In [ ]:
```
import numpy as np
import re
```

In [ ]:
```
data = """Deep learning (also known as deep structured learning) is part of a
data
```

Out[ ]:
```
'Deep learning (also known as deep structured learning) is part of a broader
family of machine learning methods based on artificial neural networks with r
epresentation learning. Learning can be supervised, semi-supervised or unsupe
rvised. Deep-learning architectures such as deep neural networks, deep belief
networks, deep reinforcement learning, recurrent neural networks, convolution
al neural networks and Transformers have been applied to fields including com
puter vision, speech recognition, natural language processing, machine transl
ation, 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.'
```

In [ ]:
```
sentences=data.split('.');
sentences
```

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

In [ ]:
```
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
```

Out[ ]:
```
['deep learning also known as deep structured learning is part of broader fam
ily of machine learning methods based on artificial neural networks with repr
esentation learning',
 'learning can be supervised semi supervised or unsupervised',
 'deep learning architectures such as deep neural networks deep belief networ
ks deep reinforcement learning recurrent neural networks convolutional neural
networks and transformers have been applied to fields including computer visi
on speech recognition natural language processing machine translation bioinfo
rmatics drug design medical image analysis climate science material inspectio
n and board game programs where they have produced results comparable to and
in some cases surpassing human expert performance']
```

In [ ]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
```

In [ ]:
```
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, 7, 8, 1, 19, 20, 21, 22, 4, 3,
23, 24, 1], [1, 25, 26, 9, 27, 9, 28, 29], [2, 1, 30, 31, 6, 2, 4, 3, 2, 32, 3,
2, 33, 1, 34, 4, 3, 35, 4, 3, 5, 36, 10, 37, 38, 11, 39, 40, 41, 42, 43, 44, 4
5, 46, 47, 8, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 5, 59, 60, 61, 62, 6
3, 10, 64, 65, 66, 11, 5, 67, 68, 69, 70, 71, 72, 73]]
```

In [ ]:
```
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: 'broader', 18: 'family', 8: 'machine', 19:
'methods', 20: 'based', 21: 'on', 22: 'artificial', 4: 'neural', 3: 'networks',
23: 'with', 24: 'representation', 25: 'can', 26: 'be', 9: 'supervised', 27: 'se
mi', 28: 'or', 29: 'unsupervised', 30: 'architectures', 31: 'such', 32: 'belie
f', 33: 'reinforcement', 34: 'recurrent', 35: 'convolutional', 5: 'and', 36: 't
ransformers', 10: 'have', 37: 'been', 38: 'applied', 11: 'to', 39: 'fields', 4
0: 'including', 41: 'computer', 42: 'vision', 43: 'speech', 44: 'recognition',
45: 'natural', 46: 'language', 47: 'processing', 48: 'translation', 49: 'bioinf
ormatics', 50: 'drug', 51: 'design', 52: 'medical', 53: 'image', 54: 'analysi
s', 55: 'climate', 56: 'science', 57: 'material', 58: 'inspection', 59: 'boar
d', 60: 'game', 61: 'programs', 62: 'where', 63: 'they', 64: 'produced', 65: 'r
esults', 66: 'comparable', 67: 'in', 68: 'some', 69: 'cases', 70: 'surpassing',
71: 'human', 72: 'expert', 73: 'performance'}

