

Spring 2024: CS5720 Neural Networks & Deep Learning - ICP-6
Bhanu Chandrika Lakkimsetti (700747439)
ICP_Basics in Keras

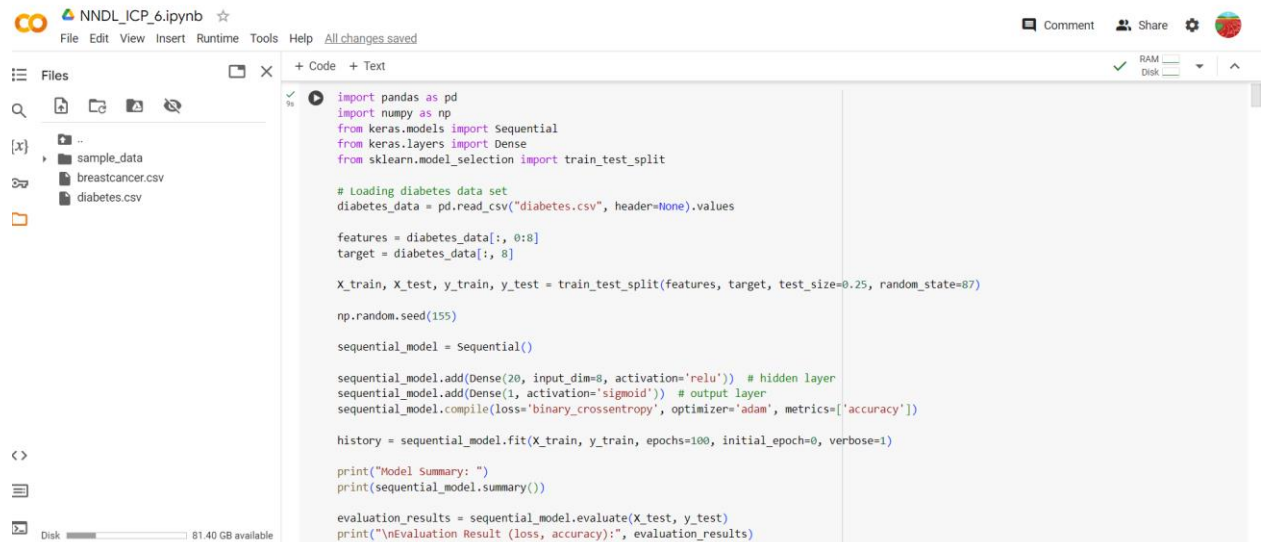
GitHub Link: https://github.com/bhanuchandrika99/NNDL_ICP_6

Use Case Description: Predicting the diabetes disease

Programming elements: Keras Basics

In class programming:

1. Use the use case in the class:



```
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split

# Loading diabetes data set
diabetes_data = pd.read_csv("diabetes.csv", header=None).values

features = diabetes_data[:, 0:8]
target = diabetes_data[:, 8]

X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.25, random_state=87)

np.random.seed(155)

sequential_model = Sequential()

sequential_model.add(Dense(20, input_dim=8, activation='relu')) # hidden layer
sequential_model.add(Dense(1, activation='sigmoid')) # output layer
sequential_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

history = sequential_model.fit(X_train, y_train, epochs=100, initial_epoch=0, verbose=1)

print("Model Summary: ")
print(sequential_model.summary())

evaluation_results = sequential_model.evaluate(X_test, y_test)
print("\nEvaluation Result (loss, accuracy):", evaluation_results)
```

+ Code + Text

9s



```
Epoch 1/100
18/18 [=====] - 2s 2ms/step - loss: 22.6529 - accuracy: 0.6615
Epoch 2/100
18/18 [=====] - 0s 2ms/step - loss: 13.0603 - accuracy: 0.6389
Epoch 3/100
18/18 [=====] - 0s 2ms/step - loss: 9.5961 - accuracy: 0.6024
Epoch 4/100
18/18 [=====] - 0s 2ms/step - loss: 7.7556 - accuracy: 0.6007
Epoch 5/100
18/18 [=====] - 0s 2ms/step - loss: 6.3340 - accuracy: 0.5781
Epoch 6/100
18/18 [=====] - 0s 2ms/step - loss: 5.1204 - accuracy: 0.5608
Epoch 7/100
18/18 [=====] - 0s 2ms/step - loss: 4.0866 - accuracy: 0.5451
Epoch 8/100
18/18 [=====] - 0s 2ms/step - loss: 3.0615 - accuracy: 0.5417
Epoch 9/100
18/18 [=====] - 0s 2ms/step - loss: 2.1218 - accuracy: 0.5451
Epoch 10/100
18/18 [=====] - 0s 2ms/step - loss: 1.4289 - accuracy: 0.5868
Epoch 11/100
18/18 [=====] - 0s 2ms/step - loss: 1.0858 - accuracy: 0.5990
Epoch 12/100
18/18 [=====] - 0s 2ms/step - loss: 0.9967 - accuracy: 0.6389
Epoch 13/100
18/18 [=====] - 0s 2ms/step - loss: 0.8826 - accuracy: 0.6615
Epoch 14/100
18/18 [=====] - 0s 2ms/step - loss: 0.8023 - accuracy: 0.6684
```

