**Module 6 & Module 7**

1. The dataset is from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether a patient has diabetes using a simple ANN model

# -\*- coding: utf-8 -\*-

"""

Created on Thu May 9 20:09:16 2024

@author: Lenovo

"""

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

# Load your dataset using pandas

# Assuming you have a CSV file named 'your\_dataset.csv'

df = pd.read\_csv(r'C:/Users/Lenovo/Downloads/Study material/AI/ANN, CNN, RNN Preliminaries/Assignment/diabetes.csv')

# Assuming your dataset contains features and labels

# Replace 'features\_column\_names' and 'label\_column\_name' with actual column names from your dataset

# Split the dataset into features and labels

features = df.iloc[:, 0:8]

labels = pd.DataFrame(df['Outcome'])

# Assuming you have already preprocessed your features and labels as required

# Split the dataset into training and testing sets

# You can adjust the test\_size parameter to change the ratio of training and testing data

x\_train, x\_test, y\_train, y\_test = train\_test\_split(features, labels, test\_size=0.2, random\_state=42)

#standardizing the dataset

mean = x\_train.mean(axis=0)

x\_train -= mean

std = x\_train.std(axis=0)

x\_train /= std

x\_test -= mean

x\_test /= std

from tensorflow.keras import models

from tensorflow.keras import layers

# Define your model

model = models.Sequential()

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(1, activation='sigmoid'))

# Compile the model

model.compile(optimizer='rmsprop', loss='binary\_crossentropy', metrics=['accuracy'])

x\_val = x\_train[:500]

partial\_x\_train = x\_train[500:]

y\_val = y\_train[:500]

partial\_y\_train = y\_train[500:]

# Train the model

history = model.fit(x\_train, y\_train, epochs=20, batch\_size=100, validation\_data=(x\_val, y\_val))

# Plotting validation scores

import matplotlib.pyplot as plt

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

plt.clf()

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

# Hyperparameter tuning the model

# Assuming you have already defined and compiled your model

# Train the model with your full dataset

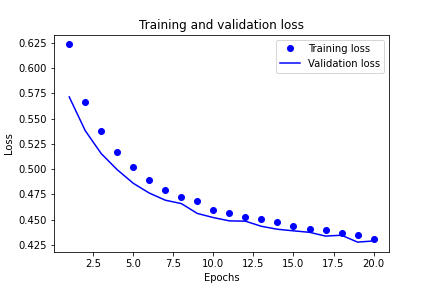
model.fit(x\_train, y\_train, epochs=4, batch\_size=100)

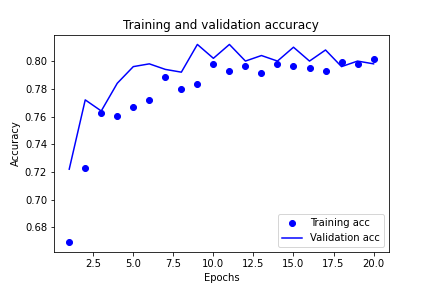
# Evaluate the model on the test data

results = model.evaluate(x\_test, y\_test)

# Make predictions

predictions = model.predict(x\_test)





7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8086 - loss: 0.4186

Out[27]: <keras.src.callbacks.history.History at 0x1cd6280d3c0>

results = model.evaluate(x\_test, y\_test)

5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.7216 - loss: 0.5362

1. Build a ANN model to predict whether a customer will churn or not. Use the attached dataset(Churn\_Modelling.csv)

# -\*- coding: utf-8 -\*-

"""

Created on Thu May 9 20:09:16 2024

@author: Lenovo

"""

import pandas as pd

from sklearn.model\_selection import train\_test\_split

# Load your dataset using pandas

# Assuming you have a CSV file named 'your\_dataset.csv'

df = pd.read\_csv(r'C:/Users/Lenovo/Downloads/Study material/AI/ANN, CNN, RNN Preliminaries/Assignment/Churn\_Modelling.csv')

# Assuming your dataset contains features and labels

# Replace 'features\_column\_names' and 'label\_column\_name' with actual column names from your dataset

# Split the dataset into features and labels

features = df.iloc[:, 0:8]

labels = pd.DataFrame(df['Exited'])

# Assuming you have already preprocessed your features and labels as required

# Split the dataset into training and testing sets

# You can adjust the test\_size parameter to change the ratio of training and testing data

x\_train, x\_test, y\_train, y\_test = train\_test\_split(features, labels, test\_size=0.2, random\_state=42)

#standardizing the dataset

mean = x\_train.mean(axis=0)

x\_train -= mean

std = x\_train.std(axis=0)

x\_train /= std

x\_test -= mean

x\_test /= std

from tensorflow.keras import models

from tensorflow.keras import layers

# Define your model

model = models.Sequential()

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(1, activation='sigmoid'))

# Compile the model

model.compile(optimizer='rmsprop', loss='binary\_crossentropy', metrics=['accuracy'])

x\_val = x\_train[:2000]

partial\_x\_train = x\_train[2000:]

y\_val = y\_train[:2000]

partial\_y\_train = y\_train[2000:]

