Relatório de Treinamento e Predição de Rede Neural

Autor: Pedro Henrique Vilaça Valverde

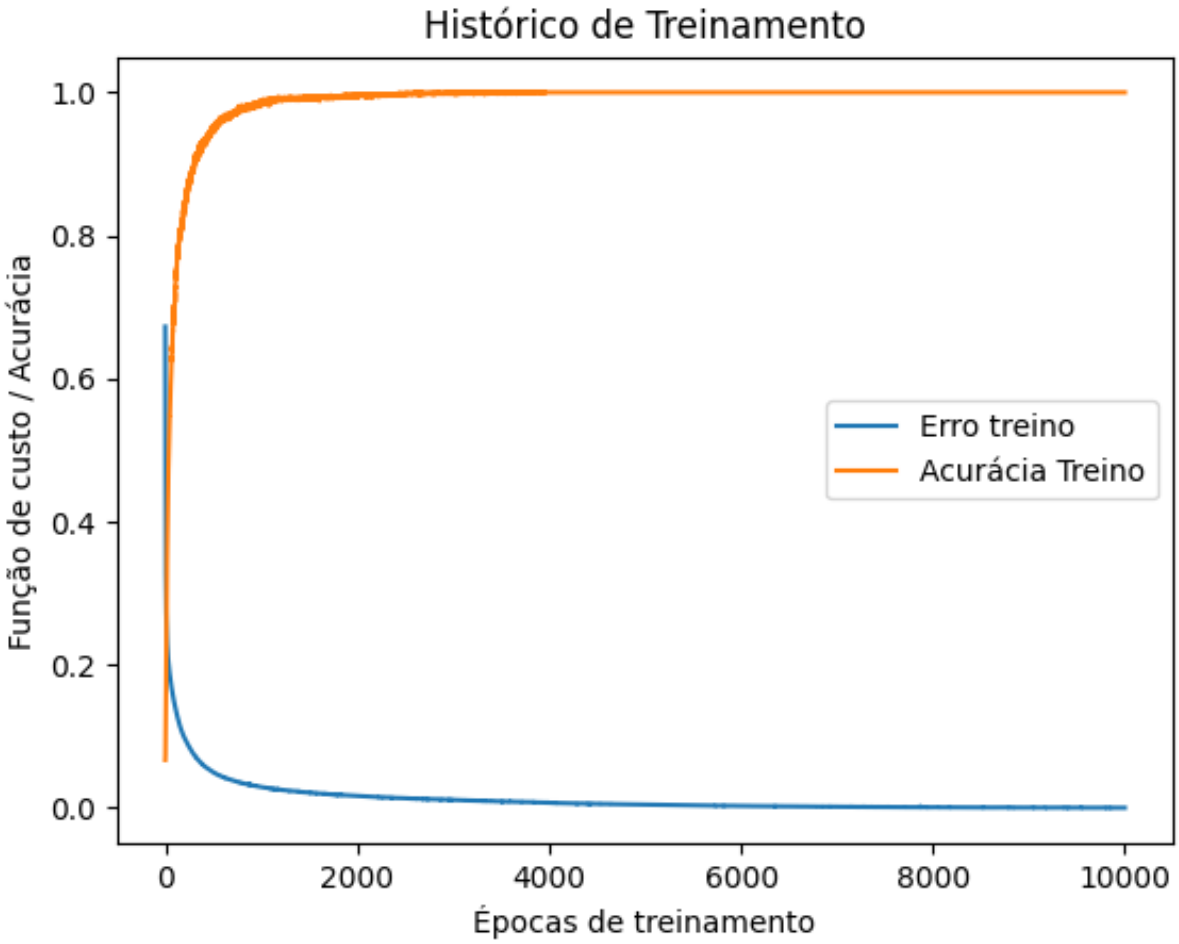
1. Resumo do Modelo

```
Model: "sequential"

 Layer (type)                        Output Shape                      Param #

| dense (Dense)                     | (None, 32)                |             384 |
+-----------------------------------------------------------------------------+
| dense_1 (Dense)                   | (None, 12)                |             396 |
+-----------------------------------+---------------------------+-----------------+
 Total params: 2,342 (9.15 KB)
 Trainable params: 780 (3.05 KB)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 1,562 (6.11 KB)
```

2. Desempenho do Treinamento

Acurácia final alcançada: 100.00%

3. Gráfico do Histórico de Treinamento


Histórico de Treinamento
Função de custo / Acurácia
Épocas de treinamento
Erro treino
Acurácia Treino
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000

4. Resultados da Predição

Legenda das Classes:

```
Index(['Esconder', 'Atirar', 'Parar', 'Andar', 'Morrer', 'Correr', 'Pular',
       'Sentar', 'Deitar', 'Fugir', 'Lutar', 'Render'],
      dtype='object')
```

Previsões Finais:

	Previsão de Classe
0	Morrer
1	Morrer
2	Deitar
3	Sentar
4	Render
5	Render
6	Andar
7	Andar
8	Esconder
9	Esconder
10	Lutar
11	Atirar
12	Lutar
13	Correr
14	Esconder
15	Andar
16	Andar
17	Parar
18	Sentar
19	Sentar
20	Deitar
21	Deitar
22	Morrer

5. Código Fonte (treinamento.py)

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
import warnings
import io
from contextlib import redirect_stdout

warnings.filterwarnings('ignore')