+ Code + Text

6s



```
18/18 [=====] - 0s 3ms/step - loss: 0.5731 - accuracy: 0.7344
Epoch 95/100
18/18 [=====] - 0s 3ms/step - loss: 0.6055 - accuracy: 0.7066
Epoch 96/100
18/18 [=====] - 0s 2ms/step - loss: 0.5670 - accuracy: 0.7101
Epoch 97/100
18/18 [=====] - 0s 2ms/step - loss: 0.5554 - accuracy: 0.7188
Epoch 98/100
18/18 [=====] - 0s 2ms/step - loss: 0.5633 - accuracy: 0.7222
Epoch 99/100
18/18 [=====] - 0s 2ms/step - loss: 0.5734 - accuracy: 0.7118
Epoch 100/100
18/18 [=====] - 0s 2ms/step - loss: 0.6140 - accuracy: 0.6944
Model Summary:
Model: "sequential_3"
```

| Layer (type) | Output Shape | Param # |
|------------------|--------------|---------|
| dense_10 (Dense) | (None, 20) | 180 |
| dense_11 (Dense) | (None, 1) | 21 |

```
=====
Total params: 201 (804.00 Byte)
Trainable params: 201 (804.00 Byte)
Non-trainable params: 0 (0.00 Byte)
```

```
None
6/6 [=====] - 0s 3ms/step - loss: 0.6438 - accuracy: 0.6823
```

```
Evaluation Result (loss, accuracy): [0.643755551528931, 0.6822916865348816]
```

a. Add more Dense layers to the existing code and check how the accuracy changes.

NNDL_ICP_6.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

5s [3] import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
import numpy as np

Load dataset
dataset = pd.read_csv("diabetes.csv", header=None).values

X_train, X_test, Y_train, Y_test = train_test_split(dataset[:,0:8], dataset[:,8],
test_size=0.25, random_state=87)
np.random.seed(155)

sequential_model = Sequential() # create model
sequential_model.add(Dense(20, input_dim=8, activation='relu'))
sequential_model.add(Dense(10, activation='relu')) # Additional hidden layer
sequential_model.add(Dense(5, activation='relu')) # Additional hidden layer
sequential_model.add(Dense(1, activation='sigmoid')) # output layer
sequential_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = sequential_model.fit(X_train, Y_train, epochs=100,
initial_epoch=0, verbose=1)

print("Model Summary: ")
print(sequential_model.summary())

evaluation_results = sequential_model.evaluate(X_test, y_test)
print("\nEvaluation Result (loss, accuracy):", evaluation_results)

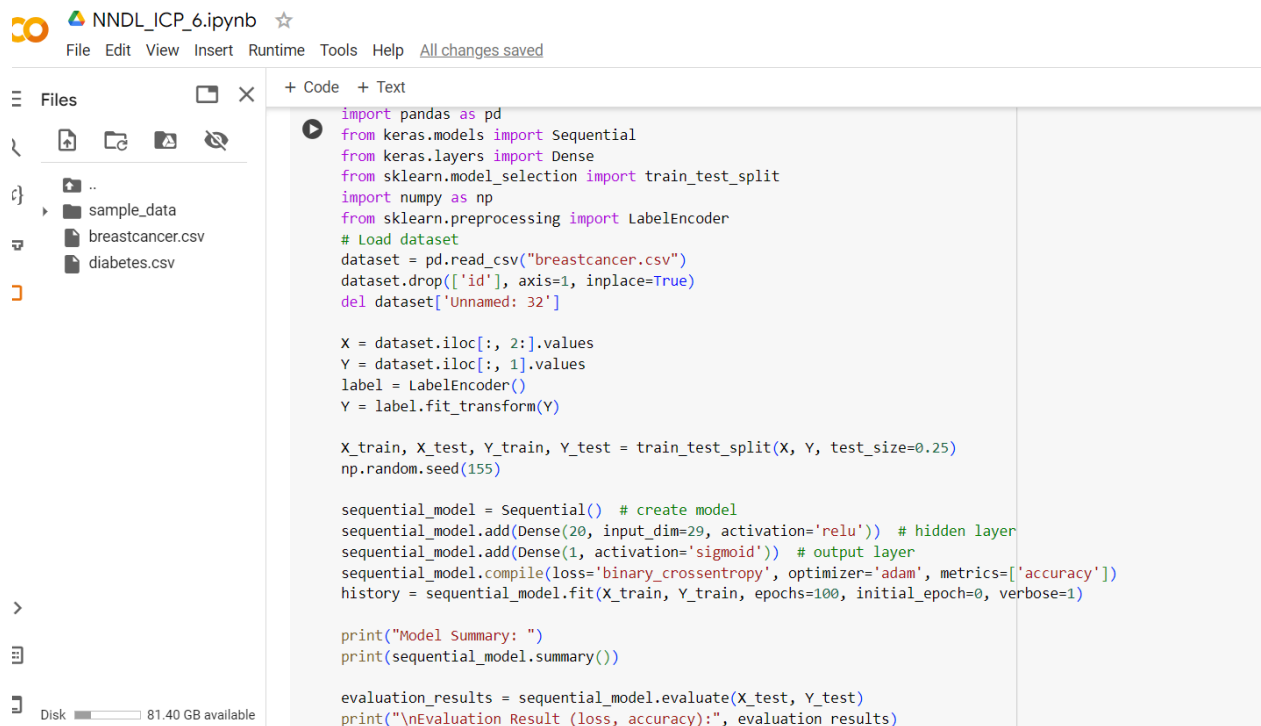
18/18 [=====] - 0s 2ms/step - loss: 0.5668 - accuracy: 0.7101
Epoch 97/100
18/18 [=====] - 0s 2ms/step - loss: 0.5674 - accuracy: 0.7101
Epoch 98/100
18/18 [=====] - 0s 2ms/step - loss: 0.5670 - accuracy: 0.7118
Epoch 99/100
18/18 [=====] - 0s 2ms/step - loss: 0.5666 - accuracy: 0.7153
Epoch 100/100
18/18 [=====] - 0s 2ms/step - loss: 0.5663 - accuracy: 0.7222
Model Summary:
Model: "sequential_1"

| Layer (type) | Output Shape | Param # |
|-----------------|--------------|---------|
| dense_2 (Dense) | (None, 20) | 180 |
| dense_3 (Dense) | (None, 10) | 210 |
| dense_4 (Dense) | (None, 5) | 55 |
| dense_5 (Dense) | (None, 1) | 6 |

=====
Total params: 451 (1.76 KB)
Trainable params: 451 (1.76 KB)
Non-trainable params: 0 (0.00 Byte)

None
6/6 [=====] - 0s 3ms/step - loss: 0.6230 - accuracy: 0.6458
Evaluation Result (loss, accuracy): [0.6229787468910217, 0.6458333134651184]

2. Change the data source to Breast Cancer dataset * available in the source code folder and make required changes. Report accuracy of the model.



NNDL_ICP_6.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Files

- sample_data
- breastcancer.csv
- diabetes.csv

```
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.preprocessing import LabelEncoder

# Load dataset
dataset = pd.read_csv("breastcancer.csv")
dataset.drop(['id'], axis=1, inplace=True)
del dataset['Unnamed: 32']

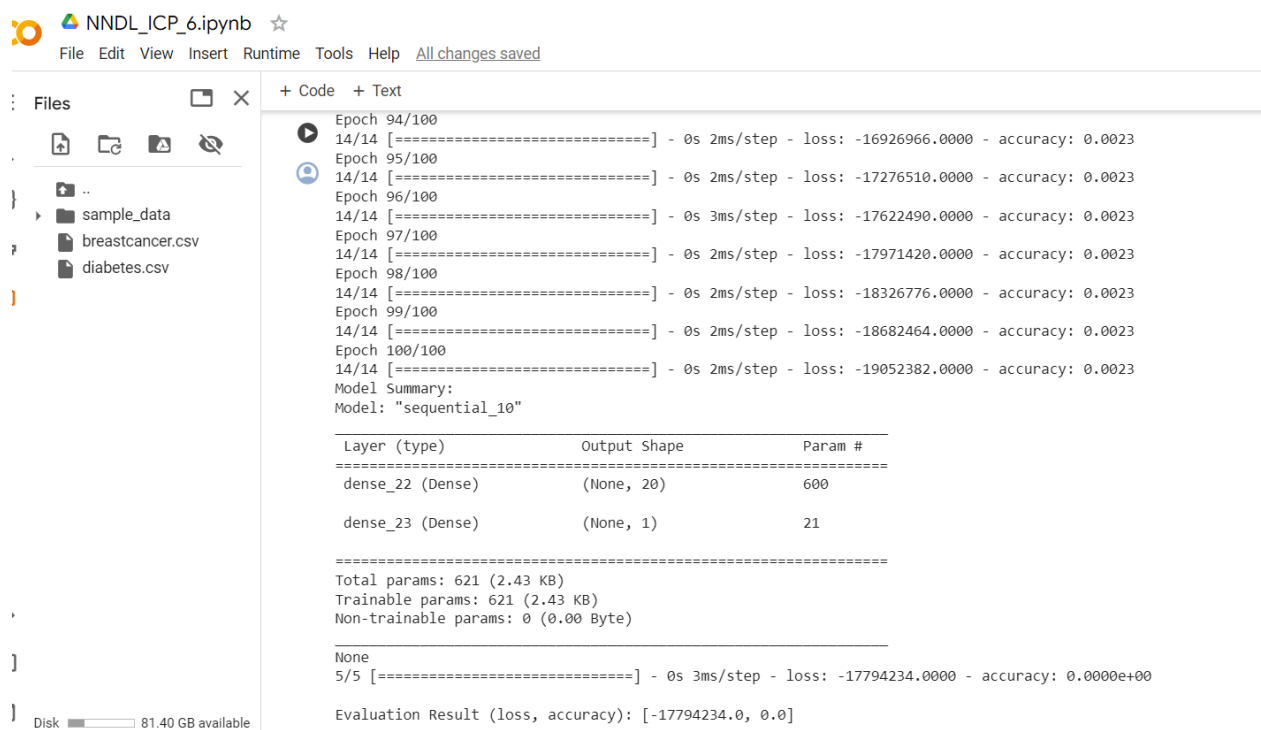
X = dataset.iloc[:, 2:].values
Y = dataset.iloc[:, 1].values
label = LabelEncoder()
Y = label.fit_transform(Y)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25)
np.random.seed(155)

sequential_model = Sequential() # create model
sequential_model.add(Dense(20, input_dim=29, activation='relu')) # hidden layer
sequential_model.add(Dense(1, activation='sigmoid')) # output layer
sequential_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
history = sequential_model.fit(X_train, Y_train, epochs=100, initial_epoch=0, verbose=1)

print("Model Summary: ")
print(sequential_model.summary())

evaluation_results = sequential_model.evaluate(X_test, Y_test)
print("\nEvaluation Result (loss, accuracy):", evaluation_results)
```



NNDL_ICP_6.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Files

- sample_data
- breastcancer.csv
- diabetes.csv

```
Epoch 94/100
14/14 [=====] - 0s 2ms/step - loss: -16926966.0000 - accuracy: 0.0023
Epoch 95/100
14/14 [=====] - 0s 2ms/step - loss: -17276510.0000 - accuracy: 0.0023
Epoch 96/100
14/14 [=====] - 0s 3ms/step - loss: -17622490.0000 - accuracy: 0.0023
Epoch 97/100
14/14 [=====] - 0s 2ms/step - loss: -17971420.0000 - accuracy: 0.0023
Epoch 98/100
14/14 [=====] - 0s 2ms/step - loss: -18326776.0000 - accuracy: 0.0023
Epoch 99/100
14/14 [=====] - 0s 2ms/step - loss: -18682464.0000 - accuracy: 0.0023
Epoch 100/100
14/14 [=====] - 0s 2ms/step - loss: -19052382.0000 - accuracy: 0.0023
Model Summary:
Model: "sequential_10"

Layer (type)                 Output Shape              Param #
=====
dense_22 (Dense)              (None, 20)                600
dense_23 (Dense)              (None, 1)                 21
=====
Total params: 621 (2.43 KB)
Trainable params: 621 (2.43 KB)
Non-trainable params: 0 (0.00 Byte)

None
5/5 [=====] - 0s 3ms/step - loss: -17794234.0000 - accuracy: 0.0000e+00

Evaluation Result (loss, accuracy): [-17794234.0, 0.0]
```

3. Normalize the data before feeding the data to the model and check how the normalization change your accuracy (code given below).

from sklearn.preprocessing

import StandardScaler sc = StandardScaler()

NNDL_ICP_6.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

files

- ..
- sample_data
- breastcancer.csv
- diabetes.csv

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

# Load dataset
dataset = pd.read_csv("breastcancer.csv")
dataset.drop(columns=['id', 'Unnamed: 32'], inplace=True)

X = dataset.iloc[:, 2:].values
Y = LabelEncoder().fit_transform(dataset.iloc[:, 1])

