DL
~~ISR~~ Exp 5

Aim: Implement the CBOW stages combe:
a) Data preparation
b) generate training data
c) Train model
d) Output.

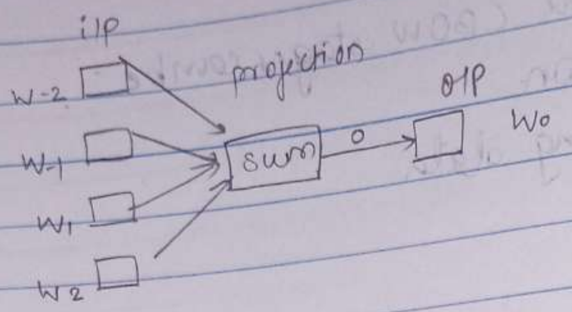
Obj: To learn and understand CBOW

Infrastructure: Computer / Laptop / VM.

Software used: Jupyter Notebook / Google colab, Tensorflow, keras.

Theory:
The CBOW model tries to understand context of the words and takes this as i/p. It then tries to predict words that are contextually accurate. Let us consider an eg for understanding this. Consider the sentence 'It's a pleasant day' and word 'pleasant' goes up to neural network. We are trying to predict word 'day' here. We will use one hot encoding for i/p words and measure the error rates with one-hot encoded target word. Doing this will help us predict output based on word with least error.

→ model architecture:



The CBOW model architecture is as shown above. The model tries to predict the target word by trying to understand the context of surrounding words. Consider the same sentence as above 'Its a pleasant day'. The model converts this sentence into word pairs in form. With these wordpairs, the model tries to predict the target word considered context words.

If we have 4 context words used for predicting one target word the i/p layer will be in the form of four 1xW i/p vectors. Finally 1xN output from hidden layer enters the sum layer where element wise summation is performed and O/p is obtained.

→ Implementation of CBOW model:

- import libraries and read dataset.
- For the implementation
- Generate func that create window sizes and pairs of target words.
- Build neural network on sample data.

Exp No.5

②

Conclusion : we saw what a CBOW model is how it works. These can be used for text recognition, speech to text conversion, etc.

In [1]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib as mpl
import matplotlib.pylab as pylab
import numpy as np
```

In [2]:
```
import re
```

In [3]:
```
sentences = """We are about to study the idea of a computational process.
Computational processes are abstract beings that inhabit computers.
As they evolve, processes manipulate other abstract things called data.
The evolution of a process is directed by a pattern of rules
called a program. People create programs to direct processes. In effect,
we conjure the spirits of the computer with our spells."""
```

In [4]:
```
sentences = re.sub('[^A-Za-z0-9]+', ' ', sentences)
```

In [5]:
```
sentences = re.sub(r'(?:^| )\w(?:$| )', ' ', sentences).strip()
```

In [6]:
```
sentences = sentences.lower()
```

In [7]:
```
words = sentences.split()
vocab = set(words)
```

In [8]:
```
vocab_size = len(vocab)
embed_dim = 10
context_size = 2
```

In [9]:
```
word_to_ix = {word: i for i, word in enumerate(vocab)}
ix_to_word = {i: word for i, word in enumerate(vocab)}
```

In [10]:
```
data = []
for i in range(2, len(words) - 2):
    context = [words[i - 2], words[i - 1], words[i + 1], words[i + 2]]
    target = words[i]
    data.append((context, target))
print(data[:5])
```

```
[(['we', 'are', 'to', 'study'], 'about'), (['are', 'about', 'study', 'the'], 't
o'), (['about', 'to', 'the', 'idea'], 'study'), (['to', 'study', 'idea', 'of'],
'the'), (['study', 'the', 'of', 'computational'], 'idea')]
```

In [11]:
```
embeddings =  np.random.random_sample((vocab_size, embed_dim))
```

In [12]:
```
def linear(m, theta):
    w = theta
    return m.dot(w)
```

In [13]:
```
def log_softmax(x):
    e_x = np.exp(x - np.max(x))
    return np.log(e_x / e_x.sum())
```

In [14]:
```
def NLLLoss(logs, targets):
    out = logs[range(len(targets)), targets]
    return -out.sum()/len(out)
```

In [15]:
```
def log_softmax_crossentropy_with_logits(logits,target):

    out = np.zeros_like(logits)
    out[np.arange(len(logits)),target] = 1

    softmax = np.exp(logits) / np.exp(logits).sum(axis=-1,keepdims=True)

    return (- out + softmax) / logits.shape[0]
```

In [16]:
```
def forward(context_idxs, theta):
    m = embeddings[context_idxs].reshape(1, -1)
    n = linear(m, theta)
    o = log_softmax(n)

    return m, n, o
```

In [17]:
```
def backward(preds, theta, target_idxs):
    m, n, o = preds

    dlog = log_softmax_crossentropy_with_logits(n, target_idxs)
    dw = m.T.dot(dlog)

