Name: **Yash Waghumbare**

Div: **BE9-S9**

Roll no: **43180**

Title: **Assignment 5: Implement the Continuous Bag of Words (CBOW) Model**

In [1]:
```
#importing libraries
from keras.preprocessing import text
from keras.utils import np_utils
from keras.preprocessing import sequence
from keras.utils import pad_sequences
import numpy as np
import pandas as pd
```

In [2]:
```
#taking random sentences as data
data = """Deep learning (also known as deep structured learning) is part of a broader family of machine learning methods based on artificial neural n
Deep-learning architectures such as deep neural networks, deep belief networks, deep reinforcement learning, recurrent neural networks, convolutiona
"""
dl_data = data.split()
```

In [3]:
```
#tokenization
tokenizer = text.Tokenizer()
tokenizer.fit_on_texts(dl_data)
word2id = tokenizer.word_index

word2id['PAD'] = 0
id2word = {v:k for k, v in word2id.items()}
wids = [[word2id[w] for w in text.text_to_word_sequence(doc)] for doc in dl_data]

vocab_size = len(word2id)
embed_size = 100
window_size = 2

print('Vocabulary Size:', vocab_size)
print('Vocabulary Sample:', list(word2id.items())[:10])
```

```
Vocabulary Size: 75
Vocabulary Sample: [('learning', 1), ('deep', 2), ('networks', 3), ('neural', 4), ('and', 5), ('as', 6), ('of', 7), ('machine', 8), ('supervised',
9), ('have', 10)]
```

In [4]:
```
#generating (context word, target/label word) pairs
def generate_context_word_pairs(corpus, window_size, vocab_size):
    context_length = window_size*2
    for words in corpus:
        sentence_length = len(words)
        for index, word in enumerate(words):
            context_words = []
            label_word   = []
            start = index - window_size
            end = index + window_size + 1

            context_words.append([words[i]
                                 for i in range(start, end)
                                 if 0 <= i < sentence_length
                                 and i != index])
            label_word.append(word)

            x = pad_sequences(context_words, maxlen=context_length)
            y = np_utils.to_categorical(label_word, vocab_size)
            yield (x, y)

i = 0
for x, y in generate_context_word_pairs(corpus=wids, window_size=window_size, vocab_size=vocab_size):
    if 0 not in x[0]:
        # print('Context (X):', [id2word[w] for w in x[0]], '-> Target (Y):', id2word[np.argwhere(y[0])[0][0]])

        if i == 10:
            break
        i += 1
```

In [5]:
```
#model building
import keras.backend as K
from keras.models import Sequential
from keras.layers import Dense, Embedding, Lambda

cbow = Sequential()
cbow.add(Embedding(input_dim=vocab_size, output_dim=embed_size, input_length=window_size*2))
cbow.add(Lambda(lambda x: K.mean(x, axis=1), output_shape=(embed_size,)))
cbow.add(Dense(vocab_size, activation='softmax'))
cbow.compile(loss='categorical_crossentropy', optimizer='rmsprop')

print(cbow.summary())

# from IPython.display import SVG
# from keras.utils.vis_utils import model_to_dot

# SVG(model_to_dot(cbow, show_shapes=True, show_layer_names=False, rankdir='TB').create(prog='dot', format='svg'))
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 4, 100)            7500

 lambda (Lambda)             (None, 100)               0

 dense (Dense)               (None, 75)                7575

=================================================================
Total params: 15,075
Trainable params: 15,075
Non-trainable params: 0
_________________________________________________________________
None
```

In [6]:
```
for epoch in range(1, 6):
    loss = 0.
    i = 0
    for x, y in generate_context_word_pairs(corpus=wids, window_size=window_size, vocab_size=vocab_size):
        i += 1
        loss += cbow.train_on_batch(x, y)
        if i % 100000 == 0:
            print('Processed {} (context, word) pairs'.format(i))

    print('Epoch:', epoch, '\tLoss:', loss)
    print()
```

```
Epoch: 1        Loss: 434.40525007247925

Epoch: 2        Loss: 429.64614844322205

Epoch: 3        Loss: 426.254625082016

Epoch: 4        Loss: 422.88486409187317

Epoch: 5        Loss: 420.2294900417328
```

In [7]:
```
weights = cbow.get_weights()[0]
weights = weights[1:]
print(weights.shape)

pd.DataFrame(weights, index=list(id2word.values())[1:]).head()
```

```
(74, 100)
```

Out[7]:

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94	95
deep	-0.028218	-0.005919	0.005274	-0.029521	-0.022013	-0.019620	0.027524	0.011648	0.025632	-0.008394	...	0.014053	0.002022	-0.046732	0.045974	-0.040925	-0.039103
networks	0.004388	-0.018607	0.009451	0.030428	-0.031672	0.031915	0.055260	0.020617	-0.008885	-0.030407	...	-0.002793	0.042926	0.050381	0.053126	0.003252	0.033827
neural	0.029352	-0.036660	0.021049	0.003298	-0.023420	0.046911	-0.039212	0.010056	0.043364	-0.042134	...	-0.033033	0.013651	-0.043134	-0.045682	0.017554	-0.042856
and	0.022546	0.006237	0.001115	-0.019212	0.003657	-0.048563	0.045061	-0.048979	-0.025171	0.004712	...	0.002042	-0.031780	0.047122	0.016723	-0.014286	-0.018209
as	-0.013465	-0.035403	0.010038	0.037268	-0.045731	0.005324	-0.017414	-0.005259	0.041465	-0.014640	...	-0.038623	0.010498	-0.013775	0.005803	0.013803	-0.037896

5 rows × 100 columns

In [8]:
```
from sklearn.metrics.pairwise import euclidean_distances

distance_matrix = euclidean_distances(weights)
print(distance_matrix.shape)

similar_words = {search_term: [id2word[idx] for idx in distance_matrix[word2id[search_term]-1].argsort()[1:6]+1]
                 for search_term in ['deep']}

similar_words
```

```
(74, 74)
```

Out[8]:
```
{'deep': ['transformers', 'climate', 'convolutional', 'of', 'family']}
```

In [ ]: