CS671A: Introduction to Natural Language Processing
End-semester exam

Time: 3 hours 24-Apr.-2018
Max marks: 140

1. *Answer all 7 questions. The question paper has 4 pages.*
2. **You can refer only to your handwritten class notes. No other paper or electronic documents or gadgets are allowed.**
3. *Keep your answers precise and brief.*
4. **Do all parts of a question together. Do not scatter answers to parts of the same question at different places in the answer script.**

1. (a) You are given the following data: vocabulary $\mathcal{V} = \{w_1, w_2, w_3\}$ and the bigram probability distribution p on $\mathcal{V} \times \mathcal{V}$ given by:
$p(w_1, w_1) = p(w_3, w_3) = 1/4$, $p(w_2, w_2) = 0$, $p(w_2, w_1) = 1/8$, $p(w_1, w_3) = 1/4$, $p(w_1, \star) = 1/2$ (that is w_1 as the first of a pair),$p(\star, w_2) = \frac{1}{8}$.
Calculate $p(w_1, w_2)$ and $p(w_2 \mid w_3)$

Solution:
The easiest is to fill in the matrix below for $p(.,.)$:

	w_1	w_2	w_3	$(w_i, \star)$
w_1	$1/4$	**0**	$1/4$	$1/2$
w_2	$1/8$	0	**0**	
w_3	**0**	**1/8**	$1/4$	
$(\star, w_j)$		$1/8$		

The bold entries are inferred, the others are given.
We have $(w_i, \star)=\sum_{j=1}^{3} p(w_i, w_j)$ and $(\star, w_j)= \sum_{i=1}^{3} p(w_i, w_j)$. The first equation makes $p(w_1, w_2) = 0$.
This makes $p(w_3, w_2) = 1/8$ from the second equation. The rest of the entries in the top-left 3×3 sub-matrix become 0 since all the 9 entries should add up to 1.
Now $p(w_2|w_3)$ is agnostic towards bigrams w_2w_3 and w_3w_2 which are also mutually exclusive so the we have to calculate contributions from $p(w_2, w_3)$ and $p(w_3, w_2)$ both ordered. However, $p(w_2, w_3) = 0$ so we only need to find contribution from $p(w_3, w_2)$.
$p(w_3, w_2) = p(w_2|w_3)p(w_3) = p(w_2|w_3)p(w_3, *)$. So, $p(w_2|w_3) = \frac{p(w_3,w_2)}{p(w_3,\star)} = \frac{1/8}{3/8} = \frac{1}{3}$

(b) How can you use a naive Bayes classifier to do word sense disambiguation? Clearly indicate what is the model, how is it trained (including the nature of the training set) and how it is used for prediction. Assume that disambiguation is for occurrence instances of a single word - for example the occurrence of `line` has different senses in the sentences below:
`The line moved slowly.`
`Two distinct lines intersect only at one point.`

Solution:
Assume S is the set of senses of word w. To do naive Bayes classification we have to calculate

$$s_w = \operatorname*{argmax}_{s \in S} P(s|w) = \operatorname*{argmax}_{s \in S} P(w|s)P(s)$$

where s_w is the sense of an occurrence instance of the word w.

To compute model parameters $P(w|s)$ and $P(s)$ we need a sense tagged text corpus where the words whose sense must be identified are sense tagged.

The prior $P(s)$ is simply the ratio ${}^{count(s)}/_{count(w)}$ in the corpus for each sense s of word w. To calculate $P(w|s)$ divide the corpus into sub-corpora based on sense, choose a window size, around each sense occurrence extract a window and train a unigram language model for each sub-corpus. One must use smoothing to avoid 0 probabilities.

To find the sense of w in a test sentence. Extract a window around w use the model parameters of each sub-corpus to compute $P(w|s)$ for words in the window (using independence assumption of naive Bayes it is just a product) and then multiply by the prior $P(s)$ to get the probability for sense s. Do this for each sense of the word and choose the sense with the highest probability.

(c) What are two key differences between how GLoVE vectors and CBoW/Skipgram (word2vec) vectors are trained? Give one major advantage of GLoVE when compared to word2vec based on the training difference.

Solution:

CBoW/Skipgram (word2vec) vectors try and predict either the probabilities of neighbours given the middle word or the converse - so it uses co-occurrence probabilities directly. GLoVE on the other hand works with co-occurrence probability ratios. Their loss functions are also different. Word2vec uses cross entropy loss while GLoVE uses a squared loss based on a direct inner product of the word vectors and the log of the co-occurrence ratio - $J = \sum_{i,j=1}^{\mathcal{V}} f(X_{ij})(w_i^T \tilde{w_j} + b_i + \tilde{b_j} - log(X_{ij}))$. X_{ij} is the ratio of co-occurrence probabilites. f is a weighting function. GLoVE vectors can be calculated by using the term co-occurrence matrix and using matrix factorization methods. GLoVE makes explicit what word2vec is actually doing.

One major advantage of GLoVE: Since the co-occurrence matrix is pre-computed GLoVE can quickly give different sized vectors via matrix factorization for which many efficient implementations are available. word2vec on the other hand must train from scratch for each vector size.

(d) Using the update equations highlight the key differences between the LSTM and GRU recurrent network architectures?

Solution:

The update equations for the two architectures are shown below:

LSTM		GRU	
i	$=\sigma(U_i x_t + W_i s_{t-1})$ input gate	z	$= \sigma(U_z x_t + W_z s_{t-1})$ update gate
f	$=\sigma(U_f x_t + W_f s_{t-1})$ forget gate	r	$= \sigma(U_r x_t + W_r s_{t-1})$ reset gate
o	$=\sigma(U_o x_t + W_o s_{t-1})$ output gate	h	$= tanh(U_h x_t + W_h(s_{t-1} \circ r))$ hidden state
g	$=tanh(U_g x_t + W_g s_{t-1})$	s_t	$= (1-z) \circ h + z \circ s_{t-1}$
c_t	$= c_{t-1} \circ f + g \circ i$		
s_t	$= tanh(c_t) \circ o$		

- GRU has 2 gates, LSTM has 3.
- GRU does not have an internal memory like c_t (in an LSTM) that is different from the internal state s_t.
- GRU does not have an output gate.

- The i and f gates are merged into the z gate.
- Overall GRU is a simpler architecture and faster computationally. LSTM allows more control, if needed.

[(2,3),8,6,6=25]

2. (a) One common way probabilities are assigned to rules in a PCFG is to use a treebank. When will some rules have a probability of 0.0? What is the problem if some rule has a probability of 0.0? How can it be overcome?

Solution:
When the tree bank has no instance of a rule among its derivation trees the rule will have a probability of 0.0.
The problem is that the probability for any parse tree using a rule with probability 0.0 will be 0.0 since the probability of the parse tree is obtained as a product of all rules in the parse.
The same smoothing methods used to handle zero counts in language models can be used here. For example, Laplace smoothing which starts with a count of 1 for all rules.

(b) Consider the treebank of 3 trees given in Figure 1. Derive a PCFG from this treebank.

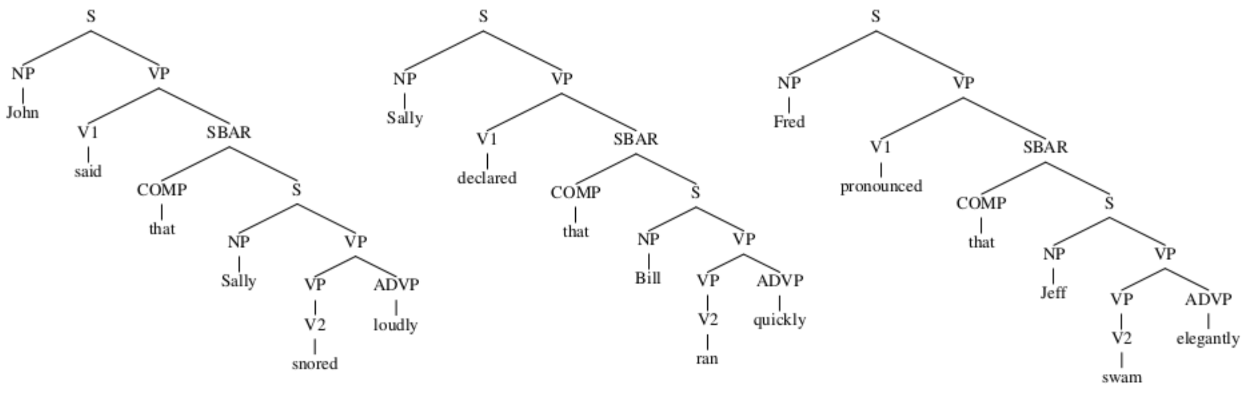


Figure 1: Treebank of 3 trees.