    return dw
```

In [18]:
```
def optimize(theta, grad, lr=0.03):
    theta -= grad * lr
    return theta
```

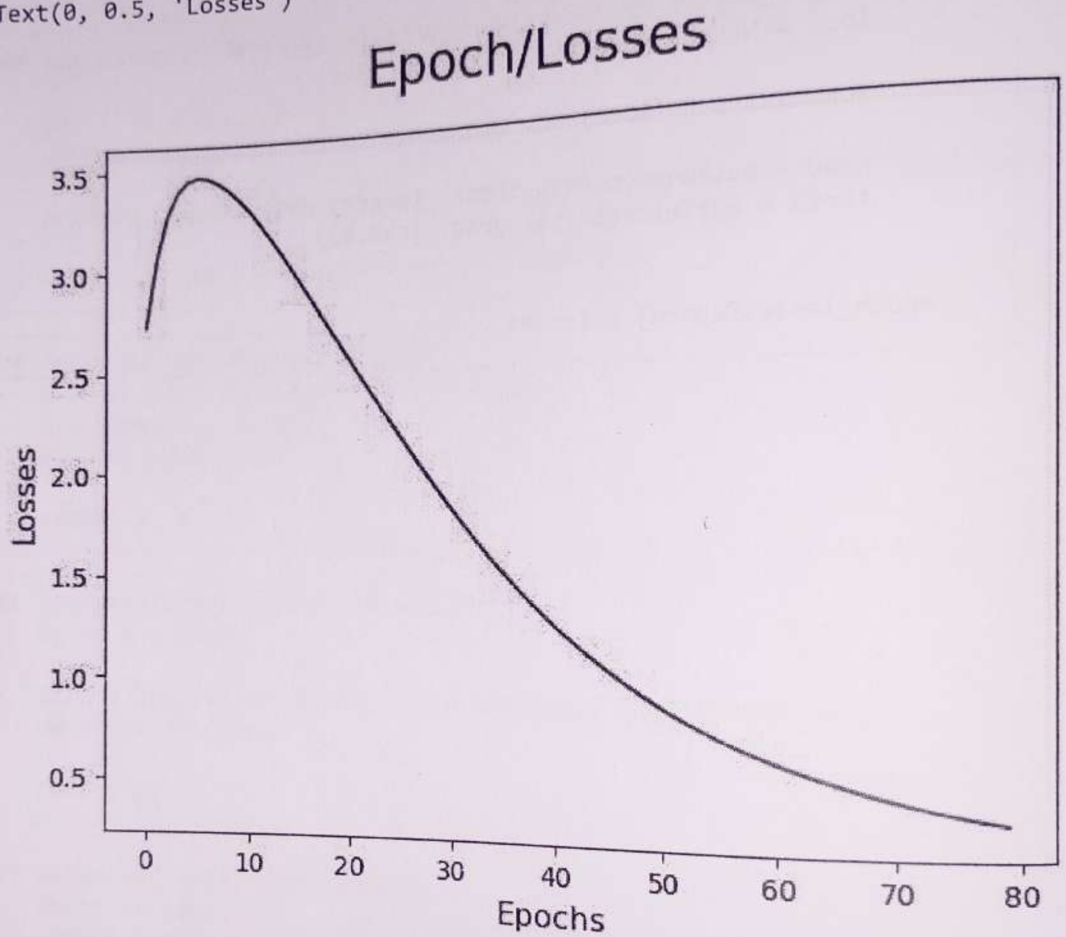
In [19]:
```
theta = np.random.uniform(-1, 1, (2 * context_size * embed_dim, vocab_size))
```

55

In [20]:
```
epoch_losses = {}

for epoch in range(80):

    losses =  []

    for context, target in data:
        context_idxs = np.array([word_to_ix[w] for w in context])
        preds = forward(context_idxs, theta)

        target_idxs = np.array([word_to_ix[target]])
        loss = NLLLoss(preds[-1], target_idxs)

        losses.append(loss)

        grad = backward(preds, theta, target_idxs)
        theta = optimize(theta, grad, lr=0.03)


    epoch_losses[epoch] = losses
```

In [21]:
```
ix = np.arange(0,80)

fig = plt.figure()
fig.suptitle('Epoch/Losses', fontsize=20)
plt.plot(ix,[epoch_losses[i][0] for i in ix])
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('Losses', fontsize=12)
```

Out[21]: Text(0, 0.5, 'Losses')



In [22]:
```
def predict(words):
    context_idxs = np.array([word_to_ix[w] for w in words])
    preds = forward(context_idxs, theta)
    word = ix_to_word[np.argmax(preds[-1])]

    return word
```

In [23]:
```
predict(['we', 'are', 'to', 'study'])
```

Out[23]: 'about'

57

In [24]:
```
def accuracy():
    wrong = 0

    for context, target in data:
        if(predict(context) != target):
            wrong += 1

    return (1 - (wrong / len(data)))
```

In [25]:
```
accuracy()
```

Out[25]: 1.0

In [26]:
```
predict(['processes', 'manipulate', 'things', 'study'])
```

Out[26]: 'pattern'

In [ ]: