Approach for Phrase extraction:

Introduction:

Phrase extraction can be seen as entity extraction. So given a sentence we have to identify if the words contained in it are eligible to be a “phrase” or not.
So in a simplified settings the whole problem statement can be seen as a “ Binary Classification” problem. Let words contained in a sentence be denoted by set {w}. So for each word we can have two classes:

- Phrasal word
- Non Phrasal word

If a sentence contains even a single “phrasal word”, it contains a phrase. If a sentence does not contain any phrasal word, we will output “Not Found”.

Observation:

Word unigrams are never a good choice for representing complex information unique to text like semantics and syntactic structure. Collocations follow the similar observation. We need to consider each word for classifying it in binary classes, but only word i.e. unigram is not powerful enough to deduce the generalized logic.

For this purpose I started labeling words as phrasal and non-phrasal, and represted the word as a collection of it’s neighbours as well. So , I took a window of 7 say label[7].
Label[3] – It is the actual word which needs to be classifies
Label[2] – It is the immediate left neighbour of the actual word
Label[4] – It is the immediate right neighbor of the actual word

Similarily, for each word, I represented it using a window of 7.

Advantages:
1) Window of 7 will capture the semantics of the word very well
2) This approach will enforce the syntactic structure on the word as the word will have to appear in accordance with it’s neighbours

The justification for using window of 7 can be found in this paper of G. Mesnil and Yoshua Bengio.

Yet another observation :

A window of 7, i.e. 3 neighbours from ether side. Now if I label this window as allowed classes “phrasal” or “non-phrasal” , immediate question appears in my mind is, which of the members of the window, this label refers to. For us, it may be easy to figure out, but for model each of the members of the window are equi probable. So for the model, this label can belong to any member with a probability 1/7.
So after training the model, the model will surely be confused about the combination of members inside a window that influence it’s decision.
To my rescue, comes the concept of statistical machine learning. If the scale of vectors is not same i.e. if the vectors are not normalized, the training of the model tend to bias towards the vector having large scale values.

One more Observation:

While using Deep learning, we create the deep networks, which are sequential in nature i.e. layer after layer architecture. So, if I have an input tensor of dimension 30*300 then there will be a lot of complexity in training as space requirement and computation requirement will be more. I have create a window of 7 out of total 30 words that I have kept fixed for each sentence. So, instead of 30*300 dimensional tensors, there are total 30 of the 7*300 tensors. So overall a boost in training complexity as the input tensors have less dimensions.

Another observation:
As I have fixed the length of sentences to 30 words, so to train the model over deep architecture, I will have to provide fix dimensional tensors. In this case 30*300.
Note: Not all sentences in the training data have 30 words.
So if I use the tensors of size 30*300, there will be lot of cases where there is no word and no label. Waste of computational capacity as well as threat of biases.

- ➔ **threat of biases?**
  As we are aware of the fact that the model is performing some complex algebraic computation for adjusting weights. So in case of plenty of short sentences, the model will set it's bias towards the most feasible value i.e. Not Found and will score brownie points as the loss will keep on decreasing.

Final Observation:
Reduction in size of training data. Originally I had 9819 sentences . If I set the sentence length to be maximum 30, I will have 294570 number of words in training. Because of window of 7, I can drop this constraint as I can have chunks that are actually present in the sentence.
In Final run of the algorithm, I had some 87 K words. Sufficient improvement.

Key steps:

1. Read all the sentences and phrases
2. token all the sentences and phrases
3. Preparing the training data as follows:

```
For sentence in list of sentences:
        for word in sentence:
            if word is present in phrase:
                mark it with class label "phrasal"
            else
                mark it with class label "non-phrasal"
```

4. Prepare the sentences embeddings which are basically 30*300 tensors using GLOVE word vectors
5. Divide the tensor in chunks of 7*300
6. With all the chunks of size 7*300 and corresponding 2 dimensional label for the two classes, train the model

Network Architecture:

```
def create_model(dim):
    model = Sequential()
    model.add(LSTM(100 , return_sequences=True , input_shape=(7 , 300)))
    model.add(Activation('sigmoid'))

    model.add(LSTM(200 , return_sequences = False))
    model.add(Activation('sigmoid'))

    model.add(Dense(400))
    model.add(Activation('sigmoid'))

    model.add(Dense(2))
    model.add(Activation('softmax'))
    return model
```

As we can see, there are two layers of LSTM and 1 layer of FNN and the final output layer.

- I have used “softmax” function in the final layer, so as to represent the two classes.
  - Phrasal
  - Non phrasal

- Checkpointing has been done to save only the best result. Monitor is on validation loss. Usually low validation loss with almost similar training loss gives the best performance. It is also the measure of less overfitiing. So training loss and validation loss shall go hand in hand

- Earlystopping is also provided so that we can early stop the model rather than running it full length of epochs if the performance of the model is not inproving.

- model.compile(loss='binary_crossentropy', optimizer='Adam')

- The model has been compiled using loss fuction “binary_crossentropy” which usually is best choice with the softmax activation function.

- Optimizer used is “Adam”.

Training:
I wanted to train the model for 100 epochs, with patience 50, but because of limited resources I had to interrupt as my laptop’s temp was almost touching 92 celcius.
I stopped the model after 30 epochs.

Loss : 0.26
val_loss : 0.25
Accuracy : 0.96 (was improving when i stopped the model)
Val_accuracy : 0.92

Other Appraoch:

I could have used 2 Dimensional convolutional layer with a filter size of 7 columns and 300 rows as the first layer instead of LSTM. But I donot have a great GPU where I can deploy my model. CNN’s generally are more computationally complex as they have lot of parameters.