# Splitting data into train and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=155)

# Standardize features
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Define model
sequential_model = Sequential()
sequential_model.add(Dense(20, input_dim=29, activation='relu')) # Hidden layer
sequential_model.add(Dense(1, activation='sigmoid')) # Output layer
sequential_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train model
my_first_nn_fitted = sequential_model.fit(X_train, Y_train, epochs=100, initial_epoch=0, verbose=1)

# Print model summary
print(sequential_model.summary())

# Evaluate model
evaluation_results = sequential_model.evaluate(X_test, Y_test)
print("\nEvaluation Result (loss, accuracy):", evaluation_results)
```

Disk 81.40 GB available

Files

- ..
- sample_data
- breastcancer.csv
- diabetes.csv

```
14/14 [=====] - 0s 3ms/step - loss: -89035.1016 - accuracy: 0.0023
Epoch 96/100
14/14 [=====] - 0s 3ms/step - loss: -90989.6797 - accuracy: 0.0023
Epoch 97/100
14/14 [=====] - 0s 3ms/step - loss: -92927.7109 - accuracy: 0.0023
Epoch 98/100
14/14 [=====] - 0s 3ms/step - loss: -94873.2266 - accuracy: 0.0023
Epoch 99/100
14/14 [=====] - 0s 3ms/step - loss: -96780.8438 - accuracy: 0.0023
Epoch 100/100
14/14 [=====] - 0s 2ms/step - loss: -98832.9219 - accuracy: 0.0023
Model: "sequential_11"

Layer (type)           Output Shape           Param #
-----
dense_24 (Dense)        (None, 20)             600
dense_25 (Dense)        (None, 1)              21

Total params: 621 (2.43 KB)
Trainable params: 621 (2.43 KB)
Non-trainable params: 0 (0.00 Byte)

None
5/5 [=====] - 0s 4ms/step - loss: -94657.6094 - accuracy: 0.0000e+00

Evaluation Result (loss, accuracy): [-94657.609375, 0.0]
```

Note :Breast Cancer dataset is designated to predict if a patient has Malignant (M) or Benign = B cancer

In class programming:

Use Image Classification on the hand written digits data set (mnist)

NDL_ICP_6.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Files

sample_data

breastcancer.csv

diabetes.csv

Code

```
from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical

(train_images,train_labels),(test_images, test_labels) = mnist.load_data()

print(train_images.shape[1:])
#process the data
#1. convert each image of shape 28*28 to 784 dimensional which will be fed to the network as a single feature
dimData = np.prod(train_images.shape[1:])
print(dimData)
train_data = train_images.reshape(train_images.shape[0],dimData)
test_data = test_images.reshape(test_images.shape[0],dimData)

#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')
#scale data
train_data /=255.0
test_data /=255.0
#change the labels from integer to one-hot encoding. to_categorical is doing the same thing as LabelEncoder()
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)
```

Text

```
#creating network
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1,
                    validation_data=(test_data, test_labels_one_hot))
```

Output

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [=====] - 0s 0us/step
(28, 28)
784
Epoch 1/10
235/235 [=====] - 14s 56ms/step - loss: 0.2885 - accuracy: 0.9125 - val_loss: 0.1193 - val_accuracy: 0.9606
Epoch 2/10
235/235 [=====] - 11s 48ms/step - loss: 0.1008 - accuracy: 0.9686 - val_loss: 0.0928 - val_accuracy: 0.9717
Epoch 3/10
235/235 [=====] - 8s 33ms/step - loss: 0.0627 - accuracy: 0.9806 - val_loss: 0.0905 - val_accuracy: 0.9709
Epoch 4/10
235/235 [=====] - 6s 25ms/step - loss: 0.0434 - accuracy: 0.9864 - val_loss: 0.0602 - val_accuracy: 0.9812
Epoch 5/10
235/235 [=====] - 6s 24ms/step - loss: 0.0318 - accuracy: 0.9900 - val_loss: 0.0673 - val_accuracy: 0.9808
Epoch 6/10
235/235 [=====] - 6s 24ms/step - loss: 0.0221 - accuracy: 0.9930 - val_loss: 0.0661 - val_accuracy: 0.9811
Epoch 7/10
235/235 [=====] - 6s 27ms/step - loss: 0.0162 - accuracy: 0.9948 - val_loss: 0.0634 - val_accuracy: 0.9818
Epoch 8/10
235/235 [=====] - 5s 23ms/step - loss: 0.0125 - accuracy: 0.9960 - val_loss: 0.0748 - val_accuracy: 0.9801
Epoch 9/10
235/235 [=====] - 6s 28ms/step - loss: 0.0092 - accuracy: 0.9975 - val_loss: 0.0808 - val_accuracy: 0.9788
Epoch 10/10
235/235 [=====] - 5s 23ms/step - loss: 0.0071 - accuracy: 0.9976 - val_loss: 0.0890 - val_accuracy: 0.9767
```