# Train the model

history = model.fit(x\_train, y\_train, epochs=20, batch\_size=100, validation\_data=(x\_val, y\_val))

# Plotting validation scores

import matplotlib.pyplot as plt

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

plt.clf()

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

# Hyperparameter tuning the model

# Assuming you have already defined and compiled your model

# Train the model with your full dataset

model.fit(x\_train, y\_train, epochs=10, batch\_size=100)

# Evaluate the model on the test data

results = model.evaluate(x\_test, y\_test)

# Make predictions

predictions = model.predict(x\_test)

Output:

Epoch 18/20

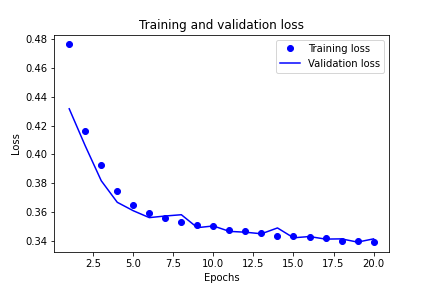
80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8621 - loss: 0.3391 - val\_accuracy: 0.8605 - val\_loss: 0.3412

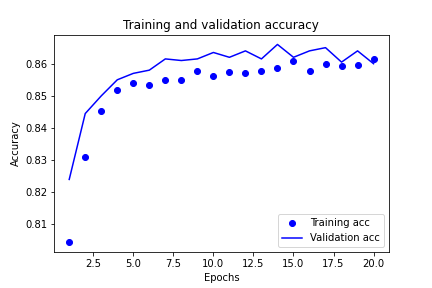
Epoch 19/20

80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8573 - loss: 0.3383 - val\_accuracy: 0.8640 - val\_loss: 0.3390

Epoch 20/20

80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8619 - loss: 0.3387 - val\_accuracy: 0.8600 - val\_loss: 0.3412





80/80 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8620 - loss: 0.3314

Out[13]: <keras.src.callbacks.history.History at 0x1cd72eedde0>

results = model.evaluate(x\_test, y\_test)

63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8629 - loss: 0.3573

1. If I have a dataset which has images of dogs and cats of size 32x32. I need to predict the output image as a dog or a cat. Fill out the parameters that is used to build a simple ANN model

Input layer should have **1024** number of neurons

Output layer should have **1** number of neurons

**Sigmoid/tanh** activation function is used in the output layer

**Binary-cross entropy** will be the loss function

1. Build an ANN model to predict Delivery Status for a logistic firm. (Refer fedex dataset)

# -\*- coding: utf-8 -\*-

"""

Created on Thu May 9 20:09:16 2024

@author: Lenovo

"""

import pandas as pd

from sklearn.model\_selection import train\_test\_split

# Load your dataset using pandas

# Assuming you have a CSV file named 'your\_dataset.csv'

df = pd.read\_csv(r'C:/Users/Lenovo/Downloads/Study material/AI/ANN, CNN, RNN Preliminaries/Assignment/fedex.csv')

# Assuming your dataset contains features and labels

# Replace 'features\_column\_names' and 'label\_column\_name' with actual column names from your dataset

# Split the dataset into features and labels

features = df.iloc[:, :6]

labels = pd.DataFrame(df['Delivery\_Status'])

# Assuming you have already preprocessed your features and labels as required

# Split the dataset into training and testing sets

# You can adjust the test\_size parameter to change the ratio of training and testing data

x\_train, x\_test, y\_train, y\_test = train\_test\_split(features, labels, test\_size=0.2, random\_state=42)

#standardizing the dataset

mean = x\_train.mean(axis=0)

x\_train -= mean

std = x\_train.std(axis=0)

x\_train /= std

x\_test -= mean

x\_test /= std

from tensorflow.keras import models

from tensorflow.keras import layers

from tensorflow.keras.layers import LeakyReLU

# Define your model

model = models.Sequential()

model.add(layers.Dense(16, activation=LeakyReLU(alpha=0.1)))

model.add(layers.Dense(16, activation=LeakyReLU(alpha=0.1)))

model.add(layers.Dense(1, activation='tanh'))

# Compile the model

model.compile(optimizer='rmsprop', loss='binary\_crossentropy', metrics=['accuracy'])

x\_val = x\_train[:300]

partial\_x\_train = x\_train[300:]

y\_val = y\_train[:300]

partial\_y\_train = y\_train[300:]

# Train the model

history = model.fit(partial\_x\_train, partial\_y\_train, epochs=20, batch\_size=100, validation\_data=(x\_val, y\_val))

# Plotting validation scores

import matplotlib.pyplot as plt

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

plt.clf()

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

# Hyperparameter tuning the model

# Assuming you have already defined and compiled your model

# Train the model with your full dataset

model.fit(x\_train, y\_train, epochs=10, batch\_size=100)

# Evaluate the model on the test data

results = model.evaluate(x\_test, y\_test)

# Make predictions

predictions = model.predict(x\_test)

Output:

