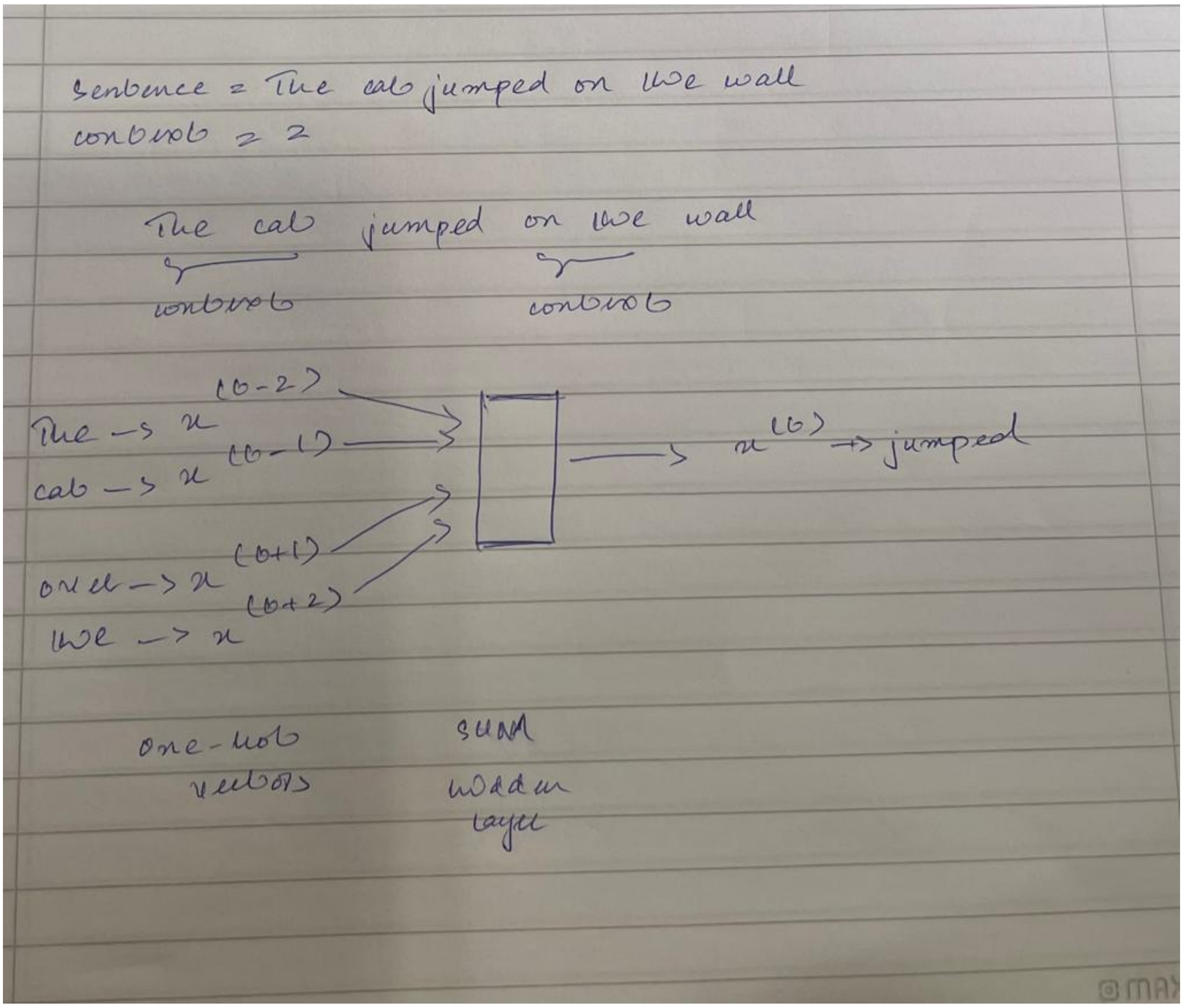
Name: Maha Tufail Agro    Spring 2022
Mid Pseudocode    NLP702

**Question # 02**

Assume you are asked to implement CBOW. Please choose a sentence and some sample training set, draw an annotated CBOW model architecture and write pseudocode. Make sure you show how you prepare the training samples, translate the training window and qualitatively and quantitatively evaluate your embedding implementation.

**Annotated CBOW Architecture**

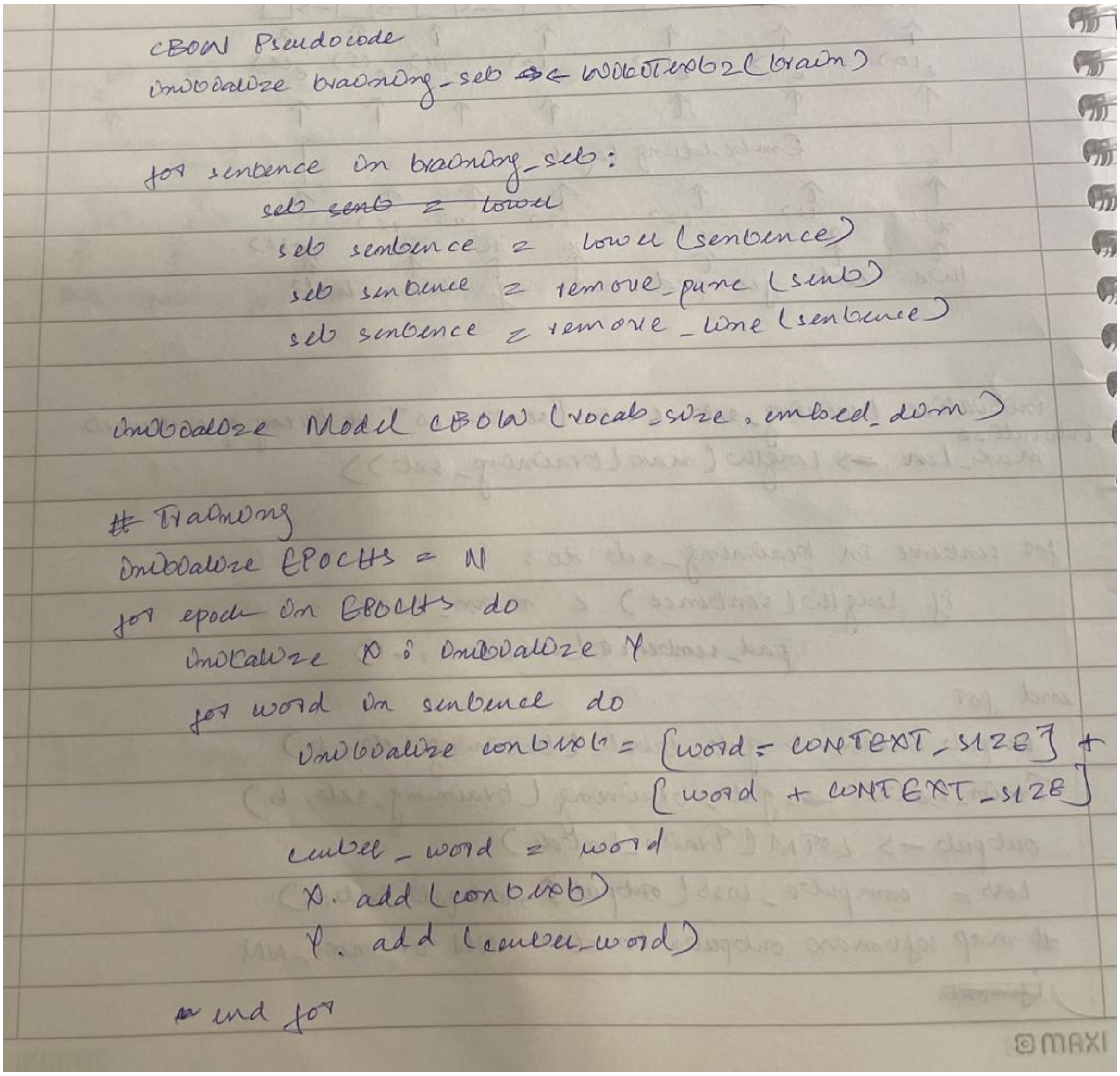


sentence = The cat jumped on the wall
context = 2

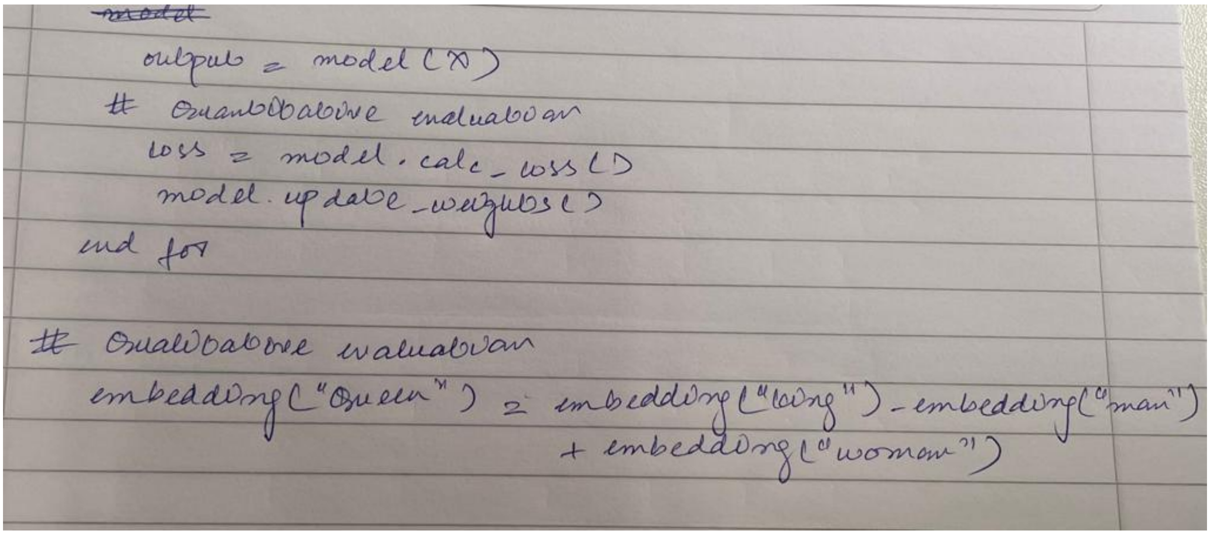
Name: Maha Tufail Agro
Mid Pseudocode

Spring 2022
NLP702

**CBOW Pseudocode**



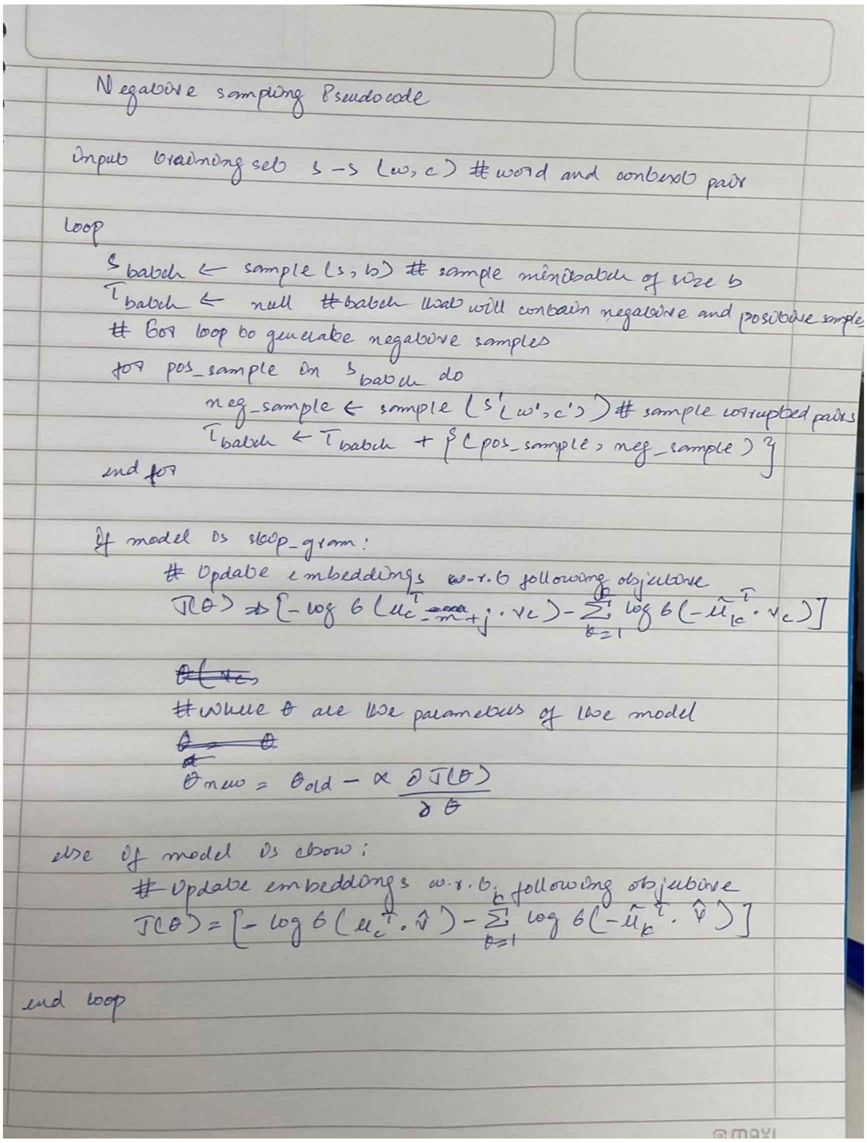
```
CBOW Pseudocode
initialize training_set ← WikiText2(train)

for sentence in training_set:
        ~~set sent = lower~~
        set sentence = lower(sentence)
        set sentence = remove_punc(sent)
        set sentence = remove_line(sentence)

initialize Model CBOW(vocab_size, embed_dim)

# Training
initialize EPOCHS = N
for epoch in EPOCHS do
    initialize X ; initialize Y
    for word in sentence do
        initialize context = [word - CONTEXT_SIZE] +
                             [word + CONTEXT_SIZE]
        center_word = word
        X.add(context)
        Y.add(center_word)
    end for
```



```
    ~~model~~
    output = model(X)
    # quantitative evaluation
    loss = model.calc_loss()
    model.update_weights()
end for

# qualitative evaluation
embedding("Queen") = embedding("king") - embedding("man")
                   + embedding("woman")
```

What is the computational bottleneck operation in word2vec? What can you adopt other than negative sampling? Briefly describe what is negative sampling. Please show how you would prepare the training data for such a task and write the pseudo code to show how the training will partially update the model weights.

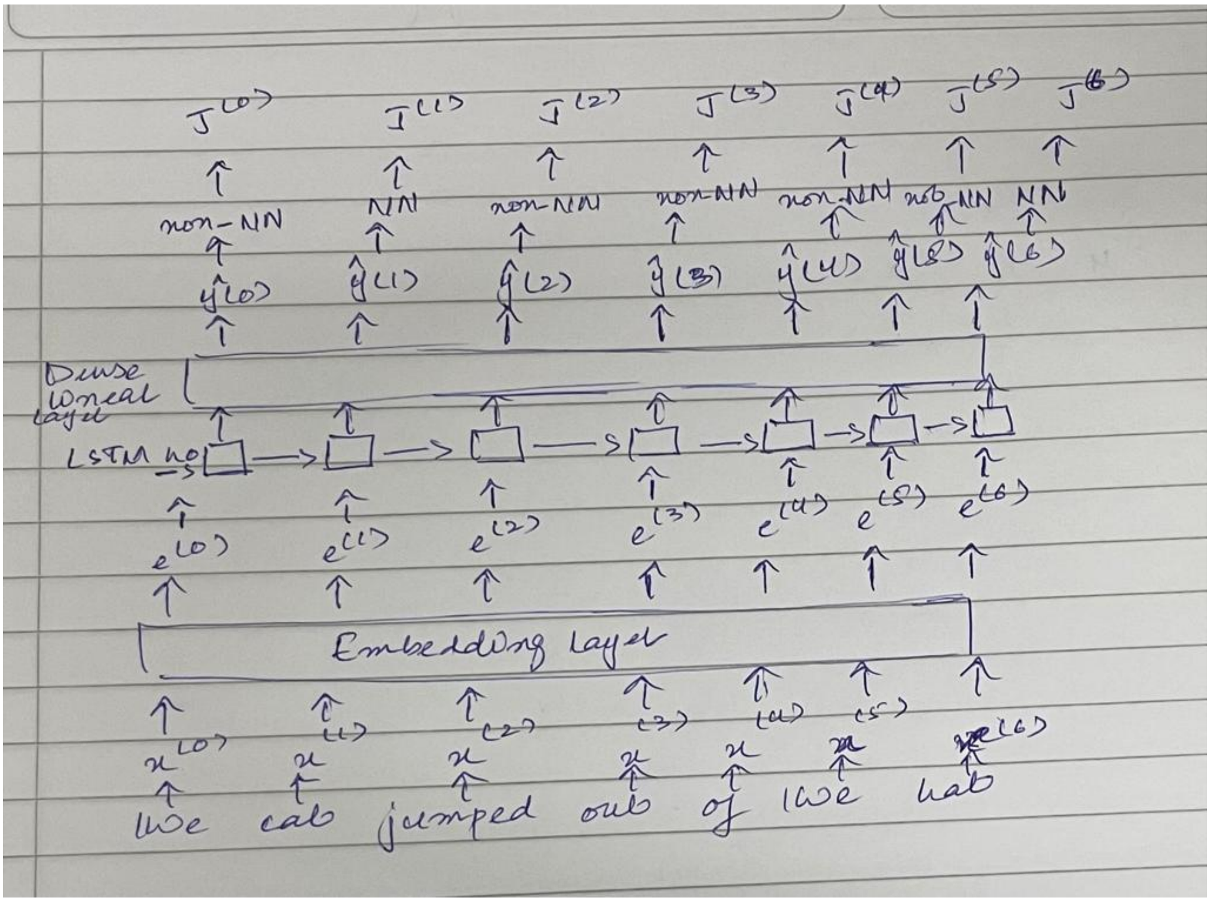
The computational bottleneck operation in the word2vec model is the final softmax layer. The softmax operation sums over the entire vocabulary making it the most expensive. Other than negative sampling a hierarchical softmax operation can be adopted which modifies the architecture of the softmax layer to improve its efficiency. The approach can yield up to 5x speedups.
**Negative Sampling:** As mentioned before, summation over |V| is computationally huge, practically |V| can be in millions. In each training step, instead of looping over the entire vocabulary, for each positive sample (word, context pair belonging to the corpus), the negative sampling algorithm generates several negative samples (word, context pair belonging to a false or negative corpus). The negative samples are sampled from a noise distribution $P_n(w)$. The algorithm maximizes the probability of a word and context being in the corpus data if it is, otherwise maximizes the probability of it not being in the corpus data if it is not.



