(Peed Rorwodod)

Experiment - 2

* TensorFlow - TensorFlow is an open source free library use for neural network.

* Keras - build and train neural network.
  - has been integrated into tensorflow.
- commonly used for.
  - Image classification (using CNN)
  - sequence analysis & NLP.
  - Generative models
  - more complex task in AI research

* Pandas - data manipulation & analysis
* Numpy → for numerical computing in python.
* Matplotlib - is a library for creating static animated and interactive visualizations in python.
* Random - The random module in python provides tools for generating random numbers & performing random operations.
  - Features
    - Random numbers
    - Random choices
    -

* MNIST - stands for modified National institute of standards & technology.
- dataset of 70,000 handwritten images.
- 28x28 pixel. about 784 features. each feature represents only one pixel intensity. from 0(black) to 255(white)
- 60,000 training.
  10,000 testing

M T W T F S S
Page No.:
Date:
YOUVA

1) mniot = tf.keras.dataset.mnist
↳ this dataset is already present in keras and then we load it into mniot.

2) (x_train, y_train) (x_test, y_test) = mniot_load_data()
splitting the data into training and testing after loading → saved here.

3) x_train shape
⇒ (60,000, 28, 28) → it means there are 60,000 images for training and these size is 28×28 pixel.

4) x_test.shape (same)
(10,000, 28, 28)

5) x_train[0] → shows the first image
- showing the image of matrix with intensity 0 to 255
= 28×28 with 784 features each represent intensity.

6) plt.matshow(x_train[0])
- To visualize the image.

7) x_train = x_train / 255
x_test = x_test / 255
} normalize the image by dividing maximum intensity

8) model = keras.sequential([
keras.layers.Flatten(input_shape=(28,28)), → input layer
keras.layers.Dense(128, activation='relu'), → Dense layer
Dense(10, activation='softmax')]) → output layer
Flatten - 2D to 1D.
Dense - each neuron in the previous layer is connected with next layer.
10 → no of classes.
sequential method it allows us to doing a model layer by layer.

Preprocessing.

preprocessing involves transforming raw data into structured format suitable for training a neural network.

common techniques-

1) Data cleaning - Remove & handle missing values.
2) Data transformation :- converting data types. encoding categorical variable.
3) Feature scaling - scaling features so they have similar ranges.
4) Tokenization - convert text into tokens. (words subwords, or char).
5) Image augmentation- rotation, flipping cropping.

Normalization.

Normalization is a way of adjusting data so it's on a similar scale or within a specific range like 0 or 1.

Types -

1) min-max scaling
2) z-score scaling
3) l2 normalization.

RELU (Rectified linear unit) = blocks negative

is used to help the model decide when to activate certain neurons in a neural network.
- If the input value is +ve relu keeps as it is and if -ve it changes to zero

softmax - used in final layer of classification model. especially when there are multiple classes.
- converts outputs into probabilities that represent the model's confidence for each

- model.summary().
- with the help of this we can see how the model is Generated

o compile the model.

model.compile(optimizer='sgd',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

loss function we used it saves time, it uses single int rather than whole vector

used to determine accuracy we used accuracy.

sgd - Sochastic gradient decrent optimizer - control the learning rate

o Train the model - with help of fit method.

history = model.fit(x_train, y_train, validation_data = (x_test, y_test), epochs=10)

The epochs means the no of time the model circulate the data.

o Evaluate the model. This method compares the loss

test_loss, test_acc = model.evaluate(x_test, y_test)
print("loss = %.3f" % test_loss)
print("Accuracy = %.3f" % test_acc)

o making Predicts on new data.
n = random.radiant(0,9999) → takes a random value
plot.imshow(x_test[n]) → It shows
plt.show()

// checks the performance. → do the prediction on testing

predicted_value = model.predict(x_test)

print("_____ %d" % np.argmax(predicted_value[n]))

↓

It returns maximum value

* plot graph for Accuracy and loss.
  in this we use history in that all the accuracy loss values it contains

history.history.keys()

→ It shows the dictionary like which keys are present.

o/p → dict_keys['loss', 'accuracy', 'val_loss', 'val_accuracy'])

- loss → Training loss
- accuracy → Training loss
- val_loss → validation data loss
- val_accuracy → Testing data.

