Universidade Federal do Rio de Janeiro
Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia

Programa de Engenharia de Sistemas e Computação

CPS769 - Introdução à Inteligência Artificial e Aprendizagem Generativa

Prof. Dr. Edmundo de Souza e Silva (PESC/COPPE/UFRJ)
Profa. Dra. Rosa M. Leão (PESC/COPPE/UFRJ)
Participação Especial: Gaspare Bruno (Diretor Inovação, ANLIX)

Lista de Exercícios 2

Luiz Henrique Souza Caldas
email: lhscaldas@cos.ufrj.br

25 de julho de 2024

Questão 1

O objetivo deste trabalho é entender como construir um modelo preditivo simples (de Redes Neurais) usando uma ou mais camadas ocultas, e se familiarizar com os códigos em Python. Nesta tarefa, você deverá construir um modelo de rede neural simples para prever a nota de cada aluno em uma turma com base em duas features:

- Fração de palestras assistidas
- Número de horas estudadas por semana (até o máximo de 8 horas em uma semana).

Sobre a Rede Neural e programa:

- A sua rede neural deverá ter uma camada oculta com 3 neurônios. Portanto teremos 2 entradas (as 2 features), uma camada oculta e, na camada de saída, um neurônio.
- Os dados de entrada são fornecidos em uma planilha .ods (libreoffice), com os dados de 500 estudantes.
- Uma vez lido, o seu dataset deve ser aleatoriamente dividido de forma a que 80% seja para treino do modelo e os restantes 20% para teste.
- No seu programa:
  - Use um Scaler padrão do sklearn para dimensionar os dados de treinamento. Explique o motivo de escalonar os dados de entrada.
  - Para a camada oculta, use a função de ativação ReLU (Rectified Linear Unit).
  - A camada de saída usa a ativação linear padrão (explique o que é).
  - Use o algoritmo de otimização Adam (Adaptive Moment Estimation). Explique bem resumidamente as vantagens em relação ao Stochastic Gradient Descent padrão.
  - Use mean square error para a função de perda. Explique bem resumidamente o objetivo da função de perda.

Responda:

1. Como o modelo de rede neural está estruturado? Explique a arquitetura.
2. Explique o papel da função de ativação usada na camada oculta.
3. Treine o modelo com o conjunto de dados fornecido. Qual o erro quadrático médio nos dados de teste?
4. Trace o erro quadrático médio em função das "épocas" (dos passos para a convergência). Descreva a tendência que você observa.
5. Mostre os pesos e bias de cada camada após o treinamento, isto é, mostre os parâmetros aprendidos do modelo.

6. Use o modelo treinado para prever as notas a partir dos dados de novos alunos (com um segundo dataset fornecido sem as notas). Mostre as previsões feitas pelo modelo e explique os resultados.

7. Modifique o programa para adicionar mais camadas e/ou maior número de neurônios ocultos. Como suas modificações afetam o desempenho do modelo?

8. Quais seriam algumas melhorias potenciais ou recursos adicionais que poderiam ser adicionados ao modelo para melhorar sua precisão preditiva?

9. Faria sentido usar a função de ativação sigmoid no modelo? Explique em poucas palavras.

Código

O código abaixo encontra-se no repositório https://github.com/lhscaldas/cps769-ai-gen, bem como o arquivo LaTex com o relatório.

Código 1: código fornecido completo com algumas modificações

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

class NeuralNetworkModel:
    def __init__(self, data_path):
        self.data_path = data_path
        self.model = None
        self.history = None
        self.scaler = StandardScaler()

    def load_data(self):
        self.data = pd.read_csv(self.data_path, decimal=',', dtype=float)
        self.X = self.data[['Presença', 'HorasEstudo']]
        self.y = self.data['Nota']

    def preprocess_data(self):
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split
            (self.X, self.y, test_size=0.2, random_state=42)
        self.X_train_scaled = self.scaler.fit_transform(self.X_train)
        self.X_test_scaled = self.scaler.transform(self.X_test)

    def build_model(self):
        self.model = tf.keras.Sequential([
            tf.keras.layers.InputLayer(input_shape=(2,)),
            tf.keras.layers.Dense(3, activation='relu'),
            tf.keras.layers.Dense(1)
        ])
        self.model.compile(optimizer='adam', loss='mean_squared_error')

```

```
    def train_model(self, epochs=600):
        self.history = self.model.fit(self.X_train_scaled, self.y_train, epochs
            =epochs, validation_split=0.2, verbose=0)

    def evaluate_model(self):
        test_loss = self.model.evaluate(self.X_test_scaled, self.y_test)
        print(f'Mean Squared Error on Test Data: {test_loss}')

    def plot_loss(self):
        plt.plot(self.history.history['loss'], label='Train Loss')
        plt.plot(self.history.history['val_loss'], label='Validation Loss')
        plt.xlabel('Epochs')
        plt.ylabel('Mean Squared Error')
        plt.legend()
        plt.show()

    def show_weights_and_biases(self):
        for layer in self.model.layers:
            weights, biases = layer.get_weights()
            print(f'Pesos da camada: {weights}')
            print(f'Bias da camada: {biases}')

    def predict_new_data(self, new_data_path):
        new_data = pd.read_csv(new_data_path, decimal='.', dtype=float)
        new_X = new_data[['Presença', 'HorasEstudo']]
        new_X_scaled = self.scaler.transform(new_X)
        predictions = self.model.predict(new_X_scaled, verbose=0)
        new_data['Nota'] = predictions
        new_data.to_csv('lista_2/lista_2-students_data_new.csv', index=False)
        print(f'Previsões salvas no arquivo lista_2-students_data_new.csv')

    def modify_and_train_new_model(self, new_layers, epochs=500):
        modified_model = tf.keras.Sequential()
        modified_model.add(tf.keras.layers.InputLayer(input_shape=(2,)))

        for units in new_layers:
            modified_model.add(tf.keras.layers.Dense(units, activation='relu'))

        modified_model.add(tf.keras.layers.Dense(1))
        modified_model.compile(optimizer='adam', loss='mean_squared_error')

        modified_history = modified_model.fit(self.X_train_scaled, self.y_train
            , epochs=epochs, validation_split=0.2, verbose=0)
        modified_test_loss = modified_model.evaluate(self.X_test_scaled, self.
            y_test)
        print(f'Mean Squared Error on Test Data with Modified Model: {
            modified_test_loss}')

if __name__ == "__main__":
    data_path = 'lista_2/lista_2-students_data.csv'
    new_data_path = 'lista_2/lista_2-students_data_new.csv'

    nn_model = NeuralNetworkModel(data_path)
    nn_model.load_data()
```

```
    nn_model.preprocess_data()
    nn_model.build_model()
    nn_model.train_model()
    nn_model.evaluate_model()
    nn_model.plot_loss()
    nn_model.show_weights_and_biases()
    nn_model.predict_new_data(new_data_path)
    nn_model.modify_and_train_new_model([5, 3])
```