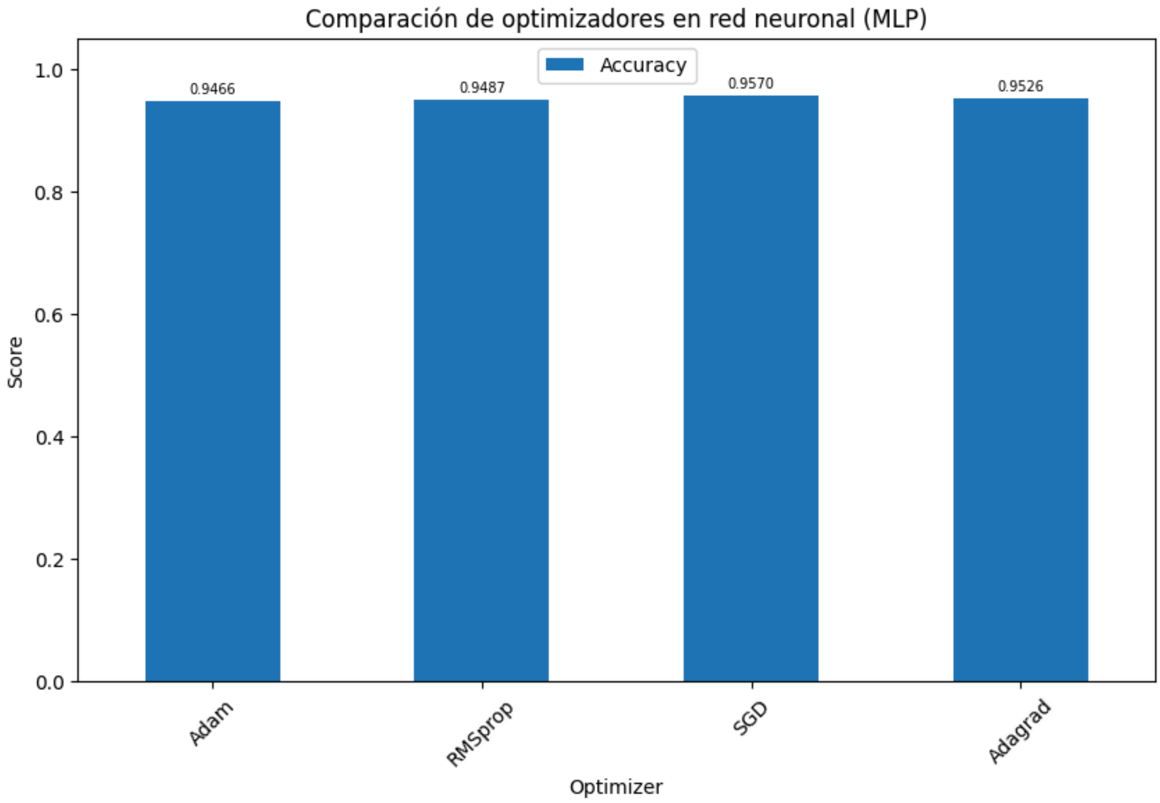
```
# ==========================
# 5. Comparación
# ==========================
print("\n📊 Resumen de Accuracy por optimizador:")
for name, rep in results.items():
    print(f"{name}: {rep['accuracy']:.4f}")
```

```
📊 Resumen de Accuracy por optimizador:
Adam: 0.9466
RMSprop: 0.9487
SGD: 0.9570
Adagrad: 0.9526
```

In [13]:
```
# ==========================
# Convertir resultados a DataFrame
# ==========================
metrics = ["accuracy", "precision", "recall", "f1-score"]

# Extraemos los promedios "weighted avg" para comparar mejor
data = []
for name, rep in results.items():
    data.append({
        "Optimizer": name,
        "Accuracy": rep["accuracy"],
        "Precision": rep["weighted avg"]["precision"],
        "Recall": rep["weighted avg"]["recall"],
        "F1-score": rep["weighted avg"]["f1-score"]
    })

results_df = pd.DataFrame(data)
print(results_df)
```

