```python
import numpy as np
import re
data = """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."""
data
```

{"type":"string"}

```python
sentences = data.split('.')
sentences
```

```
['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',
 '']
```

```python
clean_sent=[]
for sentence in sentences:
    if sentence=="":
        continue
    sentence = re.sub('[^A-Za-z0-9]+', ' ', (sentence))
    sentence = re.sub(r'(?:^| )\w (?:$| )', ' ', (sentence)).strip()
    sentence = sentence.lower()
    clean_sent.append(sentence)

clean_sent
```

```
['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']
```

```python
from tensorflow.keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer()
tokenizer.fit_on_texts(clean_sent)
sequences = tokenizer.texts_to_sequences(clean_sent)
print(sequences)
```

```
[[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]]
```

```python
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}
```

```
vocab_size = len(tokenizer.word_index) + 1
emb_size = 10
context_size = 2

contexts = []
targets = []

for sequence in sequences:
    for i in range(context_size, len(sequence) - context_size):
        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(contexts, "\n")
print(targets)
```

```
[[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, 44, 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, 52, 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, 60, 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, 11, 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]]

[12, 13, 6, 2, 14, 1, 15, 16, 7, 17, 18, 19, 7, 8, 1, 20, 21, 22, 23,
4, 3, 24, 27, 9, 28, 9, 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]

#printing features with target
for 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)

['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
X = np.array(contexts)
Y = np.array(targets)

# print(X)
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, Lambda


model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=emb_size,
input_length=2*context_size),
    Lambda(lambda x: tf.reduce_mean(x, axis=1)),
    Dense(256, activation='relu'),
    Dense(512, activation='relu'),
    Dense(vocab_size, activation='softmax')
])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/
embedding.py:90: UserWarning: Argument `input_length` is deprecated.
Just remove it.
  warnings.warn(
```

```python
model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])
history = model.fit(X, Y, epochs=80)
```

```
Epoch 1/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 2s 10ms/step - accuracy: 0.0153 - loss:
4.3179
Epoch 2/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.1250 - loss: 4.3107

Epoch 3/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.1367 - loss:
4.3032
Epoch 4/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.1229 - loss:
4.2937
Epoch 5/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.1424 - loss:
4.2811
Epoch 6/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.1346 - loss:
4.2633
Epoch 7/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.1250 - loss:
4.2315
Epoch 8/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.0984 - loss:
4.1933
Epoch 9/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.0614 - loss:
4.1613
Epoch 10/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.0479 - loss:
4.1071
Epoch 11/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.0714 - loss:
4.0318
Epoch 12/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.0714 - loss:
3.9418
Epoch 13/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.0653 - loss:
3.9138
Epoch 14/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.0810 - loss:
```

```
3.8550
Epoch 15/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.0884 - loss:
3.8797
Epoch 16/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.1381 - loss:
3.7917
Epoch 17/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.2077 - loss:
3.6504
Epoch 18/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.2347 - loss:
3.5840
Epoch 19/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.2290 - loss:
3.4785
Epoch 20/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.1726 - loss:
3.4513
Epoch 21/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.2560 - loss:
3.2866
Epoch 22/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.1957 - loss:
3.2825
Epoch 23/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.2308 - loss:
3.0998
Epoch 24/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - accuracy: 0.2422 - loss:
2.9991
Epoch 25/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.2887 - loss:
2.8387
Epoch 26/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.2827 - loss:
2.7666
Epoch 27/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.2631 - loss: 2.6647

Epoch 28/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.3075 - loss: 2.5138

Epoch 29/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.3675 - loss: 2.3812

Epoch 30/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.4496 - loss: 2.2577

Epoch 31/80
```

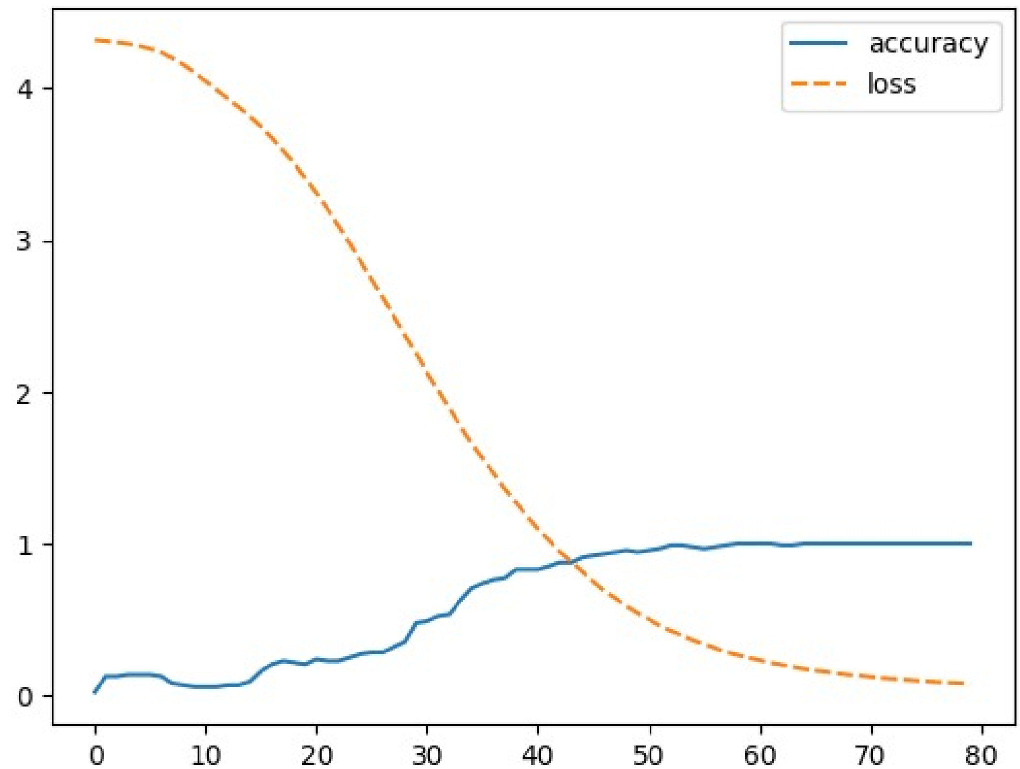
```
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.4982 - loss: 2.0802

Epoch 32/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.4996 - loss: 1.9774

Epoch 33/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5444 - loss: 1.8583

Epoch 34/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.5820 - loss: 1.8278

Epoch 35/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7312 - loss: 1.6245

Epoch 36/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7756 - loss: 1.5419

Epoch 37/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.7635 - loss: 1.4698

Epoch 38/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.7887 - loss: 1.3621

Epoch 39/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8054 - loss:
1.2670
Epoch 40/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8523 - loss: 1.1632

Epoch 41/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8445 - loss:
1.0594
Epoch 42/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.8558 - loss:
1.0118
Epoch 43/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.8750 - loss:
0.9623
Epoch 44/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.8672 - loss: 0.8789

Epoch 45/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9116 - loss: 0.8050

Epoch 46/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 0.9368 - loss: 0.7372

Epoch 47/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9308 - loss: 0.6830
```

```
Epoch 48/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9442 - loss: 0.6563

Epoch 49/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9577 - loss: 0.5726

Epoch 50/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9403 - loss: 0.5272

Epoch 51/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9734 - loss: 0.4887

Epoch 52/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9751 - loss: 0.4300

Epoch 53/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9904 - loss: 0.4109

Epoch 54/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9904 - loss: 0.3811

Epoch 55/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.9730 - loss:
0.3814
Epoch 56/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9830 - loss: 0.3378

Epoch 57/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9847 - loss: 0.2998

Epoch 58/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9826 - loss: 0.2855

Epoch 59/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.2662

Epoch 60/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.2457

Epoch 61/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 1.0000 - loss:
0.2310
Epoch 62/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.2124

Epoch 63/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9943 - loss: 0.1936

Epoch 64/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9826 - loss: 0.1971
```

```
Epoch 65/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.1889
Epoch 66/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.1649
Epoch 67/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.1560
Epoch 68/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.1389
Epoch 69/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 1.0000 - loss: 0.1356
Epoch 70/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.1338
Epoch 71/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.1186
Epoch 72/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.1158
Epoch 73/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - accuracy: 1.0000 - loss: 0.1121
Epoch 74/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.1123
Epoch 75/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0913
Epoch 76/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0906
Epoch 77/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0855
Epoch 78/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0838
Epoch 79/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0766
Epoch 80/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 1.0000 - loss: 0.0807
```

```python
import seaborn as sns
sns.lineplot(model.history.history)
```

```
<Axes: >
```




```
from sklearn.decomposition import PCA

embeddings = model.get_weights()[0]

pca = PCA(n_components=2)
reduced_embeddings = pca.fit_transform(embeddings)

print("'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.")
```

```
'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.
```

```python
# test model: select some sentences from above paragraph
test_sentenses = [
    "known as structured learning",
    "transformers have applied to",
    "where they produced results",
    "cases surpassing expert performance"
]

for sent in test_sentenses:
    test_words = sent.split(" ")
#     print(test_words)
    x_test =[]
    for i in test_words:
        x_test.append(word_to_index.get(i))
    x_test = np.array([x_test])
#     print(x_test)

    pred = model.predict(x_test)
    pred = np.argmax(pred[0])
    print("pred ", test_words, "\n=", index_to_word.get(pred),"\n\n")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step
pred  ['known', 'as', 'structured', 'learning']
= deep


1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
pred  ['transformers', 'have', 'applied', 'to']
= been


1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
pred  ['where', 'they', 'produced', 'results']
= have


1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
pred  ['cases', 'surpassing', 'expert', 'performance']
= human
```