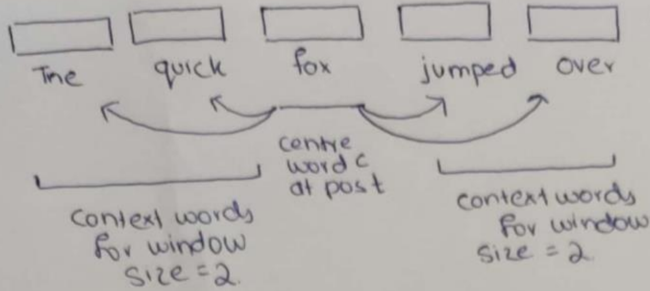
MID TERM NLP 702
PSUEDO CODES

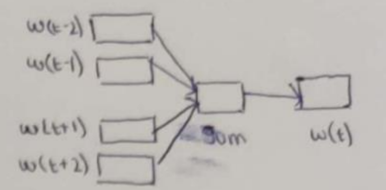
Muhammad Umar Salman - 21010241

Question 2

Assume you are asked to implement CBOW. Please choose a sentence and some sample training set, draw an annotated CBOW model architecture and write a pseudo code. Make sure you show how you prepare the training samples, translate the training window and qualitatively and quantitatively evaluate your embedding implementation. (4 points)



CBOW



Preparing Training Samples

window_size = 2
~~text~~ ← load_data()
~~text~~ ← text.lower()
text ← regex.replace(punctuation, ' ', text)
text ← text.split()
train, val, test ← train_test_split(text).
Iterate: for token in train:
if length(token)) > 2 * Window-Size + 1
then
for index, word in enumerate (window_size, length(token) - window size)
then
pre-window ← [words for words in token before centre word of size 2]
post_window ← [words for words in token after centre word of size window_size]
context_words ← pre_window + post_window
context_words ← [get_encoding(w) for w in context_words]
label_word ← [get_encoding (centre_word)

In Translate training window with more detail. ←

end
return context_words, label_word.

Translate training window

token ← [the, quick, fox, jumped, over, the, fence] # take w as token. in psuedo-code.
for w in ~~token~~ train:
if len(w) > 2 * window_size + 1
then
for i in range(window size start to length(w) - window_size)
pre-window ← [w(i-j-1) for j in window_size]
post_window ← [w(i+j+1) for j in windowsize]
context_words ← pre_window + post_window
~~context_words~~
centre_word ← w(i)

e.g

context [the, quick, jumped, over] centre → fox
context [quick, fox, over, the] centre → jumped
context [fox, ~~over~~ jumped, the, fence] centre → over.

What is the computational bottleneck operation in Word2Vec? What can you adopt other than negative sampling? Assume you decide to adopt 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 training will partially updates the model weights. (4 points)

- The computational bottleneck for Word2Vec is the Softmax operation. Although the computation is that of a dot product. But because for all pairs with the centre word we each time normalize over the entire vocabulary, it becomes very expensive and hence is the bottle neck.
- The 2 improvements other than Negative sampling for word2vec is 1) word pairs and phrases, where we prefer phrases of infrequent words. Although this increases the vocabulary size it decreases the training expense. 2) To Subsample frequent words to decrease no. of training examples.
- Negative Sampling defines a new objection function on top of the existing skip gram model which maximizes the similarity of in context words compared to out of context words which it aims to minimize. Instead of minimizing all words in the dictionary, the model causes each training sample to update only a small percentage of the model's weight. It randomly selects k negative samples and our one true sample and trains a logistic regression instead of a Softmax across the entire vocabulary.

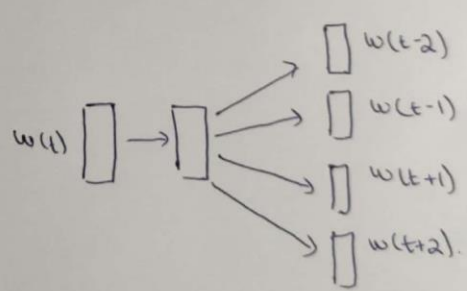
```
k=5
text = load_text_data()
new_text = [line.split() for line in text]

function get_samples(k)
        negword = []
        while negwords < k
                negwords.append(Vocab(random.randint))
                neglabels = [0] * length(negwords).
        end
        return negword, neglabels.
for N epochs
    for sentence in new_text
        for i, word in sentence.
                U, U_labels = get_samples(k)
                U = [getOHV(i) for i in U]
                C = word
                c_label = 1
                V = sentence[i+1]
```

$J(\Theta) = \log \sigma(\text{getOHV}(v)^T \cdot \text{getOHV}(c) + \sum_{i=1}^{k} E_{j \sim P(w)}[\log \sigma(-v[i]^T \cdot \text{getOHV}(c))$

$\Theta_{new} = \Theta_{old} - \eta \frac{\partial J(\Theta)}{\partial \Theta}$

* ideally $\frac{U(w)}{2}^{3/4}$ but we did random.

```
                end for
        end for
end for
```



input projection output

Skip gram

The quick brown fox . . . .

