Estudo de Caso
Redes Neurais Artificiais - Tipos de Redes Neurais Artificiais

QUESTÃO MOTRIZ

Coloque-se no lugar de Patrícia, crie uma rede neural em Python e avalie a precisão dela.

Implementação:
Repliquei o código da Patrícia, com algumas exceções:

1. O .csv não foi fornecido, logo, importei o dataset diretamente da biblioteca fornecida pela Cleveland Clinic Foundation para Python.

2. Retirei todos os tratamentos que a Patrícia implementou para visualizar os dados e gerar gráficos.

3. Retirei o tratamento que a Patrícia fez com o método get_dumies(), pois em análise, percebi que esse tratamento estava impactando negativamente no treinamento do modelo.

4. Precisei retirar as linhas que houvessem valores nulos com o método dropna(), pois haviam 4 linhas no dataset com valores nulos que estavam impactando no treinamento do modelo.

5. O dataset evoluiu da data da elaboração da ementa, agora a coluna NUM (target), não contêm apenas os valores 0 (saudável) e 1 (doente), sendo uma variação de 0 até 4, sendo 0 saudável e 1 em diante o grau da doença cardíaca.
   Como o importante para o modelo era saber se a pessoa tem potencial de estar ou ficar doente em futuro próximo, converti todos os valores acima de 0 para 1, ficando assim com os valores iguais aos da época da elaboração da ementa.

   Obs.: Todas essas alterações podem ser visualizadas abaixo:

```
# Buscar o dataset
raw_dataset = fetch_ucirepo(id=45)

# Extrair os valores dos dados corretamente
data_values = raw_dataset.data['features']
target_values = raw_dataset.data['targets']

# Criar um DataFrame com os dados principais
dataset = pd.DataFrame(data_values)

# Adicionar a coluna 'NUM' ao DataFrame principal
dataset['NUM'] = target_values

# Nomes que serão dados as colunas na mesma ordem em que aparecem
heart_disease_cols = [ 'Idade',
 'Sexo',
 'Dor_Peito',
 'PA_rep',
 'Col_Serico',
 'ASJ',
 'ECG_rep',
 'MFC',
 'AINEX',
 'DEPSTEX',
 'INCLI',
 'CA',
 'TAL',
 'NUM']
```

```python
# Renomear as colunas do DataFrame
dataset.columns = heart_disease_cols

dataset['NUM'] = dataset['NUM'].replace({2: 1})
dataset['NUM'] = dataset['NUM'].replace({3: 1})
dataset['NUM'] = dataset['NUM'].replace({4: 1})

# Remover valores nulos
dataset.dropna(inplace=True)

target_values = dataset['NUM'].copy()

# Separar as features e o target
X = dataset.drop('NUM', axis=1).copy()
y = target_values
```

✓ 4.0s Python

A partir desse ponto, crio o modelo da rede neural utilizando a biblioteca Keras com auxilio e sugestões do Copilot da Microsoft, tentei diversas combinações e esse modelo é o que alcancei a melhor acurácia:

- Adiciono uma camada densa com 128 neurônios.
- Uso a função de ativação Relu (Rectified Linear Unit), que é comum em redes neurais por ajudar a resolver problemas de gradiente.
- Adiciono uma camada de dropout que desativa aleatoriamente 50% dos neurônios durante o treinamento para evitar overfitting.
- Adiciono outra camada densa com 64 neurônios.
- Usa a função de ativação Relu novamente.
- Adiciono outra camada de dropout que desativa aleatoriamente 50% dos neurônios.
- Adiciono outra camada densa com 32 neurônios.
- Usa a função de ativação Relu novamente.
- Adiciono a camada de saída com 1 neurônio.
- Uso a função de ativação sigmoide, que é adequada para problemas de classificação binária, pois retorna um valor entre 0 e 1.

  Obs.: Este modelo é uma rede neural com três camadas ocultas e uma camada de saída.
  As camadas ocultas usam a função de ativação Relu e têm dropout para evitar overfitting.
  A camada de saída usa a função de ativação sigmoid para prever a probabilidade de uma classe binária.

```
Click to add a breakpoint et em conjuntos de treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Normalizar os dados
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Construir o modelo
model = Sequential()
model.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compilar o modelo
model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])

# Implementar early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Treinar o modelo
model.fit(X_train, y_train, epochs=60, batch_size=4, validation_split=0.20, callbacks=[early_stopping])
```

```
# Avaliar o modelo
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Model Loss: {loss}')
print(f'Model Accuracy: {accuracy}')
```

- Uso o otimizador Adam com uma taxa de aprendizado de 0.0001. Adam é um otimizador adaptativo que combina as vantagens dos otimizadores AdaGrad e RMSProp.
- Uso a função de perda de entropia cruzada binária, que é adequada para problemas de classificação binária.
- Avalio o desempenho do modelo usando a métrica de precisão.
- Implemento o Early Stopping, que monitora a perda de validação e para o treinamento se não houver melhoria após 10 épocas consecutivas, restaurando os melhores pesos:
    - Monitoro a perda de validação durante o treinamento.
    - Paro o treinamento se a perda de validação não melhorar após 10 épocas consecutivas.
    - Restauro os pesos do modelo para os melhores valores observados durante o treinamento, evitando overfitting.
- Uso 60 épocas para treinar o modelo. Uma época é uma passagem completa pelo conjunto de dados de treinamento.
- Uso 4 amostras por lote de atualização de gradiente. Um tamanho de lote menor pode levar a uma convergência mais rápida.
- Uso 20% dos dados de treinamento para validação. Isso ajuda a monitorar o desempenho do modelo em dados não vistos durante o treinamento.
- Usa o callback de early stopping para parar o treinamento antecipadamente se a perda de validação não melhorar.

Resumo:

Com isso consegui um modelo que retorna um valor de perda bem baixo e com uma acurácia variando de 88% a 91%, o que avaliei como aceitável dada a quantidade baixa de dados para treinamento.

```
Model Loss: 0.3291029632091522
Model Accuracy: 0.9111111164093018
```