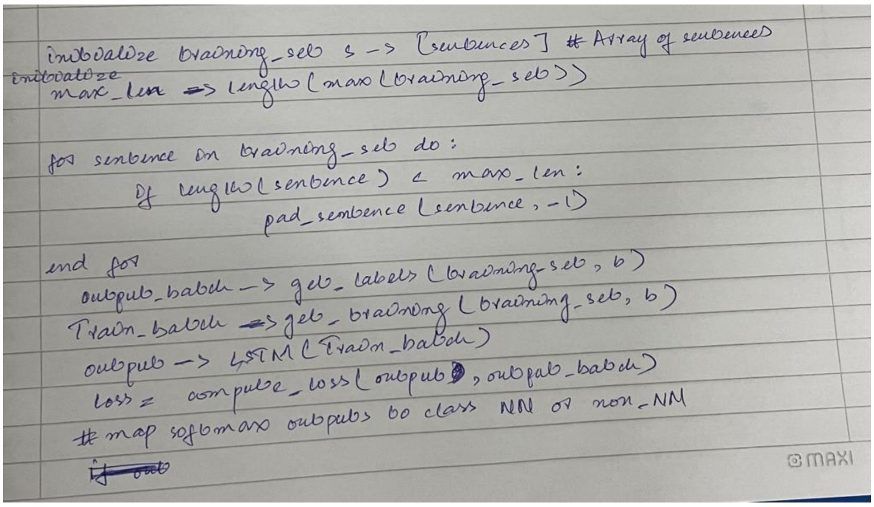
```
Negative sampling Pseudocode

Input training set s -> (w, c) # word and context pair

Loop
    $S_{batch}$ ← sample(s, b) # sample minibatch of size b
    $T_{batch}$ ← null # batch that will contain negative and positive sample
    # for loop to generate negative samples
    for pos_sample in $S_{batch}$ do
        neg_sample ← sample($S'(w', c')$) # sample corrupted pairs
        $T_{batch}$ ← $T_{batch}$ + {(pos_sample, neg_sample)}
    end for

    if model is skip_gram:
        # Update embeddings w.r.t θ following objective
        $J(\theta) = [-\log\sigma(u_{c-m+j}^T \cdot v_c) - \sum_{k=1}^{k}\log\sigma(-\tilde{u}_k^T \cdot v_c)]$

        ~~θ(vc)~~
        # where θ are the parameters of the model
        ~~θ = θ~~
        $\theta_{new} = \theta_{old} - \alpha \frac{\partial J(\theta)}{\partial \theta}$

    else if model is cbow:
        # Update embeddings w.r.t θ following objective
        $J(\theta) = [-\log\sigma(u_c^T \cdot \hat{v}) - \sum_{k=1}^{k}\log\sigma(-\tilde{u}_k^T \cdot \hat{v})]$

end loop
```

**Question # 05**

Please annotate an LSTM architecture to implement a NER task (is a NN or not) for the following sentence: the cat jumped out of the hat. What do you achieve when you include regularization in the NER cost function? Write a pseudocode showing how you handle different vector sizes and how you read LSTM output in NER.

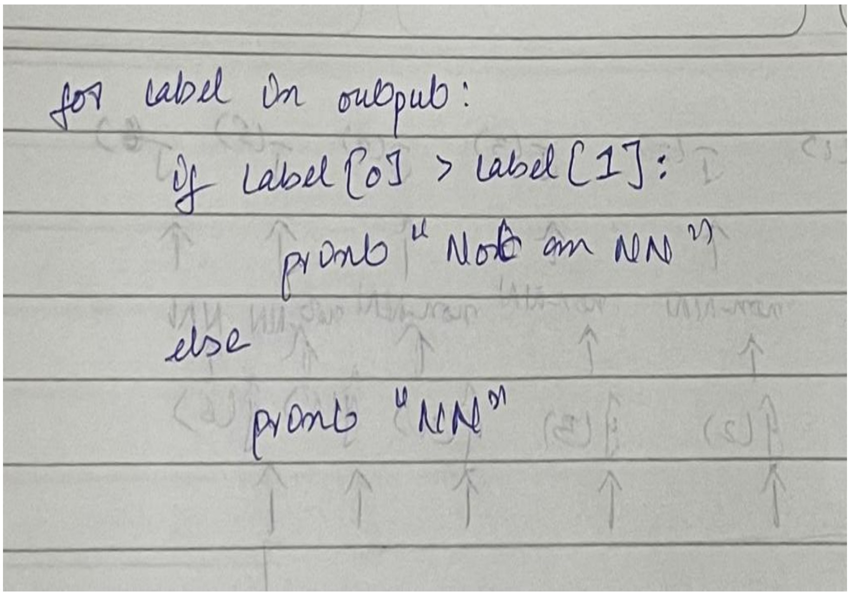


Neural network algorithms in general tend to be complex and prone to overfitting. Including a regularization term in the NER cost function, in-turn helps to reduce the complexity of the learning algorithm and reduce the variance in the model. The result is that the model becomes more general in nature.



```
initialize training_set s -> [sentences] # Array of sentences
~~initialize~~
max_len => length(max(training_set))

for sentence in training_set do:
    if length(sentence) < max_len:
        pad_sentence(sentence, -1)

end for
  output_batch -> get_labels(training_set, b)
  Train_batch => get_training(training_set, b)
  output -> LSTM(Train_batch)
  Loss = compute_loss(outputs, output_batch)
  # map softmax outputs to class NN or non_NM
  ~~if out~~
```