Then plot graph for Train & Test.

Page No.:
Date:
YOUVA

Experiment No :- 3

* image classification :-
  - load the image data
  - Defining model architecture
  - Train model.
  - Estimate model performance

* Deep learning -
  Deep learning is a type of AI and ML that teaches computer to recognize patterns in a way that mimics the human brain. It uses structure called neural networks. To learn from large amount of data.

* CNN :-
  A CNN is a special type of dl model designed to process & recognize images, videos and other visual data.
  - used in image recognition & processing.
    1) understanding the image as a Grid of pixels.
    2) using convolutions to detect features.
    3) layers of CNN
    4) Building up complexity.
    5) making predictions.
  - CNN are particularly good at handling visual data because of their ability to focus on key features.

Types of layers :-

1) Convolutional layer :-
- core layer of CNN where model starts and analyzing the image.
- in this layer small filters or kernels scan over the image to detect patterns each filter look for specific features. such as edges textures, or colors.
- convolution layer transform i/p data by using a patch or locally connecting neurons from previous layer.

2) Activation layer :-
- After convolutional layer an activation function is applied to add non-linearity to the model.
- The most common act. func is ReLU.
ReLU simply replaces any -ve value with 0, while keeping +ve values same.

Ex :-
If a part of image doesn't have the feature ReLU will make that region zero, reducing noise.

3) Pooling layer :-
- Pooling layers reduce the size of feature maps. making the model faster & reducing the amount of information to process.
- most common type is Max Pooling, where the layer divides the image into small squares & keeps only the largest value in each square. This keeps imp feature.

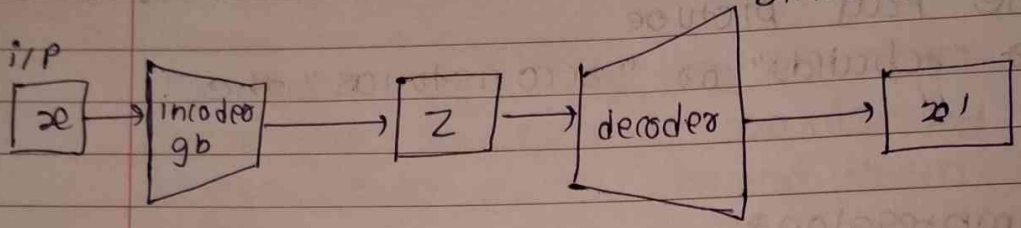
Page No.: YOUVA
Date:

Experiment No :- 4

* Autoencoder :-
- is a type of ANN used in unsupervised learning tasks, primarily for data compression and feature learning.

* Applications :- • Data compression
• Image Denoising
• Anomaly detection.
• Dimensionality Reduction.

- Structure :-



Types - 1) Denoising Autoencoder.
2) Sparse autoencoder
3) Variational autoencoder
4) Convolutional autoencoder.

* encoder-
encoder as a compressor for your data. It takes the original information (like an image or text) "squeezes" it down, keeping only the most important ~~data~~ parts.
- It reduces the size of the data step-by-step until it's in a much smaller room.

* Encoder :-
It's like a summary or a small "code" that holds the essence of the original data
Because it's ~~too~~ so small. it can't hold all the details so it has to focus on what's most important.
- Focuses the network to learn the key features of the data without all the extra information.

4) Fully Connected layer :-

- After the convolutional and pooling layer. the CNN has detected various features of the image Now, a fully connected layer uses this information to make a decision like identifying the object in the image.

exe :- If the CNN has been trained to detect animals : the layer use features like 'paws' 'fur' to decide if the image shows a cat or a dog.

5) Output layer :-
- provides the final prediction

* adam optimizer.
adaptive movement estimation is an algorithm used in training machine learning models especially in DL. It combines 2 technique. Momentum, Adaptive learning rate.
* optimizer - minimize loss function. which measures difference between the model prediction & actual outcomes

CBOW Model :-

1. data preparation
2. Generate training data.
3. Train model.
4. output.

CBOW teaches computer to guess missing word based on context

CBOW model :- (Contious Bag of words)

The CBOW model is a way of computers to learn the meaning of words by looking at the other words around them. It's like teaching a computer that words that show up together a lot have some connection.