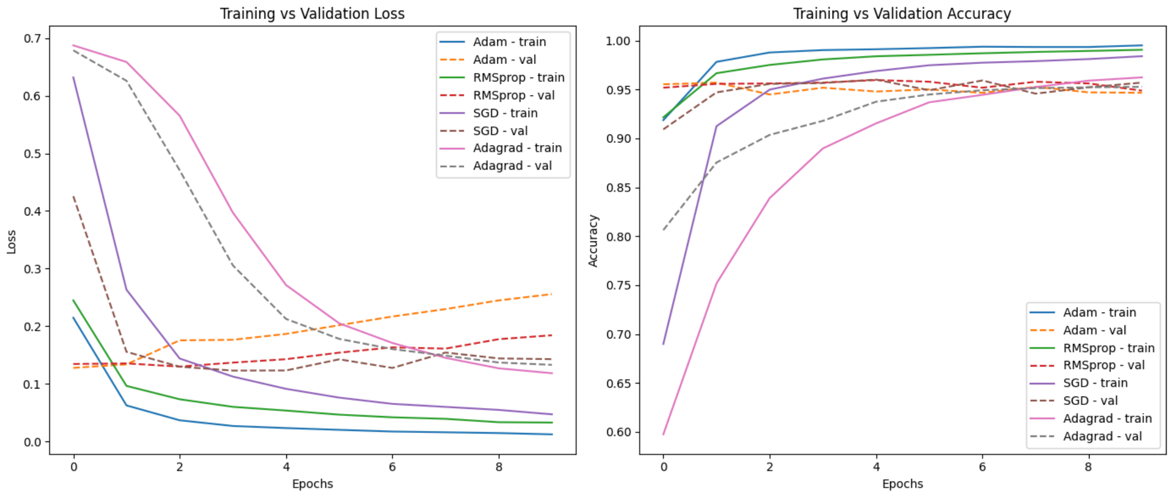
```
  Optimizer  Accuracy  Precision    Recall  F1-score
0      Adam  0.946568   0.947158  0.946568  0.946774
1   RMSprop  0.948740   0.949659  0.948740  0.949019
2       SGD  0.956994   0.957011  0.956994  0.957002
3   Adagrad  0.952650   0.953065  0.952650  0.952799
```

In [14]:
```
# ==========================
# Gráfico comparativo
# ==========================
results_df.set_index("Optimizer")[["Accuracy"]].plot(
    kind="bar", figsize=(10,6)
)
plt.title("Comparación de optimizadores en red neuronal (MLP)")
plt.ylabel("Score")
plt.ylim(0, 1.05)
plt.xticks(rotation=45)

# Anotar valores arriba de cada barra
ax = plt.gca()
for p in ax.patches:
    ax.annotate(f"{p.get_height():.4f}",
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='bottom', fontsize=7, color="black", xytext=(0,3
                textcoords="offset points")

plt.show()
```



In [15]:

```
# ==========================
# Curvas de entrenamiento
# ==========================
plt.figure(figsize=(14,6))

# ---- Training & Validation Loss ----
plt.subplot(1,2,1)
for name, history in histories.items():
    plt.plot(history.history["loss"], label=f"{name} - train")
    plt.plot(history.history["val_loss"], linestyle="--", label=f"{name} - val")
plt.title("Training vs Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()

# ---- Training & Validation Accuracy ----
plt.subplot(1,2,2)
for name, history in histories.items():
    plt.plot(history.history["accuracy"], label=f"{name} - train")
    plt.plot(history.history["val_accuracy"], linestyle="--", label=f"{name} - v
plt.title("Training vs Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()

plt.tight_layout()
plt.show()
```



In [16]:
```
# ==========================
# Crear DataFrame de resultados
# ==========================
data = []
for name, rep in results.items():
    data.append({
        "Optimizer": name,
        "Accuracy": rep["accuracy"],
        "Precision": rep["weighted avg"]["precision"],
        "Recall": rep["weighted avg"]["recall"],
        "F1-score": rep["weighted avg"]["f1-score"]
    })

results_df = pd.DataFrame(data)

# ==========================
# Mostrar tabla ordenada
# ==========================
print("\n📊 Resultados comparativos:")
print(results_df.sort_values(by="F1-score", ascending=False))

# ==========================
# Mejor optimizador (según F1-score)
# ==========================
best_f1 = results_df.loc[results_df["F1-score"].idxmax()]
best_acc = results_df.loc[results_df["Accuracy"].idxmax()]

print("\n🏆 Mejor optimizador por F1-score:")
print(best_f1)

print("\n🏆 Mejor optimizador por Accuracy:")
print(best_acc)
```

```
📊 Resultados comparativos:
  Optimizer  Accuracy  Precision    Recall  F1-score
2       SGD  0.956994   0.957011  0.956994  0.957002
3   Adagrad  0.952650   0.953065  0.952650  0.952799
1   RMSprop  0.948740   0.949659  0.948740  0.949019
0      Adam  0.946568   0.947158  0.946568  0.946774

🏆 Mejor optimizador por F1-score:
Optimizer         SGD
Accuracy     0.956994
Precision    0.957011
Recall       0.956994
F1-score     0.957002
Name: 2, dtype: object

🏆 Mejor optimizador por Accuracy:
Optimizer         SGD
Accuracy     0.956994
Precision    0.957011
Recall       0.956994
F1-score     0.957002
Name: 2, dtype: object
```

In [ ]: