task1

May 15, 2024

1 Task 1

A Feed-Forward Neural Network (FFNN) is a fundamental type of artificial neural network characterized by a unidirectional data flow from input to output without any feedback loops. It consists of an input layer that receives the initial data, one or more hidden layers where the main computational work occurs, and an output layer that produces the final prediction or classification. Each neuron in these layers applies an activation function, such as ReLU, Sigmoid, or Tanh, to introduce non-linearity. The network is parameterized by weights and biases, which are adjusted during training through a process called backpropagation, using optimization algorithms like gradient descent to minimize error. This straightforward architecture makes FFNNs suitable for various tasks, including classification and regression, by mapping input features to outputs through learned patterns.

[18]:
```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# For demonstration, we create a random dataset
np.random.seed(42)
data_size = 1000
feature_size = 20
data = pd.DataFrame(np.random.rand(data_size, feature_size),␣
 ↪columns=[f'feature_{i}' for i in range(feature_size)])
data['label'] = np.random.randint(0, 2, size=data_size)

# Output 10 rows from the generated data
print("Sample data (10 rows):")
print(data.head(10))

# Split data into features and labels
X = data.drop('label', axis=1)
y = data['label']

# Train-test split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define the FFNN model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu', input_shape=(feature_size,)),
    tf.keras.layers.Dropout(0.5),  # Adding dropout for regularization
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

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

# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32,␣
 ↪validation_split=0.2)

# Evaluate the model
y_pred = (model.predict(X_test) > 0.5).astype(int)
accuracy = accuracy_score(y_test, y_pred)
print(f'FFNN Accuracy: {accuracy * 100:.2f}%')
```

```
Sample data (10 rows):
   feature_0  feature_1  feature_2  feature_3  feature_4  feature_5  \
0   0.374540   0.950714   0.731994   0.598658   0.156019   0.155995
1   0.611853   0.139494   0.292145   0.366362   0.456070   0.785176
2   0.122038   0.495177   0.034389   0.909320   0.258780   0.662522
3   0.388677   0.271349   0.828738   0.356753   0.280935   0.542696
4   0.863103   0.623298   0.330898   0.063558   0.310982   0.325183
5   0.031429   0.636410   0.314356   0.508571   0.907566   0.249292
6   0.807440   0.896091   0.318003   0.110052   0.227935   0.427108
7   0.962447   0.251782   0.497249   0.300878   0.284840   0.036887
8   0.367783   0.632306   0.633530   0.535775   0.090290   0.835302
9   0.341066   0.113474   0.924694   0.877339   0.257942   0.659984

   feature_6  feature_7  feature_8  feature_9  …  feature_11  feature_12  \
0   0.058084   0.866176   0.601115   0.708073  …    0.969910    0.832443
1   0.199674   0.514234   0.592415   0.046450  …    0.170524    0.065052
2   0.311711   0.520068   0.546710   0.184854  …    0.775133    0.939499
```

```
3   0.140924   0.802197   0.074551   0.986887  …    0.198716    0.005522
4   0.729606   0.637557   0.887213   0.472215  …    0.713245    0.760785
5   0.410383   0.755551   0.228798   0.076980  …    0.161221    0.929698
6   0.818015   0.860731   0.006952   0.510747  …    0.222108    0.119865
7   0.609564   0.502679   0.051479   0.278646  …    0.239562    0.144895
8   0.320780   0.186519   0.040775   0.590893  …    0.016588    0.512093
9   0.817222   0.555201   0.529651   0.241852  …    0.897216    0.900418

   feature_13  feature_14  feature_15  feature_16  feature_17  feature_18  \
0    0.212339    0.181825    0.183405    0.304242    0.524756    0.431945
1    0.948886    0.965632    0.808397    0.304614    0.097672    0.684233
2    0.894827    0.597900    0.921874    0.088493    0.195983    0.045227
3    0.815461    0.706857    0.729007    0.771270    0.074045    0.358466
4    0.561277    0.770967    0.493796    0.522733    0.427541    0.025419
5    0.808120    0.633404    0.871461    0.803672    0.186570    0.892559
6    0.337615    0.942910    0.323203    0.518791    0.703019    0.363630
7    0.489453    0.985650    0.242055    0.672136    0.761620    0.237638
8    0.226496    0.645173    0.174366    0.690938    0.386735    0.936730
9    0.633101    0.339030    0.349210    0.725956    0.897110    0.887086

   feature_19  label
0    0.291229      1
1    0.440152      1
2    0.325330      1
3    0.115869      0
4    0.107891      0
5    0.539342      0
6    0.971782      1
7    0.728216      0
8    0.137521      1
9    0.779876      0

[10 rows x 21 columns]
Epoch 1/50

/opt/anaconda3/lib/python3.11/site-packages/keras/src/layers/core/dense.py:86:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When
using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

18/18             1s 3ms/step -
accuracy: 0.5009 - loss: 0.7560 - val_accuracy: 0.5429 - val_loss: 0.7009
Epoch 2/50
18/18             0s 876us/step -
accuracy: 0.4860 - loss: 0.7402 - val_accuracy: 0.4857 - val_loss: 0.6987
Epoch 3/50
18/18             0s 905us/step -
accuracy: 0.4942 - loss: 0.7308 - val_accuracy: 0.4929 - val_loss: 0.6965
```

```
Epoch 4/50
18/18             0s 927us/step -
accuracy: 0.5453 - loss: 0.7081 - val_accuracy: 0.4714 - val_loss: 0.6955
Epoch 5/50
18/18             0s 922us/step -
accuracy: 0.5349 - loss: 0.7069 - val_accuracy: 0.4786 - val_loss: 0.6979
Epoch 6/50
18/18             0s 863us/step -
accuracy: 0.5459 - loss: 0.6846 - val_accuracy: 0.4929 - val_loss: 0.6975
Epoch 7/50
18/18             0s 865us/step -
accuracy: 0.5876 - loss: 0.6781 - val_accuracy: 0.5071 - val_loss: 0.6961
Epoch 8/50
18/18             0s 856us/step -
accuracy: 0.5924 - loss: 0.6735 - val_accuracy: 0.5000 - val_loss: 0.6988
Epoch 9/50
18/18             0s 849us/step -
accuracy: 0.5641 - loss: 0.6922 - val_accuracy: 0.4929 - val_loss: 0.7008
Epoch 10/50
18/18             0s 841us/step -
accuracy: 0.5957 - loss: 0.6523 - val_accuracy: 0.4929 - val_loss: 0.7028
Epoch 11/50
18/18             0s 823us/step -
accuracy: 0.6065 - loss: 0.6526 - val_accuracy: 0.4857 - val_loss: 0.7031
Epoch 12/50
18/18             0s 822us/step -
accuracy: 0.5841 - loss: 0.6696 - val_accuracy: 0.5071 - val_loss: 0.7005
Epoch 13/50
18/18             0s 844us/step -
accuracy: 0.6336 - loss: 0.6522 - val_accuracy: 0.5000 - val_loss: 0.7013
Epoch 14/50
18/18             0s 810us/step -
accuracy: 0.5708 - loss: 0.6695 - val_accuracy: 0.5214 - val_loss: 0.7003
Epoch 15/50
18/18             0s 1ms/step -
accuracy: 0.6260 - loss: 0.6459 - val_accuracy: 0.5143 - val_loss: 0.7019
Epoch 16/50
18/18             0s 855us/step -
accuracy: 0.6212 - loss: 0.6485 - val_accuracy: 0.4786 - val_loss: 0.7019
Epoch 17/50
18/18             0s 839us/step -
accuracy: 0.5951 - loss: 0.6469 - val_accuracy: 0.4857 - val_loss: 0.7027
Epoch 18/50
18/18             0s 834us/step -
accuracy: 0.6461 - loss: 0.6267 - val_accuracy: 0.4857 - val_loss: 0.7088
Epoch 19/50
18/18             0s 835us/step -
accuracy: 0.5956 - loss: 0.6589 - val_accuracy: 0.5071 - val_loss: 0.7049
```

```
Epoch 20/50
18/18             0s 845us/step -
accuracy: 0.6397 - loss: 0.6367 - val_accuracy: 0.5071 - val_loss: 0.7041
Epoch 21/50
18/18             0s 856us/step -
accuracy: 0.6060 - loss: 0.6590 - val_accuracy: 0.5071 - val_loss: 0.7056
Epoch 22/50
18/18             0s 884us/step -
accuracy: 0.6325 - loss: 0.6297 - val_accuracy: 0.4857 - val_loss: 0.7058