Epoch 9/10

12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4334 - loss: nan

Epoch 10/10

12/12 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.4352 - loss: nan

Out[78]: <keras.src.callbacks.history.History at 0x1cd731fd720>

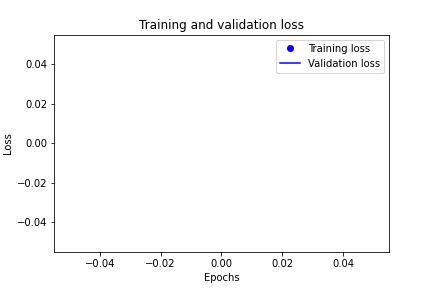
results = model.evaluate(x\_test, y\_test)

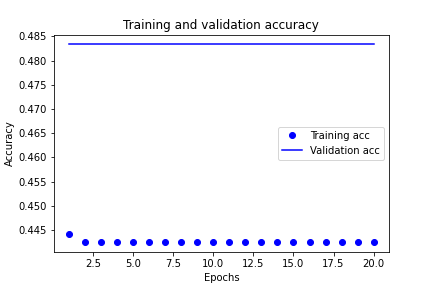
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5081 - loss: nan

# Make predictions

predictions = model.predict(x\_test)

10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step





1. Build a ANN classification model to predict breast cancer(malignant or benign)

# -\*- coding: utf-8 -\*-

"""

Created on Thu May 9 20:09:16 2024

@author: Lenovo

"""

import pandas as pd

from sklearn.model\_selection import train\_test\_split

# Load your dataset using pandas

# Assuming you have a CSV file named 'your\_dataset.csv'

df = pd.read\_csv(r'C:/Users/Lenovo/Downloads/Study material/AI/ANN, CNN, RNN Preliminaries/Assignment/breast\_cancer.csv')

# Assuming your dataset contains features and labels

# Replace 'features\_column\_names' and 'label\_column\_name' with actual column names from your dataset

# Split the dataset into features and labels

features = df.iloc[:, 1:]

labels = pd.DataFrame(df['diagnosis'])

# Assuming you have already preprocessed your features and labels as required

# Split the dataset into training and testing sets

# You can adjust the test\_size parameter to change the ratio of training and testing data

x\_train, x\_test, y\_train, y\_test = train\_test\_split(features, labels, test\_size=0.2, random\_state=42)

#standardizing the dataset

mean = x\_train.mean(axis=0)

x\_train -= mean

std = x\_train.std(axis=0)

x\_train /= std

x\_test -= mean

x\_test /= std

from tensorflow.keras import models

from tensorflow.keras import layers

# Define your model

model = models.Sequential()

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(1, activation='sigmoid'))

# Compile the model

model.compile(optimizer='rmsprop', loss='binary\_crossentropy', metrics=['accuracy'])

x\_val = x\_train[:90]

partial\_x\_train = x\_train[90:]

y\_val = y\_train[:90]

partial\_y\_train = y\_train[90:]

# Train the model

history = model.fit(x\_train, y\_train, epochs=20, batch\_size=100, validation\_data=(x\_val, y\_val))

# Plotting validation scores

import matplotlib.pyplot as plt

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(1, len(loss) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

plt.clf()

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()

# Hyperparameter tuning the model

# Assuming you have already defined and compiled your model

# Train the model with your full dataset

model.fit(x\_train, y\_train, epochs=10, batch\_size=100)

# Evaluate the model on the test data

results = model.evaluate(x\_test, y\_test)

# Make predictions

predictions = model.predict(x\_test)

**Output:**

Epoch 18/20

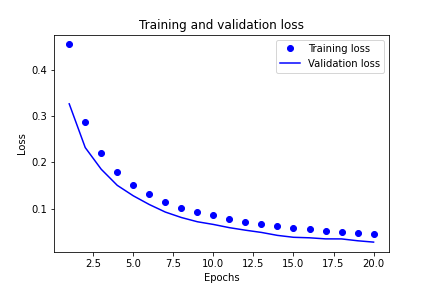
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9916 - loss: 0.0405 - val\_accuracy: 0.9889 - val\_loss: 0.0346

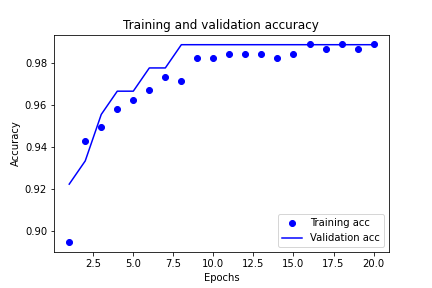
Epoch 19/20

5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.9920 - loss: 0.0399 - val\_accuracy: 0.9889 - val\_loss: 0.0306

Epoch 20/20

5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9881 - loss: 0.0491 - val\_accuracy: 0.9889 - val\_loss: 0.0277





Epoch 10/10

5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9904 - loss: 0.0279

Out[21]: <keras.src.callbacks.history.History at 0x1cd730c32e0>

# Evaluate the model on the test data

results = model.evaluate(x\_test, y\_test)

4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9707 - loss: 0.0919