How it works?

— sen - "The cat sat on the mat"

1) choose a target word - choose one word to guess. Let's say we use "sat"

2) look at the words around it
"The", "cat", "on" and "the"

3) Predict the target word
The CBOW model learns to use the words around a missing word to guess what the missing word should be.

4) Repeat many Times - computer does this over & over with lots of sentences. Each time it gets better understanding which words usually appear together & what they mean

M T W T F S S
Page No.:
Date:
YOUVA

* decoder :-

-The decoders job is to take that small compression version & try to expand it back to look like the original data.
-It's like reversing the compression to get back the full picture
-decoder "rebuilds" or "reconstructs" the data.

* data compression :-
Reducing data dimensionals without loosing essential information.

* Anomaly Detection :- process of finding anomalies in data.
- Anomaly detection is about finding things that are unusual or don't fit in rest of data.

-Imagine you're looking at a bunch of picture of dogs and suddenly you see cat that cat feature is anomaly because it doesn't match pattern.

Functions :-
1) local outlier factor
2) Isolation forest
3) KNN
4) auto encoder

* import re (regular expression)
- which allows you to work with patterns in text.
- used for searching matching & manipulating
Functions :-
- search
- match
- replace

* sentences = data.split('.')
sentences.
// This line of code is using python's split()
method to break a string (data) into
smaller strings based on specified delimeter
(.)

[^A-Za-z0-9]+ → cleans up the sentences by
removing special character & leaving only
letters & numbers

* Tokenizer converts words into integer indices
which can then be used for training.

* import re (regular expression)
- which allows you to work with patterns in text.
- used for searching matching & manipulating
Functions :-
- search
- match
- replace

* sentences = data.split('.')
sentences.
// This line of code is using python's split()
method to break a string (data) into
smaller strings based on specified delimeter
(.)

[^A-Za-z0-9]+ → clean up the sentences by
removing special character & leaving only
letters & numbers

* Tokenizer converts words into integer indices
which can then be used for training.

[4]:
```
['Deep learning (also known as deep structured learning) is part of a broader
family of machine learning methods based on artificial neural networks with
representation learning',
 ' Learning can be supervised, semi-supervised or unsupervised',
 ' Deep-learning architectures such as deep neural networks, deep belief
networks, deep reinforcement learning, recurrent neural networks, convolutional
neural networks and Transformers have been applied to fields including computer
vision, speech recognition, natural language processing, machine translation,
bioinformatics, drug design, medical image analysis, climate science, material
inspection and board game programs, where they have produced results comparable
to and in some cases surpassing human expert performance',
 '']
```

[6]:
```
clean_sent=[]    // clean sentence will be stored
for sentence in sentences: //goes through each sen. in sentences in previous.
    if sentence=="": // if equal a empty string then skip the iteration & continue
        continue
    sentence = re.sub('[^A-Za-z0-9]+', ' ', (sentence)) // that is match or any characters
    sentence = re.sub(r'(?:^| )\w (?:$| )', ' ', (sentence)).strip()
    sentence = sentence.lower() → converts entire sentence in lowercase
    clean_sent.append(sentence)
                ↳ The clean sentence is added to clean_sent.
print(clean_sent)
```

that is not a letter. so num

↳ This line removes single letter word like 'a' 'I' 's' in sentence.

```
['deep learning also known as deep structured learning is part of a broader
family of machine learning methods based on artificial neural networks with
representation learning', 'learning can be supervised semi supervised or
unsupervised', 'deep learning architectures such as deep neural networks deep
belief networks deep reinforcement learning recurrent neural networks
convolutional neural networks and transformers have been applied to fields
including computer vision speech recognition natural language processing machine
translation bioinformatics drug design medical image analysis climate science
material inspection and board game programs where they have produced results
comparable to and in some cases surpassing human expert performance']
```

[7]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
#The Tokenizer is a utility that helps in converting text into a format␣
 ↪suitable for
#machine learning models. It will convert words into integer indices, which can␣
 ↪then be used for training.
```

// from keras library

[11]:
```
tokenizer = Tokenizer() // convert text into sequence of numbers
tokenizer.fit_on_texts(clean_sent) build the vocabulary of clean seq. of words.
sequences = tokenizer.texts_to_sequences(clean_sent) // converts the list of sentences
print(sequences)                                        into list of numerical seq.
```

```
[[2, 1, 12, 13, 6, 2, 14, 1, 15, 16, 7, 17, 18, 19, 7, 8, 1, 20, 21, 22, 23, 4,
3, 24, 25, 1], [1, 26, 27, 9, 28, 9, 29, 30], [2, 1, 31, 32, 6, 2, 4, 3, 2, 33,
```

```
3, 2, 34, 1, 35, 4, 3, 36, 4, 3, 5, 37, 10, 38, 39, 11, 40, 41, 42, 43, 44, 45,
46, 47, 48, 8, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 5, 60, 61, 62, 63,
64, 10, 65, 66, 67, 11, 5, 68, 69, 70, 71, 72, 73, 74]]
```

[15]:
```
index_to_word = {}
word_to_index = {}

for i, sequence in enumerate(sequences):
#     print(sequence)
    word_in_sentence = clean_sent[i].split()
#     print(word_in_sentence)
    for j, value in enumerate(sequence):
        index_to_word[value] = word_in_sentence[j]
        word_to_index[word_in_sentence[j]] = value

print(index_to_word, "\n")
print(word_to_index)
```

```
{2: 'deep', 1: 'learning', 12: 'also', 13: 'known', 6: 'as', 14: 'structured',
15: 'is', 16: 'part', 7: 'of', 17: 'a', 18: 'broader', 19: 'family', 8:
'machine', 20: 'methods', 21: 'based', 22: 'on', 23: 'artificial', 4: 'neural',
3: 'networks', 24: 'with', 25: 'representation', 26: 'can', 27: 'be', 9:
'supervised', 28: 'semi', 29: 'or', 30: 'unsupervised', 31: 'architectures', 32:
'such', 33: 'belief', 34: 'reinforcement', 35: 'recurrent', 36: 'convolutional',
5: 'and', 37: 'transformers', 10: 'have', 38: 'been', 39: 'applied', 11: 'to',
40: 'fields', 41: 'including', 42: 'computer', 43: 'vision', 44: 'speech', 45:
'recognition', 46: 'natural', 47: 'language', 48: 'processing', 49:
'translation', 50: 'bioinformatics', 51: 'drug', 52: 'design', 53: 'medical',
54: 'image', 55: 'analysis', 56: 'climate', 57: 'science', 58: 'material', 59:
'inspection', 60: 'board', 61: 'game', 62: 'programs', 63: 'where', 64: 'they',
65: 'produced', 66: 'results', 67: 'comparable', 68: 'in', 69: 'some', 70:
'cases', 71: 'surpassing', 72: 'human', 73: 'expert', 74: 'performance'}

{'deep': 2, 'learning': 1, 'also': 12, 'known': 13, 'as': 6, 'structured': 14,
'is': 15, 'part': 16, 'of': 7, 'a': 17, 'broader': 18, 'family': 19, 'machine':
8, 'methods': 20, 'based': 21, 'on': 22, 'artificial': 23, 'neural': 4,
'networks': 3, 'with': 24, 'representation': 25, 'can': 26, 'be': 27,
'supervised': 9, 'semi': 28, 'or': 29, 'unsupervised': 30, 'architectures': 31,
'such': 32, 'belief': 33, 'reinforcement': 34, 'recurrent': 35, 'convolutional':
36, 'and': 5, 'transformers': 37, 'have': 10, 'been': 38, 'applied': 39, 'to':
11, 'fields': 40, 'including': 41, 'computer': 42, 'vision': 43, 'speech': 44,
'recognition': 45, 'natural': 46, 'language': 47, 'processing': 48,
'translation': 49, 'bioinformatics': 50, 'drug': 51, 'design': 52, 'medical':
53, 'image': 54, 'analysis': 55, 'climate': 56, 'science': 57, 'material': 58,
'inspection': 59, 'board': 60, 'game': 61, 'programs': 62, 'where': 63, 'they':
64, 'produced': 65, 'results': 66, 'comparable': 67, 'in': 68, 'some': 69,
'cases': 70, 'surpassing': 71, 'human': 72, 'expert': 73, 'performance': 74}
```

[44]:
```
#this code segment prepares the context and target data for the Continuous Bag of Words (CBOW) model
vocab_size = len(tokenizer.word_index) + 1
#This line calculates the vocabulary size, which is the number of unique words in your dataset. tokenizer.word_index returns a dictionary of words mapped to their corresponding indices.
#The +1 accounts for the fact that indexing starts at 1 (as 0 is often reserved for padding in neural networks)
emb_size = 10
# emb_size is set to 10. This variable defines the size of the embedding vectors that will be used to represent each word.
# An embedding size of 10 means each word will be represented by a vector of 10 numbers.
context_size = 2
contexts = []
targets = []

for sequence in sequences:

    for i in range(context_size, len(sequence) - context_size):
        #This inner loop iterates through the indices of the current sequence, starting from context_size and ending at len(sequence) - context_size.
        #This ensures that the model has enough words on both sides of the target word to create a full context window.

        target = sequence[i]

        context = [sequence[i - 2], sequence[i - 1], sequence[i + 1], sequence[i + 2]]
#         print(context)
        contexts.append(context)
        targets.append(target)
print('context --> ',contexts, "\n")
print('targets --> ', targets)
```

Given no of unique words

contains the vocabulary that tokenizer built during fit on text.

is added because starts from 1

each word represent 10 dimensional vector

defines the number of words to the left & right means it is using 2 words left & 2 words right.

& empty lists.

skips the first & last context_size words.

loop will skip first 2 & last 2.

target is set to current word at index i

taking the two words before & two words after.

```
context -->  [[2, 1, 13, 6], [1, 12, 6, 2], [12, 13, 2, 14], [13, 6, 14, 1], [6,
2, 1, 15], [2, 14, 15, 16], [14, 1, 16, 7], [1, 15, 7, 17], [15, 16, 17, 18],
[16, 7, 18, 19], [7, 17, 19, 7], [17, 18, 7, 8], [18, 19, 8, 1], [19, 7, 1, 20],
[7, 8, 20, 21], [8, 1, 21, 22], [1, 20, 22, 23], [20, 21, 23, 4], [21, 22, 4,
3], [22, 23, 3, 24], [23, 4, 24, 25], [4, 3, 25, 1], [1, 26, 9, 28], [26, 27,
28, 9], [27, 9, 9, 29], [9, 28, 29, 30], [2, 1, 32, 6], [1, 31, 6, 2], [31, 32,
2, 4], [32, 6, 4, 3], [6, 2, 3, 2], [2, 4, 2, 33], [4, 3, 33, 3], [3, 2, 3, 2],
[2, 33, 2, 34], [33, 3, 34, 1], [3, 2, 1, 35], [2, 34, 35, 4], [34, 1, 4, 3],
[1, 35, 3, 36], [35, 4, 36, 4], [4, 3, 4, 3], [3, 36, 3, 5], [36, 4, 5, 37], [4,
3, 37, 10], [3, 5, 10, 38], [5, 37, 38, 39], [37, 10, 39, 11], [10, 38, 11, 40],
```

```
38, 39, 40, 41], [39, 11, 41, 42], [11, 40, 42, 43], [40, 41, 43, 44], [41, 42,
4, 45], [42, 43, 45, 46], [43, 44, 46, 47], [44, 45, 47, 48], [45, 46, 48, 8],
46, 47, 8, 49], [47, 48, 49, 50], [48, 8, 50, 51], [8, 49, 51, 52], [49, 50,
2, 53], [50, 51, 53, 54], [51, 52, 54, 55], [52, 53, 55, 56], [53, 54, 56, 57],
54, 55, 57, 58], [55, 56, 58, 59], [56, 57, 59, 5], [57, 58, 5, 60], [58, 59,
0, 61], [59, 5, 61, 62], [5, 60, 62, 63], [60, 61, 63, 64], [61, 62, 64, 10],
62, 63, 10, 65], [63, 64, 65, 66], [64, 10, 66, 67], [10, 65, 67, 11], [65, 66,
1, 5], [66, 67, 5, 68], [67, 11, 68, 69], [11, 5, 69, 70], [5, 68, 70, 71],
68, 69, 71, 72], [69, 70, 72, 73], [70, 71, 73, 74]]

argets -->  [12, 13, 6, 2, 14, 1, 15, 16, 7, 17, 18, 19, 7, 8, 1, 20, 21, 22,
3, 4, 3, 24, 27, 9, 28, 9, 31, 32, 6, 2, 4, 3, 2, 33, 3, 2, 34, 1, 35, 4, 3,
5, 4, 3, 5, 37, 10, 38, 39, 11, 40, 41, 42, 43, 44, 45, 46, 47, 48, 8, 49, 50,
, 52, 53, 54, 55, 56, 57, 58, 59, 5, 60, 61, 62, 63, 64, 10, 65, 66, 67, 11,
 68, 69, 70, 71, 72]
```

```
printing features with target
or i in range(5):
    words = []
    target = index_to_word.get(targets[i])
    for j in contexts[i]:
        words.append(index_to_word.get(j))
    print(words," --> ", target, '\n')
```

iterates over first 5 element or context & target.

→ retrives the word corresponding to target index.

→ retrives actual word from index_to word.

```
deep', 'learning', 'known', 'as']  -->  also

learning', 'also', 'as', 'deep']  -->  known

also', 'known', 'deep', 'structured']  -->  as

known', 'as', 'structured', 'learning']  -->  deep

as', 'deep', 'learning', 'is']  -->  structured
```

```
Convert the contexts and targets to numpy arrays
= np.array(contexts)
= np.array(targets)
shape, Y.shape

88, 4), (88,))
```

```
print(X)
print(Y)
```

```
port tensorflow as tf
om tensorflow.keras.models import Sequential
```

```
from tensorflow.keras.layers import Dense, Embedding, Lambda

# Sequential: This class is used to create a linear stack of layers for the␣
 ↪model.
# Dense: A fully connected layer used for the model.
# Embedding: A layer that turns positive integers (indexes) into dense vectors␣
 ↪of fixed size.
# Lambda: A layer that allows you to create custom operations in your model.
```

[66]:
```
# This initializes a sequential model, which is built layer by layer.
model = Sequential([  // Linear stack of layers meaning o/p of 1 layer is
                        used as the i/p to the next layer.
# This layer transforms the integer indices from the contexts into dense␣
 ↪vectors (embeddings).
# input_dim=vocab_size: The size of the input space (number of unique words).
# output_dim=emb_size: The size of the embedding vectors (10 in this case).
# input_length=2*context_size: The length of the input sequences (4 in this␣
 ↪case, as context_size is set to 2).
    Embedding(input_dim=vocab_size, output_dim=emb_size,␣
 ↪input_length=2*context_size),  → to map input indices.

# This layer computes the mean of the embedding vectors for each context (which␣
 ↪contains 4 words).
# tf.reduce_mean(x, axis=1) calculates the average across the embedding vectors␣
 ↪along the specified axis (in this case, the context dimension).
# The output will be a single vector for each input context, summarizing the␣
 ↪information from the four context words.
    Lambda(lambda x: tf.reduce_mean(x, axis=1)), → apply arbitary function to
                                                    data.

#This fully connected layer has 256 units and uses the ReLU (Rectified Linear␣
 ↪Unit) activation function.
#It introduces non-linearity to the model, allowing it to learn complex␣
 ↪relationships.  → Full connected layer with 256 neurons
    Dense(256, activation='relu'),
    Dense(512, activation='relu'),

# The output layer has a number of units equal to the vocabulary size and uses␣
 ↪the softmax activation function.
# The softmax function outputs a probability distribution over the vocabulary,␣
 ↪predicting the likelihood of each word being the target given the context.
    Dense(vocab_size, activation='softmax')  converts the o/p into probabilities
])
```

[67]:
```
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
```

M T W T F S S
Page No.:
Date:
YOUVA

Practical 6.

* VGG 16
is a popular CNN architecture developed by visual Geometry Group. at university oxford.
- it has 16 layer. with trainable weights (13 convolutional & 3 fully connected layer)

- used for image recognition.
- using 3x3 convolution filter.
  2x2 max pooling layer.

* model