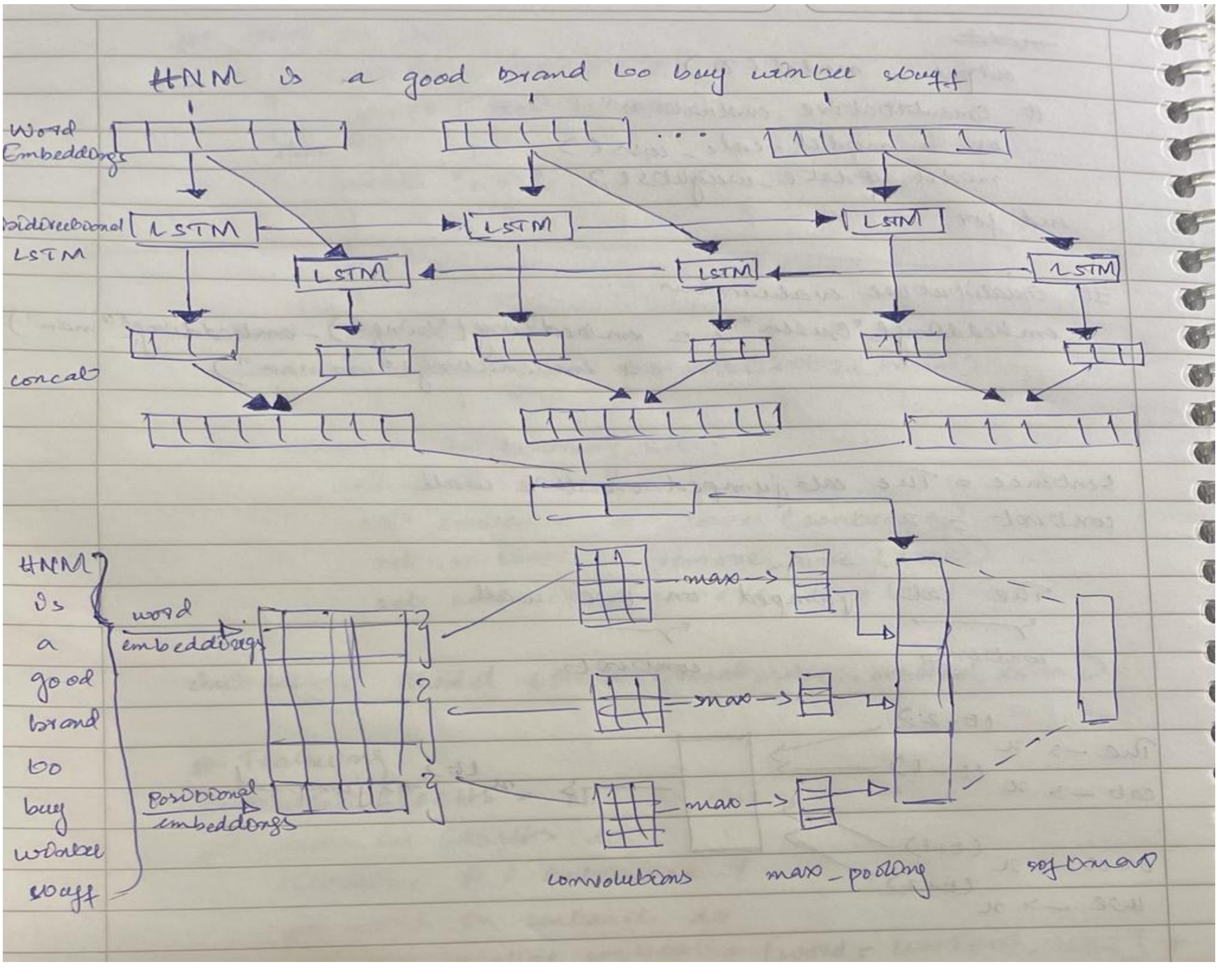
©MAXI



```
for label in output:
    if label[0] > label[1]:
        print "Not an NN"
    else
        print "NN"
```

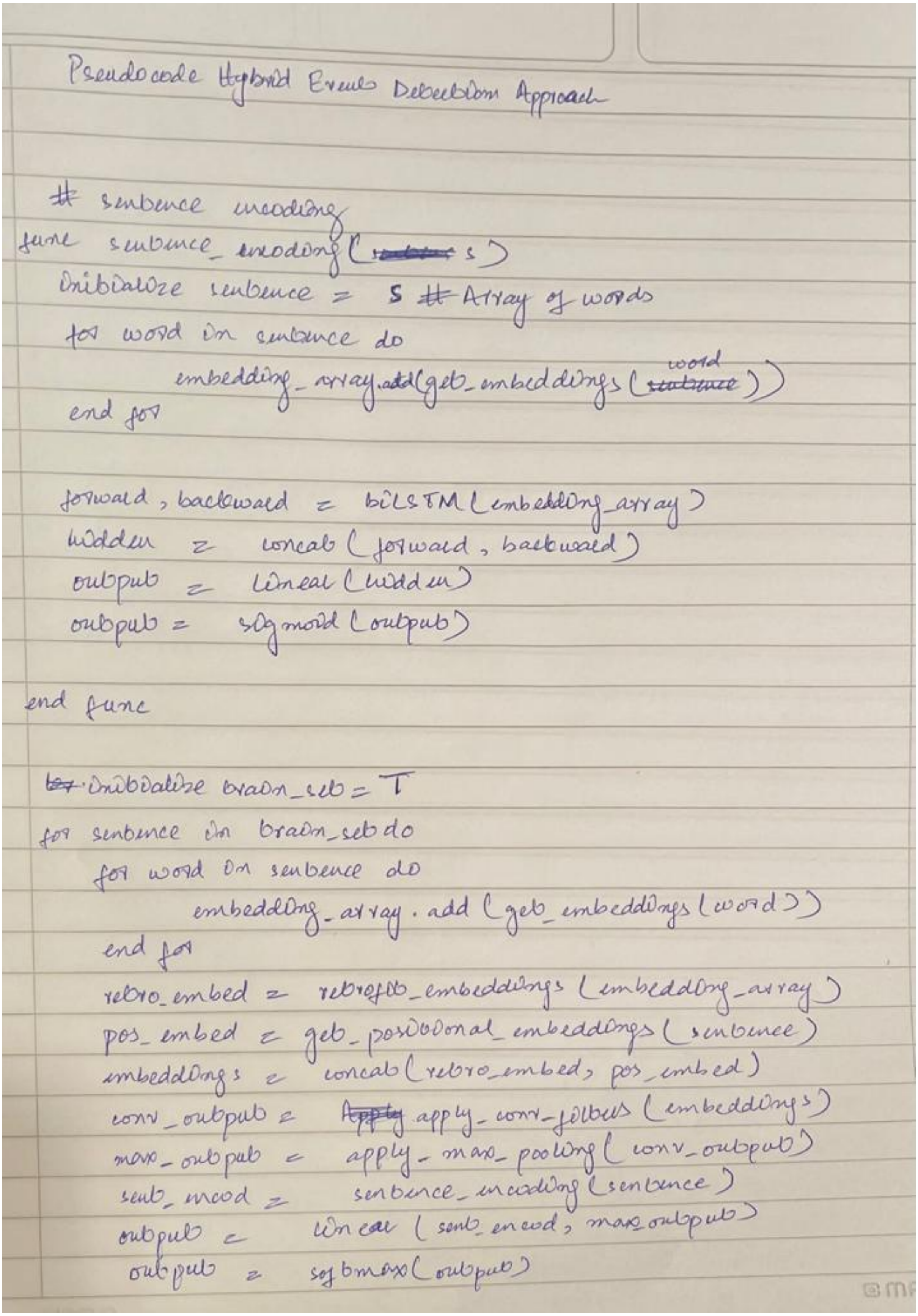
**Question # 06**

Sketch a block based diagram and write a pseudocode for a hybrid event extraction approach that uses: 1. A bidirectional LSTM for sentence encoding and 2. A CNN for event detection that uses word embeddings, positional embeddings and (bonus) a retrofitting of embeddings to existing lexicon as input.



Name: Maha Tufail Agro Spring 2022
Mid Pseudocode NLP702

**Pseudocode**



Pseudocode Hybrid Event Detection Approach

```
# sentence encoding
func sentence_encoding(~~sentence~~ s)
    initialize sentence = s # Array of words
    for word in sentence do
            embedding_array.add(get_embeddings(~~sentence~~ word))
    end for

    forward, backward = biLSTM(embedding_array)
    hidden = concat(forward, backward)
    output = linear(hidden)
    output = sigmoid(output)

end func

~~for~~ initialize train_set = T
for sentence in train_set do
    for word in sentence do
            embedding_array.add(get_embeddings(word))
    end for
    retro_embed = retrofit_embeddings(embedding_array)
    pos_embed = get_positional_embeddings(sentence)
    embeddings = concat(retro_embed, pos_embed)
    conv_output = ~~Apply~~ apply_conv_filters(embeddings)
    max_output = apply_max_pooling(conv_output)
    sent_encod = sentence_encoding(sentence)
    output = linear(sent_encod, max_output)
    output = softmax(output)
```