1. Plot the loss and accuracy for both training data and validation data using the history object in the source code.

NNDL_ICP_6.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Files

- sample_data
- breastcancer.csv
- diabetes.csv

```
from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

print(train_images.shape[1:])
#process the data
dimData = np.prod(train_images.shape[1:])
print(dimData)
train_data = train_images.reshape(train_images.shape[0], dimData)
test_data = test_images.reshape(test_images.shape[0], dimData)

#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')
#scale data
train_data /= 255.0
test_data /= 255.0
#change the labels from integer to one-hot encoding. to_categorical is doing the same thing as LabelEncoder()
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

#creating network
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

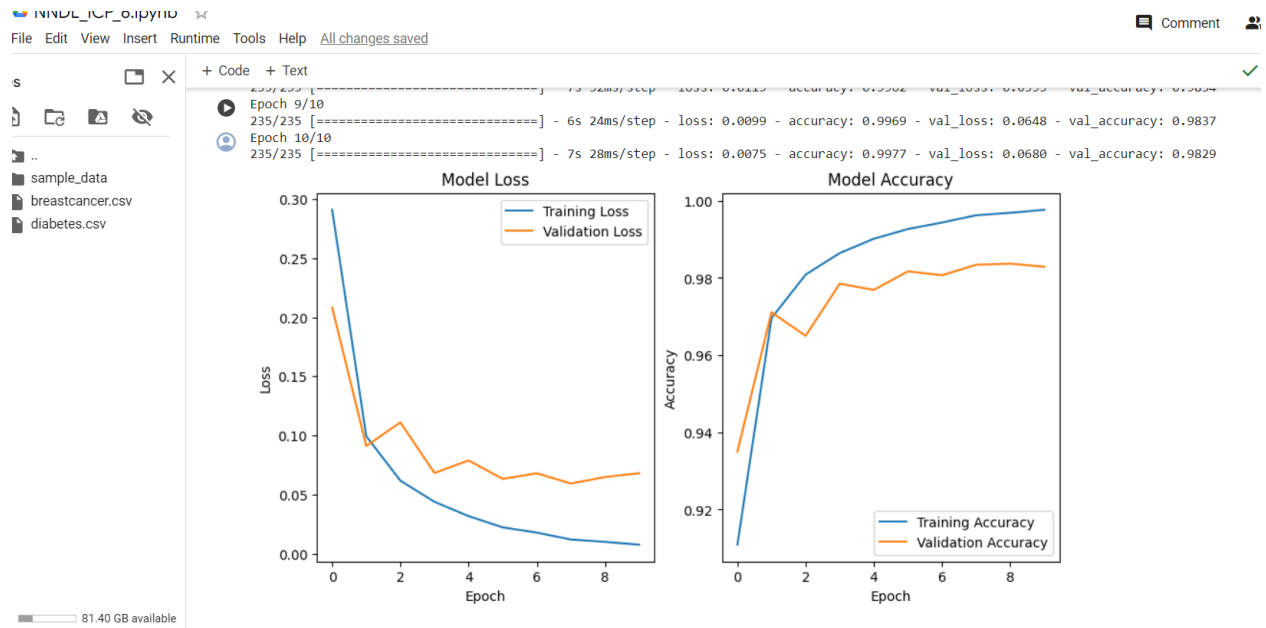
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1,
                    validation_data=(test_data, test_labels_one_hot))

import matplotlib.pyplot as plt

# Plot training & validation loss values
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
```

