CSE425

Assignment 1

Name: Muhammad Nayeem Mubasshirul Haque

ID: 19101115

Sec: 03

Implementing a Vanilla RNN and Analyzing its Limitations

Introduction

In this report, the implementation of a Vanilla Recurrent Neural Network (RNN) for the prediction of diabetes using a health dataset has been shown. The goal is to explore the capabilities of a basic RNN architecture in predicting the presence of diabetes based on health-related features.

Dataset

The dataset is gathered from Kaggle. The diabetes dataset from the following link has been used.

https://www.kaggle.com/datasets/prosperchuks/health-dataset

The dataset used for this implementation consists of health-related features, including various medical conditions, and the target variable indicating the presence of diabetes. The dataset was preprocessed to handle missing values and exclude irrelevant columns. Here the columns in the diabetes dataset consist:

Age, Sex, HighChol, CholCheck, BMI, Smoker, HeartDiseaseorAttack, PhysActivity, Fruits, Veggies, HvyAlcoholConsump, GenHlth, MentHlth, PhysHlth, DiffWalk, Stroke, HighBP, Diabetes

Dataset Preprocessing

Firstly, we've excluded diabetes as it is our target.

x = res.drop(['Diabetes'], axis=1) # Excluding diabetes, as it is our target

x

Then we declared "y" as our target for predicting diabetes.

y = res['Diabetes'] # Our target

y

Lastly, these columns were dropped in the following manner as they hold less significance with diabetes such ase sex, fruits, mental health etc.

res = res.drop([cols[1], cols[5], cols[7], cols[8], cols[9], cols[10], cols[12]], axis=1)

res

Model Architecture

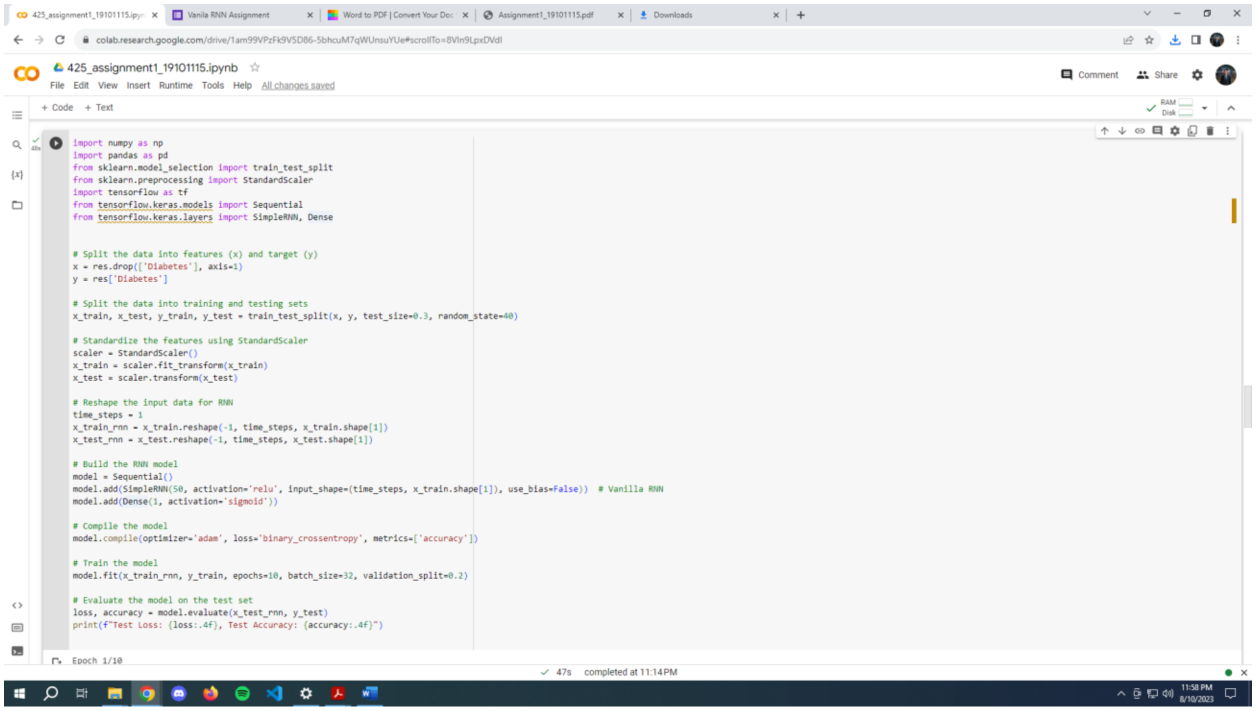
The Vanilla RNN model is constructed using the TensorFlow and Keras libraries. The architecture consists of a single RNN layer followed by a dense output layer. The model architecture is as follows:

Input Layer: A 3D input shape is expected, where the first dimension represents the number of samples, the second dimension corresponds to the sequence length (number of time steps), and the third dimension is the number of features per time step.

RNN Layer: A Vanilla RNN layer is used with 50 units and the ReLU activation function.

Output Layer: A dense output layer with 1 unit and the sigmoid activation function for binary classification.

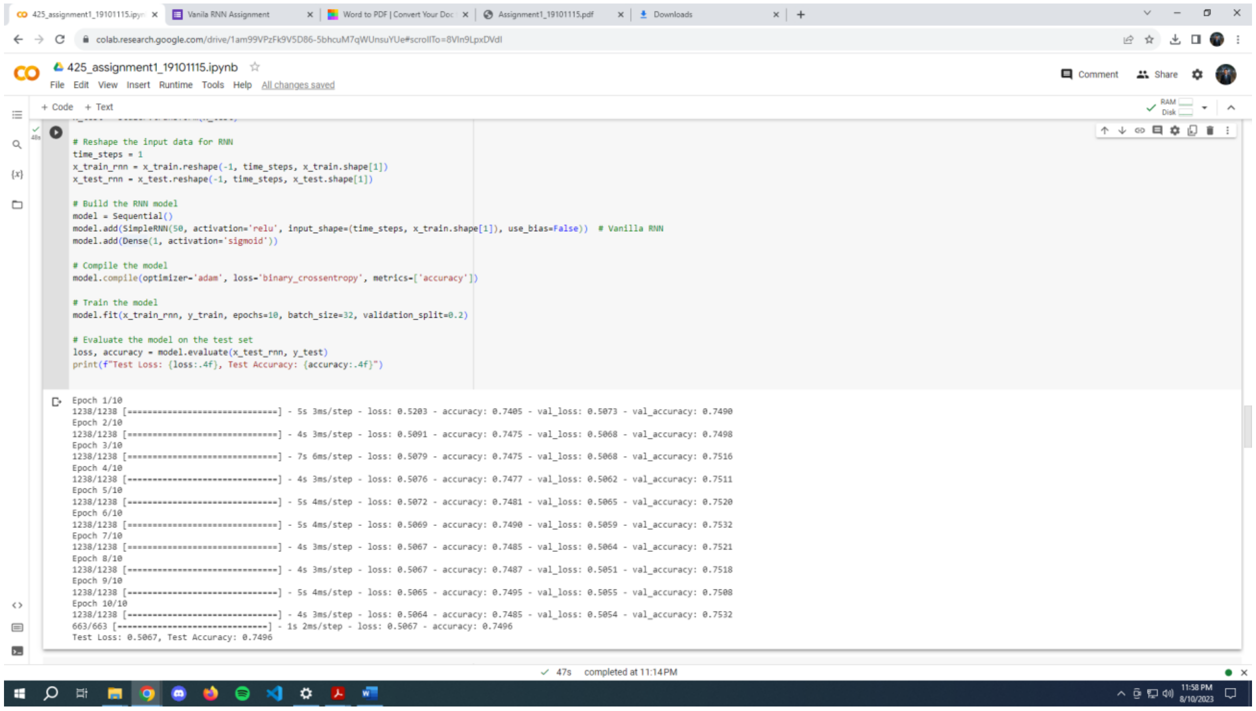
CODE:



```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense


# Split the data into features (x) and target (y)
x = res.drop(['Diabetes'], axis=1)
y = res['Diabetes']

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=40)

# Standardize the features using StandardScaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

# Reshape the input data for RNN
time_steps = 1
x_train_rnn = x_train.reshape(-1, time_steps, x_train.shape[1])
x_test_rnn = x_test.reshape(-1, time_steps, x_test.shape[1])

# Build the RNN model
model = Sequential()
model.add(SimpleRNN(50, activation='relu', input_shape=(time_steps, x_train.shape[1]), use_bias=False))  # Vanilla RNN
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(x_train_rnn, y_train, epochs=10, batch_size=32, validation_split=0.2)

# Evaluate the model on the test set
loss, accuracy = model.evaluate(x_test_rnn, y_test)
print(f"Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}")
```

```
Epoch 1/10
```

Output:



```
# Reshape the input data for RNN
time_steps = 1
x_train_rnn = x_train.reshape(-1, time_steps, x_train.shape[1])
x_test_rnn = x_test.reshape(-1, time_steps, x_test.shape[1])

# Build the RNN model
model = Sequential()
model.add(SimpleRNN(50, activation='relu', input_shape=(time_steps, x_train.shape[1]), use_bias=False))  # Vanilla RNN
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(x_train_rnn, y_train, epochs=10, batch_size=32, validation_split=0.2)

# Evaluate the model on the test set
loss, accuracy = model.evaluate(x_test_rnn, y_test)
print(f"Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}")
```

```
Epoch 1/10
1238/1238 [==============================] - 5s 3ms/step - loss: 0.5203 - accuracy: 0.7405 - val_loss: 0.5073 - val_accuracy: 0.7490
Epoch 2/10
1238/1238 [==============================] - 4s 3ms/step - loss: 0.5091 - accuracy: 0.7475 - val_loss: 0.5068 - val_accuracy: 0.7498
Epoch 3/10
1238/1238 [==============================] - 7s 6ms/step - loss: 0.5079 - accuracy: 0.7475 - val_loss: 0.5068 - val_accuracy: 0.7516
Epoch 4/10
1238/1238 [==============================] - 4s 3ms/step - loss: 0.5076 - accuracy: 0.7477 - val_loss: 0.5062 - val_accuracy: 0.7511
Epoch 5/10
1238/1238 [==============================] - 5s 4ms/step - loss: 0.5072 - accuracy: 0.7481 - val_loss: 0.5065 - val_accuracy: 0.7520
Epoch 6/10
1238/1238 [==============================] - 5s 4ms/step - loss: 0.5069 - accuracy: 0.7490 - val_loss: 0.5059 - val_accuracy: 0.7532
Epoch 7/10
1238/1238 [==============================] - 4s 3ms/step - loss: 0.5067 - accuracy: 0.7485 - val_loss: 0.5064 - val_accuracy: 0.7521
Epoch 8/10
1238/1238 [==============================] - 4s 3ms/step - loss: 0.5067 - accuracy: 0.7487 - val_loss: 0.5051 - val_accuracy: 0.7518
Epoch 9/10
1238/1238 [==============================] - 5s 4ms/step - loss: 0.5065 - accuracy: 0.7495 - val_loss: 0.5055 - val_accuracy: 0.7508
Epoch 10/10
1238/1238 [==============================] - 4s 3ms/step - loss: 0.5064 - accuracy: 0.7485 - val_loss: 0.5054 - val_accuracy: 0.7532
663/663 [==============================] - 1s 2ms/step - loss: 0.5067 - accuracy: 0.7496
Test Loss: 0.5067, Test Accuracy: 0.7496
```

Training Process

The Vanilla RNN model is compiled with the Adam optimizer and binary cross-entropy loss function. The training process involves feeding the training data into the model in batches of 32 samples. The model is trained for 10 epochs with validation data used to monitor performance during training.

Results

The model's performance is evaluated on the testing data using the binary cross-entropy loss and accuracy metrics. The achieved testing accuracy is reported as a measure of the model's ability to predict diabetes.

Limitations and Conclusion:

The implementation of a Vanilla RNN for diabetes prediction demonstrates a foundational approach to sequence modeling using neural networks. While the Vanilla RNN architecture is a simple and intuitive choice, it might face challenges in capturing long-term dependencies and complex relationships in the data.

The vanishing gradient problem in vanilla RNNs occurs when the gradients of the loss function with respect to the network's parameters become extremely small during backpropagation, leading to slow or stalled learning. Conversely, the exploding gradient problem arises when these gradients become excessively large, causing instability and difficulties in convergence during training.For our implemented diabetes prediction model, the presence of these gradient problems can hinder the RNN's capacity to effectively capture intricate temporal relationships inherent in health-related data.

Future alternatives could involve experimenting with more advanced architectures like LSTM or GRU to potentially overcome these limitations.