Rizqy Asyraff Athallah

1103210158

Week 10

1. Prepocessing Data

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer

# Load dataset
data = pd.read_csv('/content/drive/MyDrive/StudentsPerformance.csv')

# Target variable
target = "math score"

# Separate features and target
X = data.drop(columns=[target])
y = data[target]

# Identify categorical and numerical columns
categorical_columns = X.select_dtypes(include=["object"]).columns
numerical_columns = X.select_dtypes(include=["number"]).columns

# Encode categorical columns and scale numerical columns
preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numerical_columns),
        ("cat", OneHotEncoder(drop="first"), categorical_columns),
    ]
)

# Apply transformations
X = preprocessor.fit_transform(X)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Convert target to NumPy arrays
y_train = np.array(y_train)
y_test = np.array(y_test)
```

```
print("Shape of X_train:", X_train.shape)
print("Shape of y_train:", y_train.shape)
```

- Memilih *math score* sebagai target dan fitur lain nya menjadi variable input
- Kolom seperti gender, race/ethnicity, dll., adalah kategorikal dan perlu diubah menjadi numerik (encoding) untuk dipahami oleh model.
- Kolom numerik seperti math score, reading score, writing score harus diskalakan agar setiap fitur memiliki skala serupa, yang penting untuk algoritma berbasis gradien.
- **OneHotEncoder** = Mengonversi kategori menjadi representasi biner. Contohnya, gender ("male", "female") akan dikodekan menjadi dua kolom (0 atau 1).
- Scaling dengan StandardScaler= Fitur numerik dinormalisasi dengan nilai rata-rata 0 dan standar deviasi 1. Bertujuan unutk mempercepat konvergensi model.\
- **Train – Test Split** = Dataset dibagi menjadi **80% untuk pelatihan** dan **20% untuk pengujian**. Ini memastikan bahwa model tidak "menghafal" data.

2. Model MLP dengan Pytorch

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset

# Convert data to PyTorch tensors
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32)

# Prepare DataLoader
batch_size = 32
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

# Define MLP Model
class MLPModel(nn.Module):
    def __init__(self, input_size, hidden_layers, activation):
        super(MLPModel, self).__init__()
        layers = []
        prev_size = input_size
        for hidden_size in hidden_layers:
            layers.append(nn.Linear(prev_size, hidden_size))
            layers.append(activation())
            prev_size = hidden_size
        layers.append(nn.Linear(prev_size, 1))  # Output layer
```

```
        self.model = nn.Sequential(*layers)

    def forward(self, x):
        return self.model(x)

# Example: MLP with 1 hidden layer of 32 neurons and ReLU
input_size = X_train.shape[1]
hidden_layers = [32]
activation = nn.ReLU

model = MLPModel(input_size, hidden_layers, activation)
```

- **Pytoch Tensors** = Data dikonversi ke tensor, yang merupakan struktur data utama di PyTorch untuk melakukan operasi matematis yang dioptimalkan.
- **Data Loader** = Batch data (misalnya ukuran batch = 32) memungkinkan model untuk memproses sebagian kecil data pada satu waktu, sehingga penggunaan memori lebih efisien.
- **Input Layer**: Ukurannya sesuai dengan jumlah fitur yang dihasilkan setelah preprocessing (contoh: 19 fitur setelah encoding).
- **Hidden Layers**: Meningkatkan kapasitas model untuk belajar pola non-linear. Kita mulai dengan konfigurasi 1 hidden layer dengan 32 neuron.
- **Activation Function**: Fungsi aktivasi (misalnya ReLU) menambahkan non-linearitas sehingga model dapat mempelajari hubungan yang lebih kompleks.
- **Output Layer**: Menghasilkan prediksi tunggal (skor matematika) karena ini adalah masalah regresi.