(28, 28)
784
Epoch 1/10
235/235 [=====] - 6s 25ms/step - loss: 0.2910 - accuracy: 0.9109 - val_loss: 0.2084 - val_accuracy: 0.9350
Epoch 2/10
235/235 [=====] - 6s 26ms/step - loss: 0.0994 - accuracy: 0.9696 - val_loss: 0.0911 - val_accuracy: 0.9711
Epoch 3/10
235/235 [=====] - 6s 25ms/step - loss: 0.0617 - accuracy: 0.9808 - val_loss: 0.1111 - val_accuracy: 0.9650
Epoch 4/10
235/235 [=====] - 6s 24ms/step - loss: 0.0438 - accuracy: 0.9864 - val_loss: 0.0683 - val_accuracy: 0.9785
Epoch 5/10
235/235 [=====] - 5s 22ms/step - loss: 0.0317 - accuracy: 0.9901 - val_loss: 0.0788 - val_accuracy: 0.9769
Epoch 6/10
235/235 [=====] - 6s 27ms/step - loss: 0.0222 - accuracy: 0.9927 - val_loss: 0.0632 - val_accuracy: 0.9817
Epoch 7/10
235/235 [=====] - 5s 23ms/step - loss: 0.0177 - accuracy: 0.9944 - val_loss: 0.0680 - val_accuracy: 0.9807
Epoch 8/10
235/235 [=====] - 7s 32ms/step - loss: 0.0119 - accuracy: 0.9962 - val_loss: 0.0593 - val_accuracy: 0.9834
Epoch 9/10
235/235 [=====] - 6s 24ms/step - loss: 0.0099 - accuracy: 0.9969 - val_loss: 0.0648 - val_accuracy: 0.9837
Epoch 10/10

Disk 81.40 GB available



2. Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model on that single image.

NNDL_ICP_6.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment

Files

sample_data
breastcancer.csv
diabetes.csv

```
from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

print(train_images.shape[1:])
#process the data
#1. convert each image of shape 28*28 to 784 dimensional which will be fed to the network as a single feature
dimData = np.prod(train_images.shape[1:])
print(dimData)
train_data = train_images.reshape(train_images.shape[0], dimData)
test_data = test_images.reshape(test_images.shape[0], dimData)

#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')
#scale data
train_data /= 255.0
test_data /= 255.0
#change the labels from integer to one-hot encoding. to_categorical is doing the same thing as LabelEncoder()
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)
```


NNDL_ICP_6.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment
SI

Files

+

..

+

sample_data

+

breastcancer.csv

+

diabetes.csv

+

Code

+

Text

```

#creating network
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1,
                    validation_data=(test_data, test_labels_one_hot))

import matplotlib.pyplot as plt

# Plot one image from the test data
plt.figure()
plt.imshow(test_images[0], cmap='gray')
plt.title(f"True Label: {test_labels[0]}")

# Perform inference on the single image
image = test_data[0].reshape(1, dimData)
prediction = model.predict(image)
predicted_label = np.argmax(prediction)

print(f"Predicted Label: {predicted_label}")

```

(28, 28)

784

Epoch 1/10

235/235 [=====] - 7s 26ms/step - loss: 0.2913 - accuracy: 0.9108 - val_loss: 0.1687 - val_accuracy: 0.9458

Epoch 2/10

Disk

81.40 GB available

NNDL_ICP_6.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment
Share

Files

+

..

+

sample_data

+

breastcancer.csv

+

diabetes.csv

+

Code

+

Text

```

235/235 [=====] - 0s 27ms/step - loss: 0.0114 - accuracy: 0.9997 - val_loss: 0.0712 - val_accuracy: 0.9800
Epoch 9/10
235/235 [=====] - 5s 23ms/step - loss: 0.0100 - accuracy: 0.9968 - val_loss: 0.0679 - val_accuracy: 0.9833
Epoch 10/10
235/235 [=====] - 6s 27ms/step - loss: 0.0073 - accuracy: 0.9977 - val_loss: 0.0656 - val_accuracy: 0.9820
1/1 [=====] - 0s 90ms/step
Predicted Label: 7

```

True Label: 7

RAM

Disk

81.40 GB available

3. We had used 2 hidden layers and Relu activation. Try to change the number of hidden layer and the activation to tanh or sigmoid and see what happens.

NNDL_ICP_6.ipynb ☆

File Edit View Insert Runtime Tools Help All changes saved

Comment

s

+

Code

+

Text

52s

```
from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

print(train_images.shape[1:])
#process the data
#1. convert each image of shape 28*28 to 784 dimensional which will be fed to the network as a single feature
dimData = np.prod(train_images.shape[1:])
print(dimData)
train_data = train_images.reshape(train_images.shape[0], dimData)
test_data = test_images.reshape(test_images.shape[0], dimData)

#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')
#scale data
train_data /= 255.0
test_data /= 255.0
#change the labels from integer to one-hot encoding. to_categorical is doing the same thing as LabelEncoder()
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

#creating network
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(dimData,))) # Change activation to 'tanh'
model.add(Dense(256, activation='tanh')) # Add another hidden layer with 'tanh'
model.add(Dense(128, activation='tanh')) # Add one more hidden layer with 'tanh'
model.add(Dense(10, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1,
                    validation_data=(test_data, test_labels_one_hot))
```

sample_data

breastcancer.csv

diabetes.csv

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>

11490434/11490434 [=====] - 0s 0us/step

(28, 28)