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

In [ ]:

```
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
#         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,
7], [7, 17, 7, 8], [17, 18, 8, 1], [18, 7, 1, 19], [7, 8, 19, 20], [8, 1, 20, 2
1], [1, 19, 21, 22], [19, 20, 22, 4], [20, 21, 4, 3], [21, 22, 3, 23], [22, 4,
23, 24], [4, 3, 24, 1], [1, 25, 9, 27], [25, 26, 27, 9], [26, 9, 9, 28], [9, 2
7, 28, 29], [2, 1, 31, 6], [1, 30, 6, 2], [30, 31, 2, 4], [31, 6, 4, 3], [6, 2,
3, 2], [2, 4, 2, 32], [4, 3, 32, 3], [3, 2, 3, 2], [2, 32, 2, 33], [32, 3, 33,
1], [3, 2, 1, 34], [2, 33, 34, 4], [33, 1, 4, 3], [1, 34, 3, 35], [34, 4, 35,
4], [4, 3, 4, 3], [3, 35, 3, 5], [35, 4, 5, 36], [4, 3, 36, 10], [3, 5, 10, 3
7], [5, 36, 37, 38], [36, 10, 38, 11], [10, 37, 11, 39], [37, 38, 39, 40], [38,
11, 40, 41], [11, 39, 41, 42], [39, 40, 42, 43], [40, 41, 43, 44], [41, 42, 44,
45], [42, 43, 45, 46], [43, 44, 46, 47], [44, 45, 47, 8], [45, 46, 8, 48], [46,
47, 48, 49], [47, 8, 49, 50], [8, 48, 50, 51], [48, 49, 51, 52], [49, 50, 52, 5
3], [50, 51, 53, 54], [51, 52, 54, 55], [52, 53, 55, 56], [53, 54, 56, 57], [5
4, 55, 57, 58], [55, 56, 58, 5], [56, 57, 5, 59], [57, 58, 59, 60], [58, 5, 60,
61], [5, 59, 61, 62], [59, 60, 62, 63], [60, 61, 63, 10], [61, 62, 10, 64], [6
2, 63, 64, 65], [63, 10, 65, 66], [10, 64, 66, 11], [64, 65, 11, 5], [65, 66,
5, 67], [66, 11, 67, 68], [11, 5, 68, 69], [5, 67, 69, 70], [67, 68, 70, 71],
[68, 69, 71, 72], [69, 70, 72, 73]]

[12, 13, 6, 2, 14, 1, 15, 16, 7, 17, 18, 7, 8, 1, 19, 20, 21, 22, 4, 3, 23, 26,
9, 27, 9, 30, 31, 6, 2, 4, 3, 2, 32, 3, 2, 33, 1, 34, 4, 3, 35, 4, 3, 5, 36, 1
0, 37, 38, 11, 39, 40, 41, 42, 43, 44, 45, 46, 47, 8, 48, 49, 50, 51, 52, 53, 5
4, 55, 56, 57, 58, 5, 59, 60, 61, 62, 63, 10, 64, 65, 66, 11, 5, 67, 68, 69, 7
0, 71]
```

In [ ]:
```
#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
```

In [ ]:
```
X = np.array(contexts)
Y = np.array(targets)
```

In [ ]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Embedding , Lambda
```

In [ ]:
```
model = Sequential([
    Embedding(input_dim=vocab_size, output_dim=emb_size, input_length=2*contex
    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/python3.12/dist-packages/keras/src/layers/core/embedding.py:97:
UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
```

In [ ]:
```
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metric
```

In [ ]:
```
history=model.fit(X,Y,epochs=80)
```

