Natural Language Processing Course

CA#3 (POS & NER) Report

Spring 1401

Parnian Fazel – 810198516

Part 1: Part of Speech tagging

الف)

In this part I imported Penn Treebank dataset from nltk corpuses with universal tagset. I explain the difference between universal tagset and default tagset, by showing the result of the first sentence in Penn Treebank datset:

Default tagset	Universal tagset
[('Pierre', 'NNP'), ('Vinken', 'NNP'), (',', ','), ('61', 'CD'), ('years', 'NNS'), ('old', 'JJ'), (',', ','), ('will', 'MD'), ('join', 'VB'), ('the', 'DT'), ('board', 'NN'), ('as', 'IN'), ('a', 'DT'), ('nonexecutive', 'JJ'), ('director', 'NN'), ('Nov.', 'NNP'), ('29', 'CD'), ('.', '.')]	[('Pierre', 'NOUN'), ('Vinken', 'NOUN'), (',', '.'), ('61', 'NUM'), ('years', 'NOUN'), ('old', 'ADJ'), (',', '.'), ('will', 'VERB'), ('join', 'VERB'), ('the', 'DET'), ('board', 'NOUN'), ('as', 'ADP'), ('a', 'DET'), ('nonexecutive', 'ADJ'), ('director', 'NOUN'), ('Nov.', 'NOUN'), ('29', 'NUM'), ('.', '.')]

By details the default tagset of **Penn TreeBank** (**45** tags):

(All tag can be seen by following code):

```
print(nltk.help.upenn_tagset())
```

```
$: dollar
    $ -$ --$ A$ C$ HK$ M$ NZ$ S$ U.S.$ US$
'': closing quotation mark
    ' ''
(: opening parenthesis
    ( [ {
): closing parenthesis
    ) ] }
,: comma
    ,
--: dash
    --
.: sentence terminator
    . ! ?
:: colon or ellipsis
    : ; ...
CC: conjunction, coordinating
    & 'n and both but either et for less minus neither nor or plus so
    therefore times v. versus vs. whether yet
CD: numeral, cardinal
    mid-1890 nine-thirty forty-two one-tenth ten million 0.5 one forty-
    seven 1987 twenty '79 zero two 78-degrees eighty-four IX '60s .025
    fifteen 271,124 dozen quintillion DM2,000 ...
DT: determiner
    all an another any both del each either every half la many much nary
    neither no some such that the them these this those
EX: existential there
    there
FW: foreign word
    gemeinschaft hund ich jeux habeas Haementeria Herr K'ang-si vous
    lutihaw alai je jour objets salutaris fille quibusdam pas trop Monte
    terram fiche oui corporis ...
IN: preposition or conjunction, subordinating
    astride among uppon whether out inside pro despite on by throughout
    below within for towards near behind atop around if like until below
    next into if beside ...
JJ: adjective or numeral, ordinal
    third ill-mannered pre-war regrettable oiled calamitous first separable
    ectoplasmic battery-powered participatory fourth still-to-be-named
    multilingual multi-disciplinary ...
JJR: adjective, comparative
    bleaker braver breezier briefer brighter brisker broader bumper busier
    calmer cheaper choosier cleaner clearer closer colder commoner costlier
    cozier creamier crunchier cuter ...
JJS: adjective, superlative
    calmest cheapest choicest classiest cleanest clearest closest commonest
    corniest costliest crassest creepiest crudest cutest darkest deadliest
    dearest deepest densest dinkiest ...
LS: list item marker
    A A. B B. C C. D E F First G H I J K One SP-44001 SP-44002 SP-44005
    SP-44007 Second Third Three Two * a b c d first five four one six three
    two
```

```
MD: modal auxiliary
    can cannot could couldn't dare may might must need ought shall should
    shouldn't will would
NN: noun, common, singular or mass
    common-carrier cabbage knuckle-duster Casino afghan shed thermostat
    investment slide humour falloff slick wind hyena override subhumanity
    machinist ...
NNP: noun, proper, singular
    Motown Venneboerger Czestochwa Ranzer Conchita Trumplane Christos
    Oceanside Escobar Kreisler Sawyer Cougar Yvette Ervin ODI Darryl CTCA
    Shannon A.K.C. Meltex Liverpool ...
NNPS: noun, proper, plural
    Americans Americas Amharas Amityvilles Amusements Anarcho-Syndicalists
    Andalusians Andes Andruses Angels Animals Anthony Antilles Antiques
    Apache Apaches Apocrypha ...
NNS: noun, common, plural
    undergraduates scotches bric-a-brac products bodyguards facets coasts
    divestitures storehouses designs clubs fragrances averages
    subjectivists apprehensions muses factory-jobs ...
PDT: pre-determiner
    all both half many quite such sure this
POS: genitive marker
    ' 's
PRP: pronoun, personal
    hers herself him himself hisself it itself me myself one oneself ours
    ourselves ownself self she thee theirs them themselves they thou thy us
PRP$: pronoun, possessive
    her his mine my our ours their thy your
RB: adverb
    occasionally unabatingly maddeningly adventurously professedly
    stirringly prominently technologically magisterially predominately
    swiftly fiscally pitilessly ...
RBR: adverb, comparative
    further gloomier grander graver greater grimmer harder harsher
    healthier heavier higher however larger later leaner lengthier less-
    perfectly lesser lonelier longer louder lower more ...
RBS: adverb, superlative
    best biggest bluntest earliest farthest first furthest hardest
    heartiest highest largest least less most nearest second tightest worst
RP: particle
    aboard about across along apart around aside at away back before behind
    by crop down ever fast for forth from go high i.e. in into just later
    low more off on open out over per pie raising start teeth that through
    under unto up up-pp upon whole with you
SYM: symbol
    % & ' '' ''. ) ). * + ,. < = > @ A[fj] U.S U.S.S.R * ** ***
TO: "to" as preposition or infinitive marker
    to
UH: interjection
    Goodbye Goody Gosh Wow Jeepers Jee-sus Hubba Hey Kee-reist Oops amen
    huh howdy uh dammit whammo shucks heck anyways whodunnit honey golly
    man baby diddle hush sonuvabitch ...

VB: verb, base form
    ask assemble assess assign assume atone attention avoid bake balkanize
    bank begin behold believe bend benefit bevel beware bless boil bomb
    boost brace break bring broil brush build ...
VBD: verb, past tense
    dipped pleaded swiped regummed soaked tidied convened halted registered
    cushioned exacted snubbed strode aimed adopted belied figgered
    speculated wore appreciated contemplated ...
VBG: verb, present participle or gerund
    telegraphing stirring focusing angering judging stalling lactating
    hankerin' alleging veering capping approaching traveling besieging
    encrypting interrupting erasing wincing ...
VBN: verb, past participle
    multihulled dilapidated aerosolized chaired languished panelized used
    experimented flourished imitated reunifed factored condensed sheared
    unsettled primed dubbed desired ...
VBP: verb, present tense, not 3rd person singular
    predominate wrap resort sue twist spill cure lengthen brush terminate
    appear tend stray glisten obtain comprise detest tease attract
    emphasize mold postpone sever return wag ...
VBZ: verb, present tense, 3rd person singular
    bases reconstructs marks mixes displeases seals carps weaves snatches
    slumps stretches authorizes smolders pictures emerges stockpiles
    seduces fizzes uses bolsters slaps speaks pleads ...
WDT: WH-determiner
    that what whatever which whichever
WP: WH-pronoun
    that what whatever whatsoever which who whom whosoever
WP$: WH-pronoun, possessive
    whose
WRB: Wh-adverb
    how however whence whenever where whereby whereever wherein whereof why
``: opening quotation mark
    ` ``

None
```

There is also a table of these tags in here.[1]

By details the **universal tagset** (**12** tags) is a simplified tagset:

Universal Part-of-Speech Tagset

Tag	Meaning	English Examples
ADJ	adjective	*new, good, high, special, big, local*
ADP	adposition	*on, of, at, with, by, into, under*
ADV	adverb	*really, already, still, early, now*
CONJ	conjunction	*and, or, but, if, while, although*
DET	determiner, article	*the, a, some, most, every, no, which*
NOUN	noun	*year, home, costs, time, Africa*
NUM	numeral	*twenty-four, fourth, 1991, 14:24*
PRT	particle	*at, on, out, over per, that, up, with*
PRON	pronoun	*he, their, her, its, my, I, us*
VERB	verb	*is, say, told, given, playing, would*
.	punctuation marks	*. , ; !*
X	other	*ersatz, esprit, dunno, gr8, univeristy*

* X – other consists of foreign words, typos, abbreviations, etc.

The main difference of these two tagsets is how much they show the **details** of a POS tag. Consider a noun tag; by the universal tagset we only can say "NOUN", but by the default tagset there are "NN", "NNP", "NNS", "NNPS"! So, by the default tagset we can have more information about tags and use it to be more accurate. However, the universal tagset is useful too since it is more abstract and can be used in tasks where we do not need to be so specific about tags.

I will use universal tagset in this part:

```
set_tagset:
 self.ptb = list(treebank.tagged_sents(tagset='universal'))
```

ب)

In the notebook file, I defined PTBPOSLoader and POSTagger classes which are related to part 1.

I splitted train and test sets by **80% train** & **10% test set**, and **10%** of train set for **validation set (8% of all data)**. The numbers are:

[1] https://www.sketchengine.eu/penn-treebank-tagset/

```
train set: 3169
validation set: 353
test set: 392
```

پ)

A **Markov chain** is a model that tells us something about the probabilities of sequences of random states. A Markov chain makes a very strong assumption that if we want to predict the future in the sequence, all that matters is the **current state**.

A **hidden Markov model (HMM)** allows us to talk about both **observed events** (like words that we see in the input) and **hidden events** (like part-of-speech tags) that we think of as causal factors in a probabilistic model.

HMM has five components:

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} \ldots a_{ij} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, \ldots, v_V$
$B = b_i(o_t)$	a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state i
$\pi = \pi_1, \pi_2, \ldots, \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

The A matrix (**transition matrix**) contains the tag transition probabilities $P(t_i|t_{i-1})$ which represent the probability of a tag occurring given the previous tag. As an example: Calculating A[Verb][Noun]:

$$P(Noun|Verb) = \frac{count(Noun\ \&\ Verb)}{count(Verb)}$$

The transition matrix:

	.	ADJ	ADP	ADV	CONJ	DET	NOUN	NUM	PRON	PRT	VERB	X
.	0.092753	0.044369	0.091274	0.050708	0.057997	0.172090	0.225650	0.081978	0.065709	0.002324	0.087999	0.027044
ADJ	0.064417	0.067310	0.077917	0.004822	0.016008	0.004243	0.699711	0.021986	0.000579	0.010415	0.011958	0.020636
ADP	0.039820	0.106104	0.017601	0.013107	0.000874	0.324554	0.323805	0.062414	0.068281	0.001498	0.008239	0.033704
ADV	0.132597	0.130229	0.115627	0.074191	0.008287	0.072218	0.030781	0.034333	0.014207	0.014207	0.350829	0.022494
CONJ	0.036158	0.121964	0.052348	0.053427	0.000540	0.118187	0.349703	0.045332	0.055586	0.003778	0.154884	0.008095
DET	0.017539	0.206082	0.010184	0.012871	0.000141	0.005941	0.635785	0.021216	0.003678	0.000283	0.041443	0.044837
NOUN	0.240111	0.012234	0.176854	0.016922	0.043393	0.013640	0.264834	0.009079	0.004689	0.042754	0.147059	0.028431
NUM	0.117182	0.031959	0.035052	0.003093	0.013058	0.003436	0.353952	0.190378	0.001375	0.023368	0.016838	0.210309
PRON	0.038391	0.074954	0.023309	0.035192	0.005941	0.009598	0.207495	0.006856	0.006856	0.011426	0.483547	0.096435
PRT	0.044583	0.080563	0.018381	0.009777	0.002346	0.104028	0.245209	0.055925	0.017208	0.001955	0.406336	0.013688
VERB	0.034388	0.064763	0.092310	0.082733	0.005473	0.133266	0.110645	0.023716	0.034480	0.031013	0.168020	0.219192
X	0.160591	0.017386	0.145635	0.024491	0.010469	0.055524	0.064124	0.002617	0.056085	0.183773	0.204711	0.074593

The B **emission probabilities** $P(w_i|t_i)$ represent the probability, given a tag (say Verb), that it will be associated with a given word (say bowed). The emission probability B[Verb][bowed] is calculated using:

$$P(bowed|Verb) = \frac{count(bowed\ \&\ Verb)}{count(Verb)}$$

* I saved the emission matrix in a csv file named “emission_ptb.csv” to avoid spending time on creating a same emission matrix every time. Pay attention that this csv file is made for the train, validation and test split by 82% train, 8% validation and 10% test.

The Algorithm of Viterbi:

function smoothed_viterbi(tagged_words, transition_matrix, emission_matrix) returns best tags

```
create an array Viterbi
for word in tagged_words:
        max_prob = 0
        best_tag  = None
        for tag in all_tagset:
```

```
                if word is the first word:
                        trans = transition_matrix['.', tag]
                else:
                        last_state = last state of Viterbi
                        trans = transition_matrix[last_state][tag]
                if word is in Vocabulary:
                        emiss = emission_matrix[word][tag]
                else://OOV
                        emiss = 0.0001
                prob = trans * emis
                If prob > max_prob:
                        max_prob = prob
                        best_tag = tag
        Viterbi.append(best_tag)

return tagged_words, Viterbi
```

Here I could add a <s> tag as a first tag of all sentences in dataset, but instead I used the "." Character to do this, because this actually shows the end of a sentence. Also, since the Viterbi algorithm is stochastic, we can consider all words in a sequence than all sentences separately.

Accuracy:

- Vanilla Viterbi: `0.8985163816196168`
- Smoothed Vitrebi: `0.9325159695034`

The 93% accuracy shows this Viterbi algorithm works acceptable.

ت)

10 wrong tagged words:

```
word: book | predicted: NOUN | expected: VERB
word: stocks | predicted: NOUN | expected: ADV
word: up | predicted: PRT | expected: ADP
word: fractionally | predicted: VERB | expected: ADV
word: over | predicted: ADP | expected: PRT
word: mine | predicted: NOUN | expected: ADJ
word: Palestinian | predicted: ADJ | expected: NOUN
word: first | predicted: ADJ | expected: ADV
word: clamped | predicted: X | expected: VERB
word: ankle | predicted: VERB | expected: NOUN
```

Most of these words do not always have a unique pos tag so these mistakes are acceptable. For instance, “book” or “ankle” is noun in some cases and in some cases, it is a verb. Imagine: I wrote this book (noun) or I booked a flight (verb). Or consider word clamped, since it is not a common verb the algorithm predicted as X tag it means it did not recognize it or took it as a typo maybe, but in fact it’s a verb. If we used a rule-based approach, we could have tagged this word as a verb because it has “ed” at the end of it.

ث)

I employed smoothing for unknown words. Therefore, I used 0.0001 as emission probability for unknown words which have 0 emission probability. In order to choose a number, I tested 0.1, 0.001 and 0.0001 which 0.001 resulted the highest accuracy.

There are other ways as well. I mention some of them here:

- ✓ Using morphological information like word ending with “ed” or “s” or “able”.
- ✓ Use a lot more data to decrease the chance of having OOV words. (however, we should use smoothing here too)
- ✓ We can consider these words as group of words (for example using bigrams or trigrams) to get information about the unknown word and set a probability for it based on the neighbored words.

ج و ح)

In the notebook file, I defined RecurrentPOSTagger class which is related to this part.

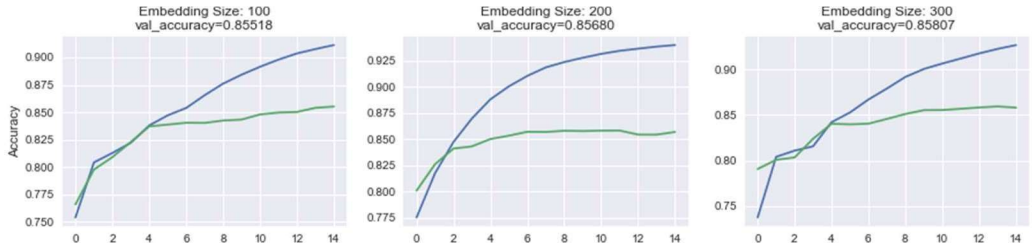
Hyperparameters which I studied in this part:

- Batch size
- Units
- Embedding Size
- Max sequence length (for padding)

First, I studied the effect of these hyper parameters on the accuracy:

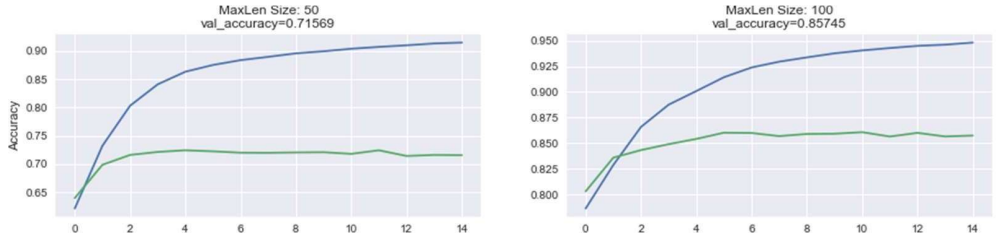
RNN

- Embedding size



- Batch size

- Max sequence length (for padding)

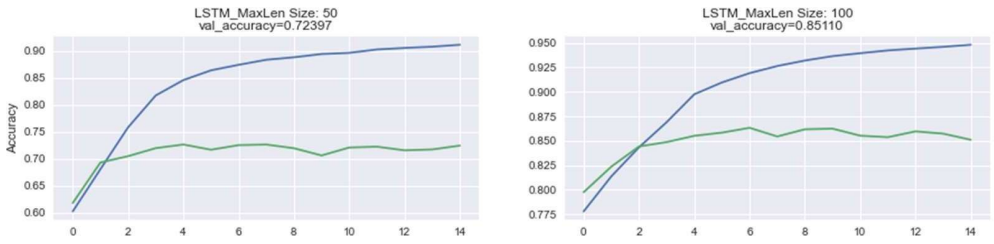


LSTM

- Embedding size

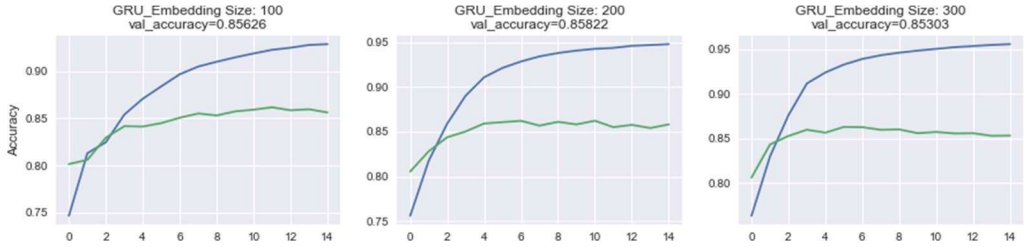
- Batch size

- Max sequence length (for padding)



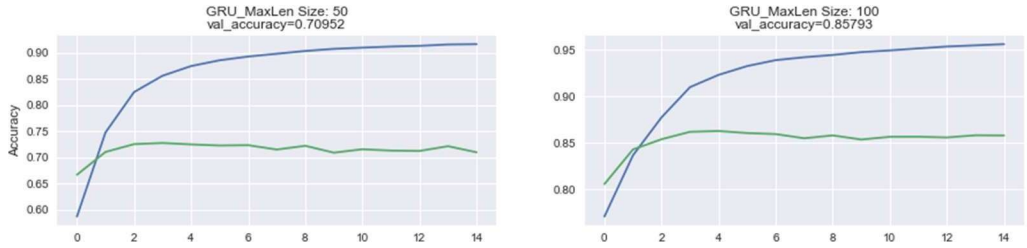
GRU

- Embedding size



- Batch size

- Max sequence length (for padding)



After observing these effects and testing the combination of these parameters, I tested these parameters by some values for RNN, LSTM and GRU. The results are in a txt file named “tuning_results.txt”.
some of the results can be seen here:

- RNN:

```
--------------> LAYER TYPE: RNN <--------------
Layer: RNN, Batch size: 16, Padding Max length: 50, Embedding size: 100
val_accuracy=0.71428
------------------------------
Layer: RNN, Batch size: 16, Padding Max length: 50, Embedding size: 300
val_accuracy=0.70669
------------------------------
Layer: RNN, Batch size: 16, Padding Max length: 100, Embedding size: 100
val_accuracy=0.85470
------------------------------
Layer: RNN, Batch size: 16, Padding Max length: 100, Embedding size: 300
val_accuracy=0.85782
------------------------------
Layer: RNN, Batch size: 32, Padding Max length: 50, Embedding size: 100
val_accuracy=0.71025
------------------------------
Layer: RNN, Batch size: 32, Padding Max length: 50, Embedding size: 300
val_accuracy=0.72261
------------------------------
Layer: RNN, Batch size: 32, Padding Max length: 100, Embedding size: 100
val_accuracy=0.83895
------------------------------
Layer: RNN, Batch size: 32, Padding Max length: 100, Embedding size: 300
val_accuracy=0.85643
------------------------------
Layer: RNN, Batch size: 64, Padding Max length: 50, Embedding size: 100
val_accuracy=0.69184
------------------------------
Layer: RNN, Batch size: 64, Padding Max length: 50, Embedding size: 300
val_accuracy=0.72261
------------------------------
Layer: RNN, Batch size: 64, Padding Max length: 100, Embedding size: 100
val_accuracy=0.82983
------------------------------
Layer: RNN, Batch size: 64, Padding Max length: 100, Embedding size: 300
val_accuracy=0.85190
```

- LSTM

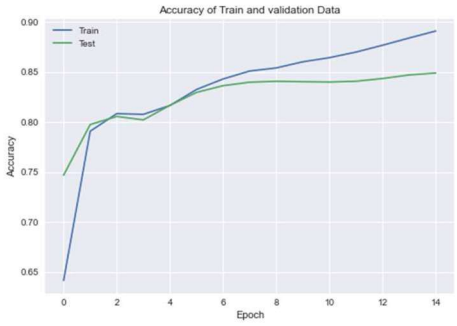
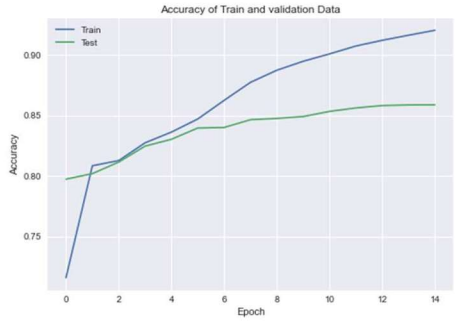
```
--------------> LAYER TYPE: LSTM <--------------
Layer: LSTM, Batch size: 16, Padding Max length: 50, Embedding size: 100
val_accuracy=0.72306
------------------------------
Layer: LSTM, Batch size: 16, Padding Max length: 50, Embedding size: 300
val_accuracy=0.72363
------------------------------
Layer: LSTM, Batch size: 16, Padding Max length: 100, Embedding size: 100
val_accuracy=0.85592
------------------------------
Layer: LSTM, Batch size: 16, Padding Max length: 100, Embedding size: 300
val_accuracy=0.86079
------------------------------
Layer: LSTM, Batch size: 32, Padding Max length: 50, Embedding size: 100
val_accuracy=0.71082
------------------------------
Layer: LSTM, Batch size: 32, Padding Max length: 50, Embedding size: 300
val_accuracy=0.72884
------------------------------
Layer: LSTM, Batch size: 32, Padding Max length: 100, Embedding size: 100
val_accuracy=0.84207
------------------------------
Layer: LSTM, Batch size: 32, Padding Max length: 100, Embedding size: 300
val_accuracy=0.86295
------------------------------
Layer: LSTM, Batch size: 64, Padding Max length: 50, Embedding size: 100
val_accuracy=0.68799
------------------------------
Layer: LSTM, Batch size: 64, Padding Max length: 50, Embedding size: 300
val_accuracy=0.72827
------------------------------
Layer: LSTM, Batch size: 64, Padding Max length: 100, Embedding size: 100
val_accuracy=0.80578
------------------------------
Layer: LSTM, Batch size: 64, Padding Max length: 100, Embedding size: 300
val_accuracy=0.84377
------------------------------
```

- GRU

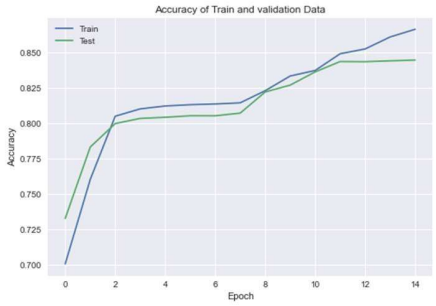
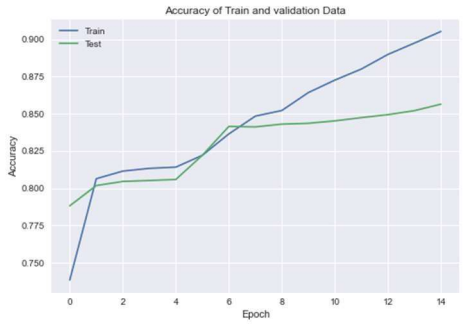
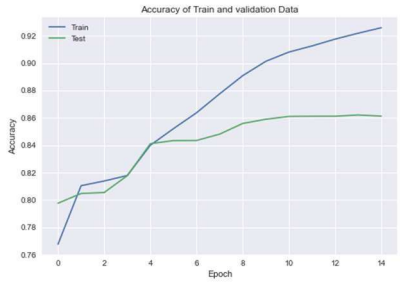
```
--------------> LAYER TYPE: GRU <--------------
Layer: GRU, Batch size: 16, Padding Max length: 50, Embedding size: 100
val_accuracy=0.71813
------------------------------
Layer: GRU, Batch size: 16, Padding Max length: 50, Embedding size: 300
val_accuracy=0.71008
------------------------------
Layer: GRU, Batch size: 16, Padding Max length: 100, Embedding size: 100
val_accuracy=0.85788
------------------------------
Layer: GRU, Batch size: 16, Padding Max length: 100, Embedding size: 300
val_accuracy=0.85935
------------------------------
Layer: GRU, Batch size: 32, Padding Max length: 50, Embedding size: 100
val_accuracy=0.71756
------------------------------
Layer: GRU, Batch size: 32, Padding Max length: 50, Embedding size: 300
val_accuracy=0.71830
------------------------------
Layer: GRU, Batch size: 32, Padding Max length: 100, Embedding size: 100
val_accuracy=0.85388
------------------------------
Layer: GRU, Batch size: 32, Padding Max length: 100, Embedding size: 300
val_accuracy=0.86037
------------------------------
Layer: GRU, Batch size: 64, Padding Max length: 50, Embedding size: 100
val_accuracy=0.70924
------------------------------
Layer: GRU, Batch size: 64, Padding Max length: 50, Embedding size: 300
val_accuracy=0.73014
------------------------------
Layer: GRU, Batch size: 64, Padding Max length: 100, Embedding size: 100
val_accuracy=0.84153
------------------------------
Layer: GRU, Batch size: 64, Padding Max length: 100, Embedding size: 300
val_accuracy=0.85887
------------------------------
```

Now I consider 3 unit values:

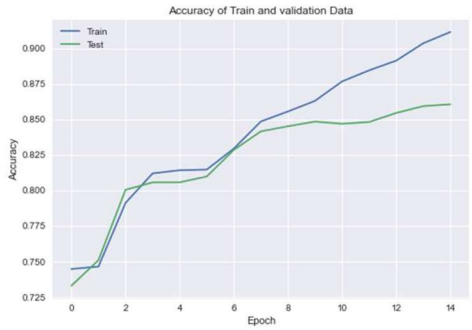
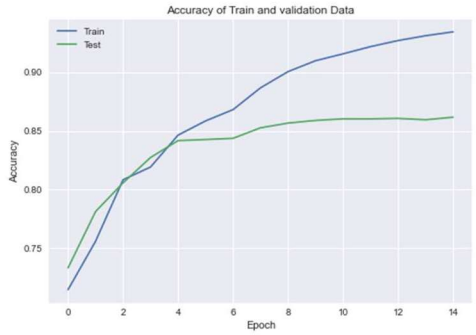
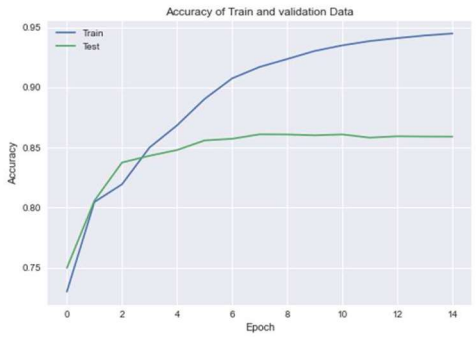
- RNN

Unit	result	plot
32	accuracy: 0.89115 val accuracy: 0.84915	
64	accuracy: 0.92054 val accuracy: 0.85884	
128	accuracy: 0.93234 val accuracy: 0.85904	

- LSTM

Unit	result	plot
32	accuracy: 0.86658 val accuracy: 0.84487	
64	accuracy: 0.90531 val accuracy: 0.85649	
128	accuracy: 0.92591 val accuracy: 0.86130	

- GRU

Unit	result	plot
32	accuracy: 0.91142 val accuracy: 0.86057	
64	accuracy: 0.93469 val accuracy: 0.86176	
128	accuracy: 0.94491 val accuracy: 0.85907	

I also tested **bidirecetional** format of rnn but did not make a noticeable difference on my model.

Why we use validation set? Using **validation set** give us the ability to tune our hyperparameters without bias, and to evaluate the performance of our model for different combinations of hyperparameter values, then use the test data set for the final test of our model.

ج و ح)

After tuning the hyperparameters, I finally tested the best model on test set.
(epoch = 12)

- RNN

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_212 (Embedding)    (None, 100, 300)          3339900

simple_rnn_165 (SimpleRNN)   (None, 100, 128)          54912

time_distributed_211 (TimeD  (None, 100, 13)           1677
istributed)

    -
accuracy: 0.92232
```

Best model hyperparameters:

Batch size	Units	Embedding Size	Padding max sequence length
64	128	300	100

- LSTM

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_215 (Embedding)    (None, 100, 300)          3339900

lstm_24 (LSTM)               (None, 100, 64)           93440

time_distributed_214 (TimeD  (None, 100, 13)           845
istributed)
```

```
accuracy: 0.93046
```

Batch size	Units	Embedding Size	Padding max sequence length
32	64	300	100

- GRU

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_216 (Embedding)    (None, 100, 300)          3339900

gru_23 (GRU)                 (None, 100, 64)           70272

time_distributed_215 (TimeD  (None, 100, 13)           845
istributed)

accuracy: 0.94115
```

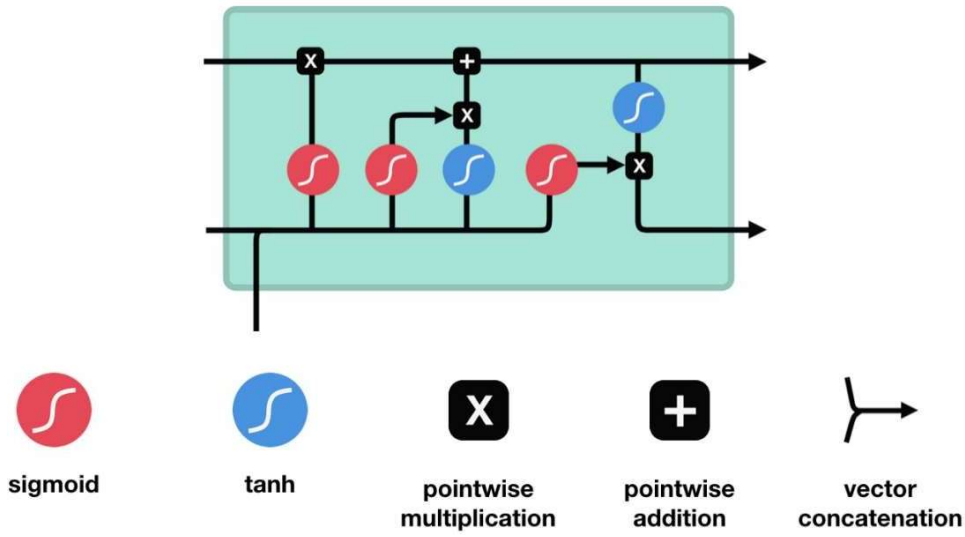
Batch size	Units	Embedding Size	Padding max sequence length
32	64	300	100

ج)

The accuracies can be ordered as: $\text{RNN} < \text{LSTM} < \text{GRU}$

As we can see model with RNN got the lowest accuracy, and this is actually expected because LSTM and GRU has more complex structure for gates and they have a longer memory than RNN so at it is expected LSTM and GRU are more powerful than RNN. RNN can show a better performance on shorter sequences. LSTM and GRU accuracies are very close. Generally, we can't tell surly which of them always perform better. As said in the course lectures, GRU has a simpler structure than LSTM but sometimes can give better results based on this simpler structure. I can also mention that GRU has a better performance since it is simpler.

LSTM:



LSTMs use a series of gates which control how the information in a sequence of data comes into, is stored in and leaves the network. The core concept of LSTM is the **cell state**, and it's various gates. The cell state act as a transport highway that transfers relative information all the way down the sequence chain. This is somehow the "memory" of the network. The cell state, in theory, can carry relevant information throughout the processing of the **sequence**. The gates can learn what information is relevant to **keep or forget** during training.

There are three gates in a typical LSTM:

1. **Input gate**: Input gate is used to quantify the importance of the new information carried by the input and determines what new information should be added to the networks long-term memory (cell state), given the previous hidden state and new input data. The sigmoid output will decide which information is important to keep from the tanh output.

Input Gate:

- $i_t = \sigma(x_t * U_i + H_{t-1} * W_i)$

2. **Output gate**: The hidden state contains information on previous inputs and is used for prediction. The output gate regulates the present hidden state. The previous hidden state and current input are passed to the sigmoid function. This output is multiplied with the output of the tanh function to obtain the present hidden state. The current state and present hidden state are the final outputs from a classic LSTM unit.

Output Gate:

- $o_t = \sigma(x_t * U_o + H_{t-1} * W_o)$

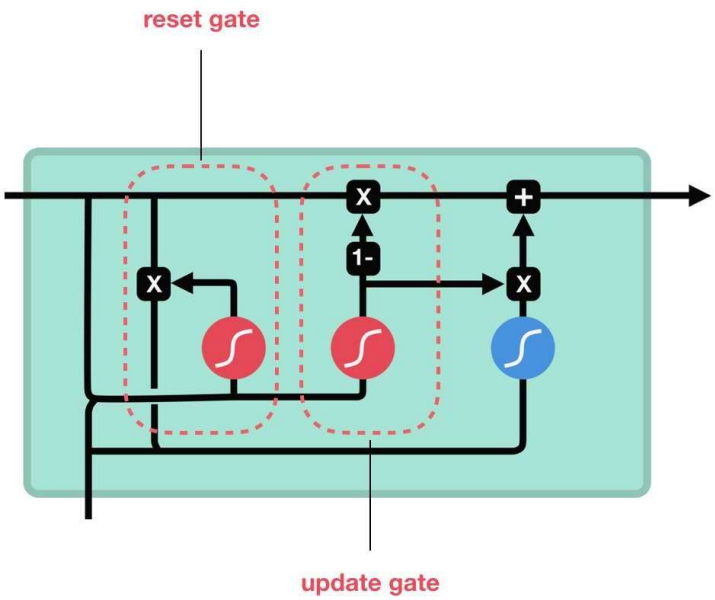
3. **Forget gate**: the forget gate decides which pieces of the long-term memory should now be forgotten (have less weight) given the previous hidden state and the new data point in the sequence.

Forget Gate:

- $f_t = \sigma(x_t * U_f + H_{t-1} * W_f)$

So, to sum up the Forget gate decides what is relevant to keep from prior steps. The input gate decides what information is relevant to add from the current step. The output gate determines what the next hidden state should be. These gates can be thought of as filters and are each their own neural network.

GRU:



The GRU is the newer generation of Recurrent Neural networks and is pretty similar to an LSTM. GRU does not have cell state and used the hidden state to transfer information.

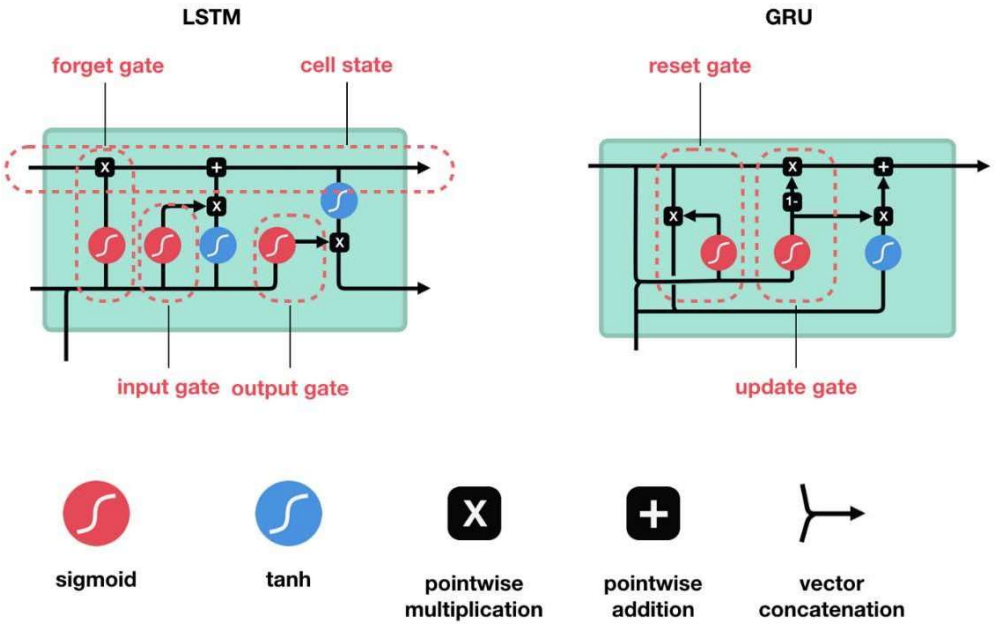
It has two gates:

1. **Reset gate**: The reset gate is another gate is used to decide how much past information to forget.
2. **Update gate**: The update gate acts similar to the forget and input gate of an LSTM. It decides what information to throw away and what new information to add.

LSTM vs GRU:

GRU is related to LSTM as both are utilizing different way of gating information to prevent vanishing gradient problem.

Here we can see a general difference of gates between LSTM and GRU:



As discussed in lectures and explained here, a GRU has **two gates**, an LSTM has **three gates**, GRUs don't possess and internal memory, also, the input and forget gates in LSTM are coupled by an update gate and the reset gate. Thus, the responsibility of the reset gate in a LSTM is really split up into both reset and

update gate. GRU's has fewer tensor operations; therefore, they are a little speedier to train then LSTM's. There isn't a clear winner which one is better.

The GRU controls the flow of information like the LSTM unit, but without having to use a memory unit. It just exposes the full hidden content without any control. Also, GRU is computationally more efficient.

د)

The best accuracy belongs to GRU which is 94.11%. the accuracy of Viterbi was 93.2%. Both of them have very good accuracies but GRU is better since it is recurrent and has a memory. Also, GRU is a network so it can use embedding layer, use and pass information bidirectionally and it can has layers to make it more powerful. However, Viterbi algorithm is faster than GRU (given emission and transition matrix). For instance Viterbi algorithm took about 6 seconds but GRU for 12 epochs took around a minute. Viterbi algorithm gave same result as LSTM but is was better than RNN. We can conclude that Viterbi is a powerful algorithm and it can compete with Recurrent Networks.

Part 2: Named entity Recognition

I defined a "NER" class in notebook file which is related to this part.

الف)

Explanation of IOB tag: The **BIO / IOB** format is short for **inside, outside, beginning**. The **B-** prefix before a tag indicates that the tag is the **beginning** of a chunk, and an **I-** prefix before a tag indicates that the tag is **inside** a chunk. An **O** tag indicates that a token belongs to no entity. This link can help understand it better.

I used the nltk.chunk package and the tree2conlltags function, to tag Penn TreeBank data by IOB format:

```
all_tags = []
word_ne = []
for record in ptb_dataset:
    tree = nltk.ne_chunk(record)
    iob_tags = tree2conlltags(tree)
    tags = [pair[2] for pair in iob_tags]
    word_ne = word_ne + [(i[0], i[2]) for i in iob_tags]
```

We can see the output of tree2conlltags for a sample in PTB dataset:

```
[('Mr.', 'NNP', 'B-PERSON'),
 ('Vinken', 'NNP', 'B-PERSON'),
 ('is', 'VBZ', 'O'),
 ('chairman', 'NN', 'O'),
 ('of', 'IN', 'O'),
 ('Elsevier', 'NNP', 'B-ORGANIZATION'),
 ('N.V.', 'NNP', 'O'),
 (',', ',', 'O'),
 ('the', 'DT', 'O'),
 ('Dutch', 'NNP', 'B-GPE'),
 ('publishing', 'VBG', 'O'),
 ('group', 'NN', 'O'),
 ('.', '.', 'O')]
```

Tagsets (IOB)

```
tagset size: 13
tagsets in train set: ['B-FACILITY', 'B-GPE', 'B-GSP', 'B-LOCATION', 'B-ORGANIZATION', 'B-PERSON', 'I-FACILITY', 'I-G
PE', 'I-GSP', 'I-LOCATION', 'I-ORGANIZATION', 'I-PERSON', 'O']
```

ب)

I splitted the data by **85%** train set and **15%** test set.

We need to change the Viterbi used in part 1:

- ✓ Creating a new transition matrix (for 13 tags I mentioned above)
- ✓ Creating a new emission matrix
- ✓ Adding conditions to avoid some sequences of tags. For instance, we know that an “O” tag can not be followed by “I” tag, or The “B” tag is used only when a tag is followed by a tag of the same type without “O” tokens between them.

* Pay attention that I did not the Viterbi algorithm, I just created a new emission and transition matrix.

Here is how I handled the impossible sequences of tags:

```
    for i, t1 in enumerate(list(self.ne_tagsets)):
        for j, t2 in enumerate(list(self.ne_tagsets)):
            self.ne_tags_matrix[i, j] = self.get_transition(t2, t1)
            if "I-" in t2:
                if ("B-" in t1 and t1[2:] != t2[2:]) or ("I-" in t1 and t1[2:] != t2[2:]) or "O" in t1:
                    transition_p = 0
```

Here is the transition matrix:

	B-FACILITY	B-GPE	B-GSP	B-LOCATION	B-ORGANIZATION	B-PERSON	I-FACILITY	I-GPE	I-GSP	I-LOCATION	I-ORGANIZATION	I-PERSON	O
B-FACILITY	0.000000	0.000000	0.000000	0.000000	0.020408	0.020408	0.000000	0.000000	0.000000	0.000000	0.000000	0.040816	0.918367
B-GPE	0.001764	0.015873	0.000588	0.000000	0.014697	0.021752	0.001764	0.002352	0.000000	0.000000	0.007055	0.012934	0.921223
B-GSP	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
B-LOCATION	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.037037	0.000000	0.000000	0.000000	0.000000	0.962963
B-ORGANIZATION	0.000000	0.028369	0.000000	0.000000	0.019858	0.015603	0.000000	0.000709	0.000000	0.000000	0.009220	0.011348	0.914894
B-PERSON	0.000526	0.015781	0.000000	0.000000	0.017885	0.019463	0.000526	0.002104	0.000000	0.000526	0.009469	0.012625	0.921094
I-FACILITY	0.000000	0.000000	0.000000	0.000000	0.020408	0.020408	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.959184
I-GPE	0.000000	0.012397	0.000000	0.000000	0.012397	0.033058	0.000000	0.000000	0.000000	0.000000	0.012397	0.012397	0.917355
I-GSP	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000
I-LOCATION	0.000000	0.000000	0.000000	0.000000	0.000000	0.034483	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.965517
I-ORGANIZATION	0.001252	0.021277	0.000000	0.000000	0.015019	0.012516	0.001252	0.002503	0.000000	0.000000	0.002503	0.016270	0.927409
I-PERSON	0.000000	0.021812	0.000839	0.000000	0.013423	0.016779	0.000839	0.001678	0.000000	0.000000	0.007550	0.012584	0.924497
O	0.000529	0.018733	0.000457	0.000325	0.015511	0.021210	0.000517	0.002741	0.000024	0.000337	0.008922	0.013190	0.917491

* I saved the emission matrix in a csv file named “NER_emission_15.csv” to avoid spending time on creating a same emission matrix every time. Pay attention that this csv file is made for the train and test split by 85% train, 15% test.

پ)

NER Report

accuracy = 0.9588134021983843

	precision	recall	f1-score	support
B-FACILITY	0.80	0.89	0.84	9
B-GPE	0.82	0.70	0.76	294
B-GSP	0.50	0.33	0.40	3
B-LOCATION	0.80	0.80	0.80	10
B-ORGANIZATION	0.70	0.50	0.58	241
B-PERSON	0.67	0.57	0.61	276
I-FACILITY	0.75	0.67	0.71	9
I-GPE	0.78	0.74	0.76	34
I-LOCATION	0.40	0.40	0.40	5
I-ORGANIZATION	0.47	0.49	0.48	129
I-PERSON	0.51	0.36	0.42	191
O	0.98	0.99	0.99	13901
accuracy			0.96	15102
macro avg	0.68	0.62	0.65	15102
weighted avg	0.95	0.96	0.96	15102

Based on this report, number of record for classes are imbalanced and it leads to biased results. For instance “O” tag has 13901 records and the f1-score for this tag is 99% but say I-Location has only 5 records and the results for this class is not good.

ت)

Yes, Recurrent Neural Nets can be used to recognize named entities: RNN, LSTM, Bidirectional LSTM…

According to the lectures, recurrent approaches doesn’t guarantee that the resulting sequence will be **well-formed**, means does not guarantee to prevents an output sequence from containing an “I” following an “O” and nothing would prevent an I-LOC tag from following a B-PER tag. To solve this problem we use an algorithm like Viterbi or CRF instead of argmax in the final layer to check the conditions of tag sequences.

Resources:

http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/hmms.pdf

https://www.freecodecamp.org/news/a-deep-dive-into-part-of-speech-tagging-using-viterbi-algorithm-17c8de32e8bc/

https://www.cs.utexas.edu/~gdurrett/courses/sp2021/lectures/lec4-4pp.pdf

https://personal.utdallas.edu/~sanda/courses/NLP/Lecture7.pdf

https://medium.com/analytics-vidhya/parts-of-speech-pos-and-viterbi-algorithm-3a5d54dfb346

https://www.atlantis-press.com/journals/ijcis/125941274/view

https://www.semanticscholar.org/paper/Named-Entity-Recognition-using-Hidden-Markov-Model-Morwal-Jahan/95284b31f27b9b8901fdc18554603610ebbc2752?p2df

https://analyticsindiamag.com/guide-to-named-entity-recognition-with-spacy-and-nltk/

*** For running the code, in case you want to read the emission matrix for Viterbi algorithm make sure you have “emission_ptb.csv” and “NER_emission_15.csv” files.