Epoch 23/50
18/18             0s 913us/step -
accuracy: 0.6042 - loss: 0.6416 - val_accuracy: 0.4929 - val_loss: 0.7077
Epoch 24/50
18/18             0s 857us/step -
accuracy: 0.6170 - loss: 0.6427 - val_accuracy: 0.5000 - val_loss: 0.7097
Epoch 25/50
18/18             0s 874us/step -
accuracy: 0.6475 - loss: 0.5999 - val_accuracy: 0.4857 - val_loss: 0.7129
Epoch 26/50
18/18             0s 1ms/step -
accuracy: 0.6886 - loss: 0.6191 - val_accuracy: 0.4929 - val_loss: 0.7159
Epoch 27/50
18/18             0s 879us/step -
accuracy: 0.6472 - loss: 0.6316 - val_accuracy: 0.4786 - val_loss: 0.7169
Epoch 28/50
18/18             0s 867us/step -
accuracy: 0.6653 - loss: 0.6117 - val_accuracy: 0.5071 - val_loss: 0.7139
Epoch 29/50
18/18             0s 833us/step -
accuracy: 0.6657 - loss: 0.6157 - val_accuracy: 0.5071 - val_loss: 0.7131
Epoch 30/50
18/18             0s 815us/step -
accuracy: 0.6534 - loss: 0.6278 - val_accuracy: 0.5000 - val_loss: 0.7091
Epoch 31/50
18/18             0s 805us/step -
accuracy: 0.6471 - loss: 0.6285 - val_accuracy: 0.4929 - val_loss: 0.7139
Epoch 32/50
18/18             0s 822us/step -
accuracy: 0.6952 - loss: 0.5854 - val_accuracy: 0.5000 - val_loss: 0.7174
Epoch 33/50
18/18             0s 820us/step -
accuracy: 0.7389 - loss: 0.5703 - val_accuracy: 0.5143 - val_loss: 0.7195
Epoch 34/50
18/18             0s 820us/step -
accuracy: 0.6763 - loss: 0.5886 - val_accuracy: 0.5214 - val_loss: 0.7187
Epoch 35/50
18/18             0s 815us/step -
accuracy: 0.6543 - loss: 0.6185 - val_accuracy: 0.5286 - val_loss: 0.7173
```

```
Epoch 36/50
18/18             0s 820us/step -
accuracy: 0.6809 - loss: 0.5989 - val_accuracy: 0.5429 - val_loss: 0.7213
Epoch 37/50
18/18             0s 811us/step -
accuracy: 0.6889 - loss: 0.5737 - val_accuracy: 0.5214 - val_loss: 0.7281
Epoch 38/50
18/18             0s 804us/step -
accuracy: 0.6882 - loss: 0.5856 - val_accuracy: 0.5071 - val_loss: 0.7331
Epoch 39/50
18/18             0s 810us/step -
accuracy: 0.7248 - loss: 0.5621 - val_accuracy: 0.5500 - val_loss: 0.7324
Epoch 40/50
18/18             0s 789us/step -
accuracy: 0.6981 - loss: 0.5604 - val_accuracy: 0.5357 - val_loss: 0.7362
Epoch 41/50
18/18             0s 797us/step -
accuracy: 0.6882 - loss: 0.5827 - val_accuracy: 0.5357 - val_loss: 0.7378
Epoch 42/50
18/18             0s 1ms/step -
accuracy: 0.6713 - loss: 0.5852 - val_accuracy: 0.5357 - val_loss: 0.7405
Epoch 43/50
18/18             0s 844us/step -
accuracy: 0.7308 - loss: 0.5428 - val_accuracy: 0.5429 - val_loss: 0.7446
Epoch 44/50
18/18             0s 864us/step -
accuracy: 0.7215 - loss: 0.5421 - val_accuracy: 0.5357 - val_loss: 0.7527
Epoch 45/50
18/18             0s 812us/step -
accuracy: 0.7501 - loss: 0.5324 - val_accuracy: 0.5214 - val_loss: 0.7575
Epoch 46/50
18/18             0s 790us/step -
accuracy: 0.7008 - loss: 0.5574 - val_accuracy: 0.5500 - val_loss: 0.7596
Epoch 47/50
18/18             0s 823us/step -
accuracy: 0.6689 - loss: 0.6051 - val_accuracy: 0.5071 - val_loss: 0.7583
Epoch 48/50
18/18             0s 827us/step -
accuracy: 0.7260 - loss: 0.5446 - val_accuracy: 0.5071 - val_loss: 0.7586
Epoch 49/50
18/18             0s 819us/step -
accuracy: 0.7438 - loss: 0.5474 - val_accuracy: 0.4857 - val_loss: 0.7597
Epoch 50/50
18/18             0s 816us/step -
accuracy: 0.7453 - loss: 0.5304 - val_accuracy: 0.5143 - val_loss: 0.7578
10/10             0s 1ms/step
FFNN Accuracy: 49.33%
```

In this example, a basic CNN architecture is defined using Keras, and the model is trained and evaluated on the prepared dataset. Finally, the trained model is saved for deployment.

Please use the "Kernel>Restart & Run All" command in Jupyter Notebook and check your results before submitting your homework. Note that I rerun all boxes on my side before grading.