784

Epoch 1/10

235/235 [=====] - 5s 21ms/step - loss: 0.3397 - accuracy: 0.8967 - val_loss: 0.1980 - val_accuracy: 0.9384

Epoch 2/10

235/235 [=====] - 7s 28ms/step - loss: 0.1482 - accuracy: 0.9547 - val_loss: 0.1653 - val_accuracy: 0.9478

Epoch 3/10

235/235 [=====] - 4s 19ms/step - loss: 0.0995 - accuracy: 0.9700 - val_loss: 0.1383 - val_accuracy: 0.9558

Epoch 4/10

235/235 [=====] - 5s 23ms/step - loss: 0.0727 - accuracy: 0.9777 - val_loss: 0.0831 - val_accuracy: 0.9737

Epoch 5/10

235/235 [=====] - 5s 20ms/step - loss: 0.0540 - accuracy: 0.9827 - val_loss: 0.0936 - val_accuracy: 0.9702

Epoch 6/10

235/235 [=====] - 4s 19ms/step - loss: 0.0416 - accuracy: 0.9876 - val_loss: 0.0820 - val_accuracy: 0.9736

Epoch 7/10

235/235 [=====] - 5s 23ms/step - loss: 0.0309 - accuracy: 0.9911 - val_loss: 0.0907 - val_accuracy: 0.9707

Epoch 8/10

235/235 [=====] - 4s 19ms/step - loss: 0.0240 - accuracy: 0.9925 - val_loss: 0.0680 - val_accuracy: 0.9784

Epoch 9/10

235/235 [=====] - 5s 21ms/step - loss: 0.0180 - accuracy: 0.9948 - val_loss: 0.0797 - val_accuracy: 0.9767

Epoch 10/10

235/235 [=====] - 5s 21ms/step - loss: 0.0125 - accuracy: 0.9969 - val_loss: 0.0923 - val_accuracy: 0.9762

4. Run the same code without scaling the images and check the performance?

File Edit View Insert Runtime Tools Help All changes saved

Files

- sample_data
- breastcancer.csv
- diabetes.csv

```
from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

print(train_images.shape[1:])
#process the data
#1. convert each image of shape 28*28 to 784 dimensional which will be fed to the network as a single feature
dimData = np.prod(train_images.shape[1:])
print(dimData)
train_data = train_images.reshape(train_images.shape[0], dimData)
test_data = test_images.reshape(test_images.shape[0], dimData)

#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')
#scale data
#train_data /=255.0
#test_data /=255.0
#change the labels from integer to one-hot encoding. to_categorical is doing the same thing as LabelEncoder()
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

#creating network
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1,
                    validation_data=(test_data, test_labels_one_hot))
```

(28, 28)
784

Epoch 1/10
235/235 [=====] - 6s 23ms/step - loss: 6.4012 - accuracy: 0.8754 - val_loss: 0.6357 - val_accuracy: 0.9178

Epoch 2/10
235/235 [=====] - 6s 24ms/step - loss: 0.3976 - accuracy: 0.9432 - val_loss: 0.3577 - val_accuracy: 0.9403

Epoch 3/10
235/235 [=====] - 6s 23ms/step - loss: 0.2422 - accuracy: 0.9575 - val_loss: 0.3688 - val_accuracy: 0.9349

Epoch 4/10
235/235 [=====] - 5s 22ms/step - loss: 0.2041 - accuracy: 0.9652 - val_loss: 0.2314 - val_accuracy: 0.9657

Epoch 5/10
235/235 [=====] - 6s 26ms/step - loss: 0.1645 - accuracy: 0.9723 - val_loss: 0.2120 - val_accuracy: 0.9673

Epoch 6/10
235/235 [=====] - 5s 22ms/step - loss: 0.1542 - accuracy: 0.9744 - val_loss: 0.2623 - val_accuracy: 0.9680

Epoch 7/10
235/235 [=====] - 6s 26ms/step - loss: 0.1382 - accuracy: 0.9775 - val_loss: 0.3868 - val_accuracy: 0.9615

Epoch 8/10
235/235 [=====] - 5s 22ms/step - loss: 0.1309 - accuracy: 0.9799 - val_loss: 0.3064 - val_accuracy: 0.9651

Epoch 9/10
235/235 [=====] - 6s 26ms/step - loss: 0.1170 - accuracy: 0.9825 - val_loss: 0.5044 - val_accuracy: 0.9626

Epoch 10/10
235/235 [=====] - 5s 21ms/step - loss: 0.1306 - accuracy: 0.9841 - val_loss: 0.3639 - val_accuracy: 0.9684