# --- DADOS DE TREINAMENTO ---

dados = pd.read_csv('./content/Treinamento_NPC.txt', header=None)

QUANTIDADE_COLUNAS = dados.shape[1]

print(f"Quantidade de colunas no dataset: {QUANTIDADE_COLUNAS}")

# Separa as features (X, primeiras QUANTIDADE_COLUNAS colunas) e o alvo (Y, última coluna)
X_df = dados.iloc[:, 0:QUANTIDADE_COLUNAS-1]
Y_series = dados.iloc[:, QUANTIDADE_COLUNAS-1]

# Converter colunas de texto em números
X_processado = X_df.copy()
mapeamento_features = {}

for col_idx in [1, 2, 3]:
    codigos, legenda = pd.factorize(X_df[col_idx])
    X_processado[col_idx] = codigos
    mapeamento_features[col_idx] = {texto: codigo for codigo, texto in enumerate(legenda)}



# Converte a coluna alvo (Y) de texto para números
Y_numerico, legenda_classes = pd.factorize(Y_series)
num_classes = len(legenda_classes)

# Normaliza os dados
scaler = StandardScaler()
X_final = scaler.fit_transform(X_processado)

# Transforma os dados
Y_one_hot = []
for i in range(len(Y_numerico)):
    linha = []
    for j in range(num_classes):
      if j == Y_numerico[i]:
        linha.append(1)
      else:
        linha.append(0)
    Y_one_hot.append(linha)
```

```
Y_final = np.array(Y_one_hot)


# --- MODELO E TREINAMENTO ---

modelo = Sequential()
modelo.add(Dense(32, input_dim=X_final.shape[1], activation='relu'))
# A camada de saída tem o número de neurônios igual ao número de classes
modelo.add(Dense(num_classes, activation='sigmoid'))

# Compila o modelo
modelo.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Treina o Modelo
print("Iniciando o treinamento...")
resultado = modelo.fit(X_final, Y_final, batch_size=32, epochs=10000, verbose=0)

# --- CAPTURAR ACURÁCIA FINAL ---
acuracia_final = resultado.history['accuracy'][-1]
print(f"\nAcurácia final do treinamento: {acuracia_final:.2%}")

# Mostra a rede
string_io = io.StringIO()
with redirect_stdout(string_io):
    modelo.summary()
resumo_modelo_texto = string_io.getvalue()

print("\nResumo do Modelo:")
print(resumo_modelo_texto)



# Mostra o gráfico do histórico de treinamento
plt.plot(resultado.history['loss'])
plt.plot(resultado.history['accuracy'])
plt.title('Histórico de Treinamento')
plt.ylabel('Função de custo / Acurácia')
plt.xlabel('Épocas de treinamento')
plt.legend(['Erro treino', 'Acurácia Treino'])
plt.savefig('./result/Historico_de_treinamento.png') # Mantém o salvamento da imagem

plt.show()



# --- DADOS DE TESTE E PREVISÃO ---

# Carrega conjunto de teste
testes = pd.read_csv('./content/Teste_NPC.txt', header=None)

# Separa as features de teste
Xtestes_df = testes.iloc[:, 0:QUANTIDADE_COLUNAS-1]

# Conversão de  texto para número usada no treinamento
```

```
Xtestes_processado = Xtestes_df.copy()
for col_idx in [1, 2, 3]:
    mapa = mapeamento_features[col_idx]
    Xtestes_processado[col_idx] = Xtestes_df[col_idx].map(mapa).fillna(-1) # -1 para categorias não vistas

# Normaliza os dados de teste
Xtestes_final = scaler.transform(Xtestes_processado)

# Testa a rede
Y_predito_prob = modelo.predict(Xtestes_final)
print("\nProbabilidades Preditas:\n", Y_predito_prob[:5]) # Mostra as 5 primeiras

# Converte as previsões de volta para a classe original
indices_preditos = np.argmax(Y_predito_prob, axis=1)
classes_preditas = legenda_classes[indices_preditos]

print("\nÍndices das Classes Preditas:", indices_preditos)
print("\nLegenda das Classes:", legenda_classes)
print("\nClasses Preditas:\n", classes_preditas)



# Ver Resposta
Y_Resposta = pd.DataFrame(data=classes_preditas, columns=['Previsão de Classe'])
print("\nResposta Final:")
print(Y_Resposta)



# ----------------------------------------------------------------
# --- SALVANDO OS RESULTADOS EM ARQUIVOS EXTERNOS ---
# ----------------------------------------------------------------

print("\nSalvando resultados em arquivos...")

# Salvar o resumo do modelo em um arquivo de texto
with open('./result/resumo_modelo.txt', 'w', encoding='utf-8') as f:
    f.write(resumo_modelo_texto)

# Salvar a legenda das classes em um arquivo de texto
with open('./result/legenda_classes.txt', 'w', encoding='utf-8') as f:
    f.write(str(legenda_classes))

# Salvar o DataFrame de respostas em um arquivo CSV para facilitar a leitura
Y_Resposta.to_csv('./result/previsoes_finais.csv', index=False)

# --- [NOVO] SALVAR A ACURÁCIA FINAL ---
with open('./result/acuracia.txt', 'w', encoding='utf-8') as f:
    f.write(f"{acuracia_final:.2%}")

print("Arquivos 'resumo_modelo.txt', 'legenda_classes.txt', 'previsoes_finais.csv' e 'acuracia.txt' salvos com
sucesso!")
```