```
Epoch 1/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 29ms/step - accuracy: 0.0483 - loss: 4.3044
Epoch 2/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.1046 - loss: 4.2978
Epoch 3/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - accuracy: 0.0988 - loss: 4.2909
Epoch 4/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.1103 - loss: 4.2815
Epoch 5/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.1085 - loss: 4.2693
Epoch 6/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.1181 - loss: 4.2485
Epoch 7/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.0947 - loss: 4.2332
Epoch 8/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.1064 - loss: 4.1995
Epoch 9/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - accuracy: 0.1007 - loss: 4.1556
Epoch 10/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.0811 - loss: 4.1051
Epoch 11/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - accuracy: 0.0696 - loss: 4.0413
Epoch 12/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.0678 - loss: 3.9541
Epoch 13/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.0600 - loss: 3.9057
Epoch 14/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - accuracy: 0.0561 - loss: 3.8405
Epoch 15/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 110ms/step - accuracy: 0.0522 - loss: 3.7472
Epoch 16/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step - accuracy: 0.0830 - loss: 3.7287
Epoch 17/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 57ms/step - accuracy: 0.1763 - loss: 3.5847
Epoch 18/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step - accuracy: 0.1430 - loss: 3.5247
Epoch 19/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.1664 - loss: 3.3557
Epoch 20/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - accuracy: 0.1643 - loss: 3.3346
Epoch 21/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step - accuracy: 0.1839 - loss: 3.2059
Epoch 22/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - accuracy: 0.1839 - loss: 3.1260
Epoch 23/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.1234 - loss: 3.0866
Epoch 24/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - accuracy: 0.1701 - loss: 2.9506
Epoch 25/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.2186 - loss: 2.8271
Epoch 26/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.2399 - loss: 2.7359
Epoch 27/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.2767 - loss: 2.6429
```

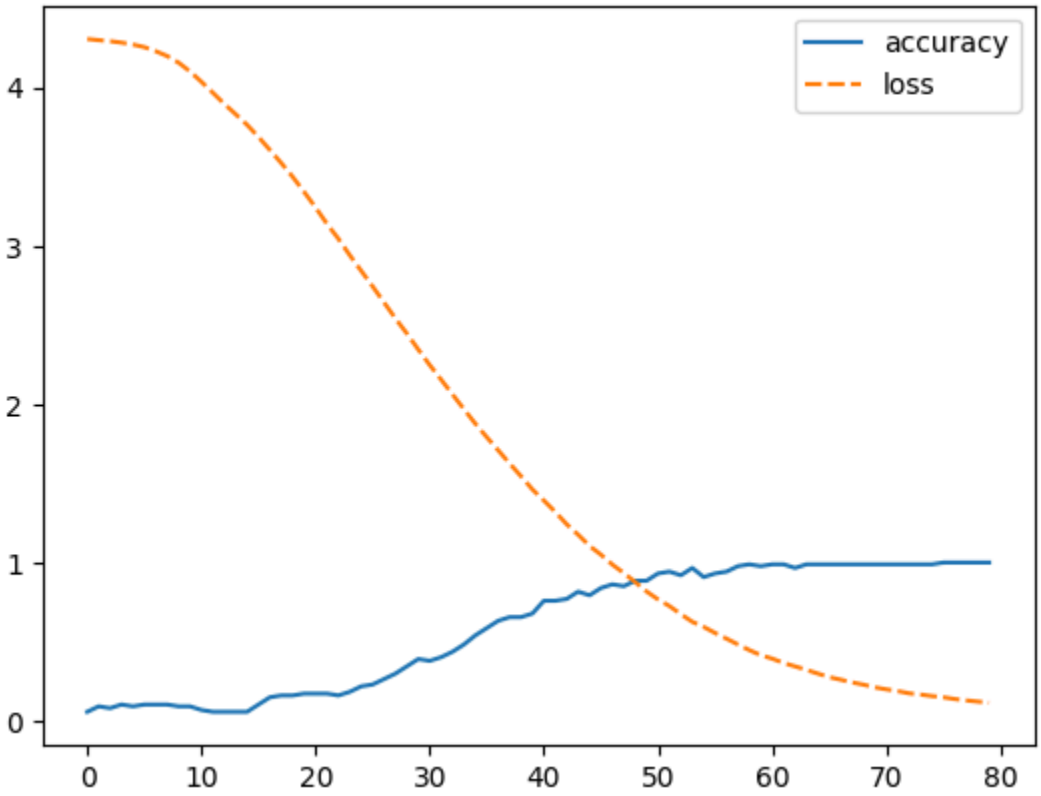
```
Epoch 28/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.2979 - loss: 2.5379
Epoch 29/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.3248 - loss: 2.4848
Epoch 30/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.3751 - loss: 2.3757
Epoch 31/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.3459 - loss: 2.2385
Epoch 32/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.3847 - loss: 2.1860
Epoch 33/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.4489 - loss: 2.0367
Epoch 34/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.5226 - loss: 1.9526
Epoch 35/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.5826 - loss: 1.8516
Epoch 36/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.5822 - loss: 1.7996
Epoch 37/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.6286 - loss: 1.7106
Epoch 38/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.6518 - loss: 1.6274
Epoch 39/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.6635 - loss: 1.5125
Epoch 40/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.6828 - loss: 1.4543
Epoch 41/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.7660 - loss: 1.3959
Epoch 42/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.7934 - loss: 1.2884
Epoch 43/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.7796 - loss: 1.2072
Epoch 44/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.8534 - loss: 1.1626
Epoch 45/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.8145 - loss: 1.0949
Epoch 46/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.8180 - loss: 1.0658
Epoch 47/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.8646 - loss: 1.0081
Epoch 48/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.8667 - loss: 0.9294
Epoch 49/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.8839 - loss: 0.8780
Epoch 50/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.8996 - loss: 0.8011
Epoch 51/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9265 - loss: 0.7819
Epoch 52/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9400 - loss: 0.7382
Epoch 53/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9168 - loss: 0.6739
Epoch 54/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9593 - loss: 0.6384
```