3. Training Model

```
def train_model(model, train_loader, epochs, learning_rate):
    # Define loss and optimizer
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    for epoch in range(epochs):
        model.train()
        epoch_loss = 0
        for X_batch, y_batch in train_loader:
            optimizer.zero_grad()
            y_pred = model(X_batch).squeeze()
            loss = criterion(y_pred, y_batch)
            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()
        print(f"Epoch {epoch+1}/{epochs}, Loss: {epoch_loss/len(train_loader)}")

# Train the model
```

```
train_model(model, train_loader, epochs=10, learning_rate=0.01)
```

- **Fungsi Loss – MSE** = Mean Squared Error (MSE) digunakan sebagai fungsi loss karena ini adalah regresi. Tujuan model adalah meminimalkan rata-rata kesalahan kuadrat antara prediksi dan nilai sebenarnya.
- **Optimizer Adam** = Optimizer ini lebih stabil dibandingkan SGD karena secara adaptif menyesuaikan learning rate untuk setiap parameter.
- **Training Loop** = Proses iteratif di mana model terus memperbarui bobot berdasarkan loss hingga mencapai jumlah epoch tertentu.
- **Epochs**: Setiap epoch adalah satu siklus penuh di mana model melihat seluruh dataset.

4. Experimen dengan Parameter

```
# Define parameter combinations
hidden_layers_list = [[32], [32, 16], [64, 32, 16]]  # 1, 2, 3 hidden layers
activation_functions = [nn.ReLU, nn.Sigmoid]
epochs_list = [10, 50]
learning_rates = [0.01, 0.001]
batch_sizes = [32, 128]

# Loop through parameters
results = []
for hidden_layers in hidden_layers_list:
    for activation in activation_functions:
        for epochs in epochs_list:
            for lr in learning_rates:
                for batch_size in batch_sizes:
                    print(f"Testing: {hidden_layers}, {activation.__name__}, Epochs={epochs}, LR={lr},
Batch={batch_size}")

                    # Update DataLoader with new batch size
                    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

                    # Initialize model
                    model = MLPModel(input_size, hidden_layers, activation)

                    # Train model
                    train_model(model, train_loader, epochs, lr)
```

- **Hidden Layer** = Menambah hidden layers memungkinkan model mempelajari representasi yang lebih kompleks, tetapi dapat menyebabkan overfitting jika dataset kecil. Kombinasi yang diuji adalah 1, 2, dan 3 hidden layers.
- **Neuron per Layer** = Ukuran layer memengaruhi kapasitas model untuk belajar. Layer kecil mungkin tidak cukup kompleks, sementara layer besar dapat terlalu lambat atau overfitting.

- **Activation Functions** = Fungsi aktivasi seperti ReLU membantu menangkap hubungan non-linear. Sigmoid dan Tanh sering digunakan tetapi cenderung menyebabkan gradien menghilang pada jaringan yang dalam.
- **Epoch** = Semakin tinggi jumlah epochs, semakin lama model dilatih. Namun, jika terlalu banyak, model dapat overfit (terlalu "menghafal" data). Dipilih 10 hingga 50 epochs untuk keseimbangan waktu dan performa.
- **Learning Rate** = Learning rate menentukan seberapa besar langkah model saat memperbarui bobot. Nilai rendah seperti 0.001 memberikan pembelajaran stabil tetapi lambat, sedangkan nilai besar seperti 0.1 lebih cepat tetapi berisiko kehilangan solusi optimal.
- **Batch Size** = Ukuran batch yang lebih kecil memungkinkan pembaruan lebih sering tetapi lambat, sedangkan batch besar membutuhkan lebih sedikit iterasi.

5. Evaluasi Model

```
def evaluate_model(model, X_test_tensor, y_test_tensor):
    model.eval()
    with torch.no_grad():
        y_pred = model(X_test_tensor).squeeze()
        mse = nn.MSELoss()(y_pred, y_test_tensor)
    print(f"Test MSE: {mse.item()}")

# Evaluate model
evaluate_model(model, X_test_tensor, y_test_tensor)
```

- **Evaluasi MSE** = Digunakan untuk mengukur seberapa baik model memprediksi math score. MSE yang rendah menunjukkan prediksi mendekati nilai sebenarnya.
- **Model Evaluasi** = Model diatur dalam **mode evaluasi** (model.eval()) agar tidak menghitung gradien selama inferensi, mempercepat evaluasi.