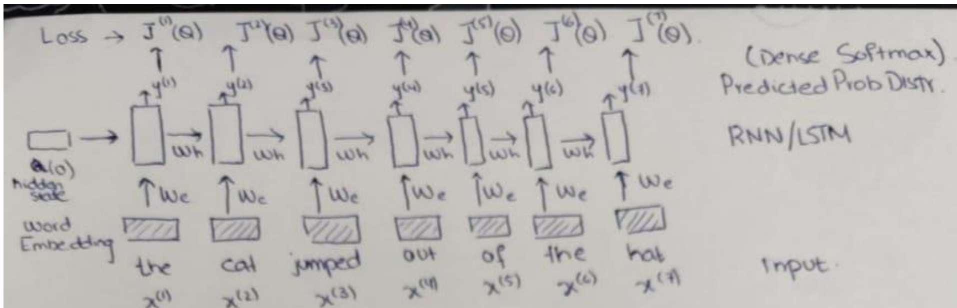
	context	centre	output
	quick	brown	1
	quick	king	0
	quick	brother	0
k=5	quick	fire	0
	quick	food	0
	quick	orange	0

* Changed multinomial to binary classification problem

* change from prediction of word to just one words neighbours?

Question 5

Please annotate an LSTM architecture to implement an 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 a regularization term in the NER cost function? Write a pseudo code showing how you handle different vector sizes and how you read LSTM output in NER.



$$J(\theta) = \frac{1}{m} \sum_{i=1}^{t} \left[-y^{(i)} \log(h_\theta(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]$$
$$+ \frac{c}{2m} \left[\sum_{j=1}^{nC} \sum_{k=1}^{H} W_{kj}^2 + \sum_{k=1}^{H} U_k^2 \right]$$

•) The regularization term helps control the weights, so that we avoid overfitting the model.

Handling Different Vector Sizes

```
text ← load_vectors()
text ← [~~text~~t.split() for t in text]
encodedText ← [[get_encoding(w) for w in line] for line in text]
max_length ← max(length(line) for line in encodedText)
train_data ← zeros(number of lines, max_length)
test_data ← -1 * ones(number of lines, max_length).
for i, line in encodedText:
    for j, w in line:
        train_data[i][j] = encodedText[i][j]
        test_data[i][j] = encodedTags[i][j]
    end
end
```

Redd LSTM output for NER

x_train, y-train ← prepare_data()

Length_arr ← [len(rows) for row in x_train] # list of all lengths.

masked_arr ← length_arr ≥ 0 # masking excludes padding which has -1 as labels.

~~padded~~ Packed-seq ← pack-padded_seq (masked_arr, length_arr)

Lstm_out ← LSTM(~~padded~~ packed seq)

unpacked_seq ← pad_packed_seq(Lstm_out) # unpack output

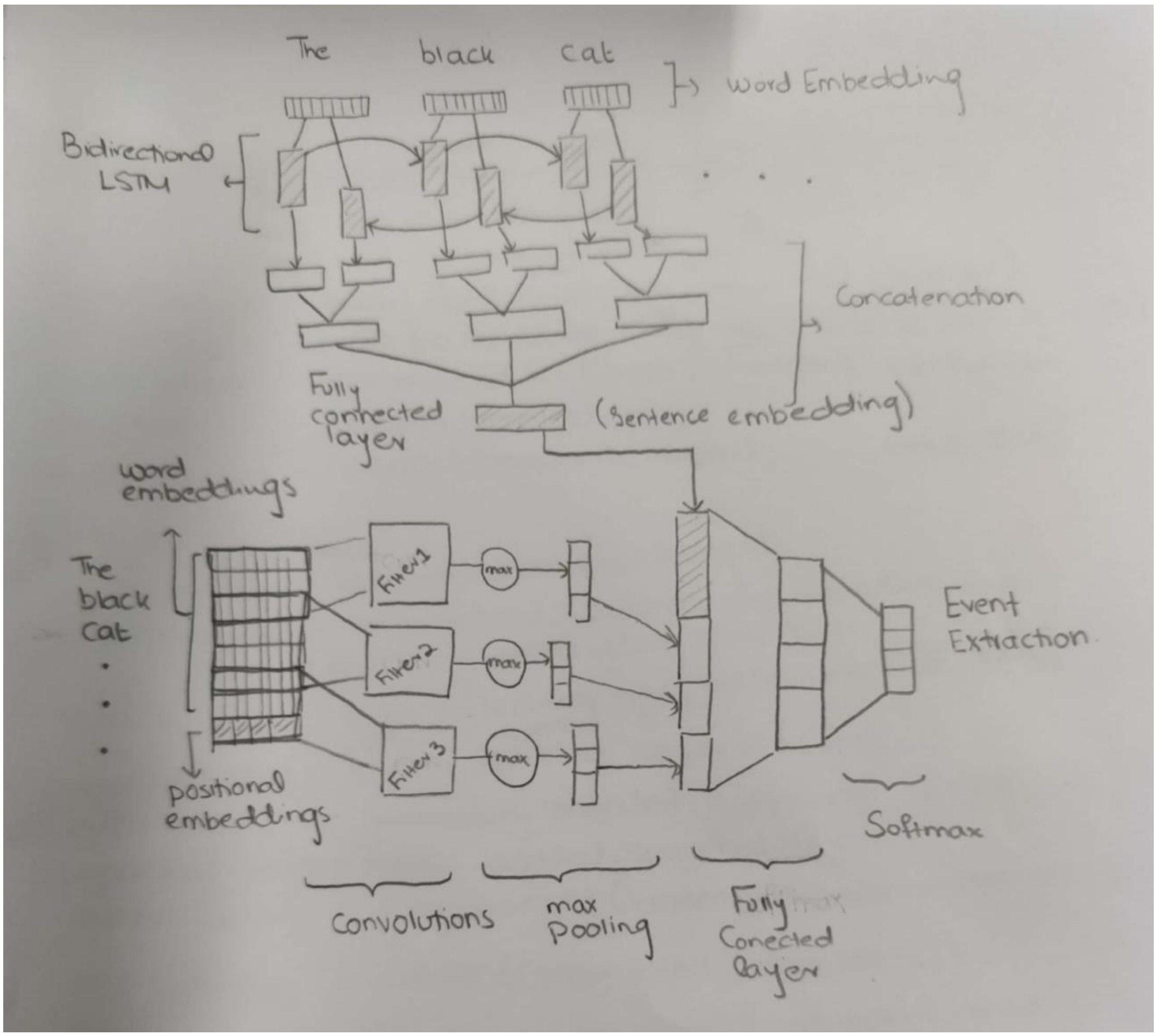
linear_out ← FullyConnectedLayer (unpacked_seq)

output_prob ← log_softmax (linear_out)

NER_Tag ← IDtoTag(max(output_prob))

Question 6

Sketch a block-based diagram and write a pseudo code 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 embedding, a positional embedding and (bonus) a retrofitting of embeddings to existing lexicon as input



CNN

word_embedding = load_pretrained_embeddings()
pos-emb 1 = get-pos-embedding (1)
pos-emb2 = get - pos - embedding(2)
embedding = concat (word_embedding, pos_emb1, pos_emb2)
convolution = Conv1D (embedding, num_kernels, kernel_size, padding, stride)
masked-conv = conv. masked_fill (mask, convolution) # for 'PAD' tokens.
max -pool = maxpool (masked_conv)
all-ma~~xked~~_pool = concat ([max-pool1; maxpool2; . . . . max-pool~~n~~;])

LSTM

word-embedding = load-pretrained-embedding()
packed-sequence = pack_padded_sequence (word_embedding, lengths)
pack sequence to ignore 'PAD' token.
Lstm 1 = LSTM (packed-sequence, hidden-units, ~~num_classes~~ embed_dim)
LSTM 2 = LSTM (" reversed ")
unpacked_seq1 = unpack-padded-sequence (Lstm1).
output-Lstm1 = ~~Linear~~ Fully connected (hidden-units, num_classes, unpacked-seq1)
Unpacked-seq2 = unpack_padded_sequence (LSTM2)
Output-Lstm2 = Fullyconnected (hidden-units, num_classes, unpacked_seq2)
output_concat = concat ([output_lstm1, output_lstm2])
Sentence-embedding = [concat (output_concat) for all output_concats]

JOINT

feature_vector = all_max_pool + sentence_embedding
Output-FC = Fullyconnected (output_FC, hidden-units, num_classes)
finaloutput = log_softmax(output-FC).