```
Epoch 55/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9071 - loss: 0.6027
Epoch 56/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9304 - loss: 0.5639
Epoch 57/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9322 - loss: 0.5379
Epoch 58/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9885 - loss: 0.4657
Epoch 59/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9903 - loss: 0.4196
Epoch 60/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9768 - loss: 0.4188
Epoch 61/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9903 - loss: 0.3923
Epoch 62/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9943 - loss: 0.3762
Epoch 63/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9828 - loss: 0.3374
Epoch 64/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9903 - loss: 0.3163
Epoch 65/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9903 - loss: 0.2867
Epoch 66/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9825 - loss: 0.2864
Epoch 67/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9903 - loss: 0.2532
Epoch 68/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 0.9825 - loss: 0.2541
Epoch 69/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9903 - loss: 0.2382
Epoch 70/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.9825 - loss: 0.2083
Epoch 71/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9825 - loss: 0.1918
Epoch 72/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9825 - loss: 0.1844
Epoch 73/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9903 - loss: 0.1732
Epoch 74/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9825 - loss: 0.1715
Epoch 75/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.9903 - loss: 0.1574
Epoch 76/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - accuracy: 1.0000 - loss: 0.1547
Epoch 77/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 1.0000 - loss: 0.1477
Epoch 78/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 1.0000 - loss: 0.1390
Epoch 79/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 1.0000 - loss: 0.1144
Epoch 80/80
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 1.0000 - loss: 0.1232
```

In [ ]:
```
import seaborn as sns
```

```
sns.lineplot(model.history.history)
```

Out[ ]: <Axes: >




In [ ]:
```
from sklearn.decomposition import PCA

embeddings = model.get_weights()[0]

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

In [ ]:
```
print("'Deep learning (also known as deep structured learning) is part of a br
```

```
'Deep learning (also known as deep structured learning) is part of a broader fa
mily of machine learning methods based on artificial neural networks with repre
sentation learning. Learning can be supervised, semi-supervised or unsupervise
d. Deep-learning architectures such as deep neural networks, deep belief networ
ks, deep reinforcement learning, recurrent neural networks, convolutional neura
l networks and Transformers have been applied to fields including computer visi
on, speech recognition, natural language processing, machine translation, bioin
formatics, drug design, medical image analysis, climate science, material inspe
ction and board game programs, where they have produced results comparable to a
nd in some cases surpassing human expert performance.
```

In [ ]:
```
test_sentenses = [
    "known as structured learning",
    "transformers have applied to",
    "where they produced results",
    "cases surpassing expert performance"
```

```
]
```

In [ ]:
```
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 85ms/step
pred  ['known', 'as', 'structured', 'learning']
= deep


1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
pred  ['transformers', 'have', 'applied', 'to']
= been


1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
pred  ['where', 'they', 'produced', 'results']
= have


1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
pred  ['cases', 'surpassing', 'expert', 'performance']
= human
```

In [ ]:
```
#Terminal
```

In [ ]:
```
```