Solution:
The grammar and corresponding probabilities are:

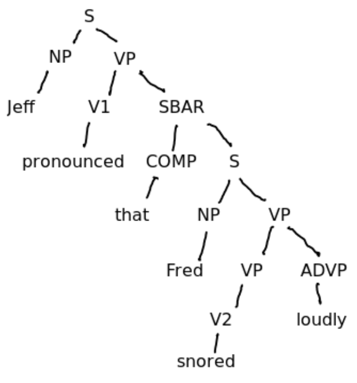
Rule	Probability
S→NP VP	1
SBAR→COMP S	1
COMP→*that*	1
VP→V1 SBAR | VP ADVP | V2	1/3
NP→*Sally*	1/3
NP→*John* | *Bill* | *Fred* | *Jeff*	1/6
V1→*said* | *declared* |*pronounced*	1/3
V2→*snored* | *ran* |*swam*	1/3
ADVP→*loudly* | *quickly* |*elegantly*	1/3

(c) Show two parse trees for the sentence:

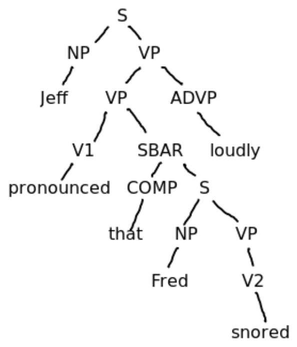
`Jeff pronounced that Fred snored loudly.`

using the PCFG constructed in (b) and calculate their probabilities. Do you think the higher probability tree is the right one? Justify.

Solution:



1



2

$P(1) = (\frac{1}{3})^6 \times (\frac{1}{6})^2$
$P(2) = (\frac{1}{3})^6 \times (\frac{1}{6})^2$

The probabilities of both parses are the same. However, we know that parse 1 is much more likely than parse 2. So parse 2 probability should be lower (or parse 1 probability higher). Actually, since there is no instance of parse 2 in the treebank it is better to restructure the grammar so as to avoid parse 2.

(d) The alternative parse for the sentence above where *loudly* qualifies *pronounced* is called high attachment. Do you think the probability of the high attachment parse is reasonable? If yes, justify - given that the treebank has no such examples. If no, transform the PCFG by altering some non-terminal labels and increasing the number of non-terminals so as to capture the distinction between high and low attachment.

Solution:

The probability of the high attachment parse 2 should be much lower than the probability of parse 1 - actually since there is no instance in the corpus it should not have such a parse. However, note that a transformation of the grammar to suppress parse 2 should not end up generating sentences that are not generated by the earlier grammar.

There are multiple ways to achieve this. A bit of inspection shows that it is possible to avoid parse 2 by noticing that the subordinate sentence (or clause) introduced by the rule SBAR$\rightarrow$ COMP S should be treated differently. This can be done by introducing a new non-terminal SS, removing the rules

VP$\rightarrow$ VP ADVP
SBAR$\rightarrow$COMP S

and introducing the following rules:

VP$\rightarrow$ V2 ADVP allows sentences like *John snored loudly.*
SBAR$\rightarrow$ COMP SS
SS$\rightarrow$ NP VPS
VPS$\rightarrow$ V1 SBAR

VPS→ V2 ADVP

This allows sentences like: *Jeff pronounced that Sally said that Fred snored loudly.* Actually, nesting can happen to any level but high attachment is not possible at any level. So, parse 2 or similar parses are not possible. This can also be achieved by restructuring just the VP rules.

The above example also shows the difficulties with high attachment. *Loudly* can qualify *snored* or *said* or *pronounced*. If we have low(or more appropriately nearest) attachment then we should allow sentences like:
Jeff pronounced loudly that Sally said that Fred snored.
Jeff pronounced that Sally loudly said that Fred snored.

But, the original grammar does not generate such sentences.

A solution that just manipulates probabilities should not be tried in general since it affects the parse tree probabilities of all parses that use those rules. Probabilities should normally be left to treebank statistics. Similarly, a lexicalization solution is also not appropriate. The problem is a structural one in this case and a structural solution is preferable. If the grammar has to be refactored later to accommodate more language data lexicalized rules can pose a big hurdle.

[5,6,8,6=25]

3. Consider the grammar G given below:

1	S	→	NP VP
2	VP	→	VT NP
3	NP	→	D N
4	N	→	ADJ N
5	VT	→	*saw*
6	D	→	*the*
7	D	→	*a*
8	N	→	*dragon*
9	N	→	*boy*
10	ADJ	→	*young*

(a) You are given the sentence below with the positions marked:

$_0$ `the` $_1$ `young` $_2$ `boy` $_3$ `saw` $_4$ `the` $_5$ `dragon` $_6$

Using the CYK parsing algorithm fill in the table/chart that indicates whether the above sentence has been parsed or not.

Solution:
If sentence s is $s = w_1, \ldots, w_n$ then any non-terminal in the table derives the partial sentence $w_{i+1}, \ldots, w_j$ where $\{A\}_{i,j}$ is an entry in the table and A is a non-terminal.
If we have $\{S\}_{0,6}$ in the table then we know that the grammar parses the sentence. There are multiple ways of organizing the table - for example it can be lower triangular instead of upper triangular.

0	1	2	3	4	5	6
	the	young	boy	saw	the	dragon
	$\{D\}_{0,1}$	$\{ADJ\}_{1,2}$	$\{N\}_{2,3}$	$\{VT\}_{3,4}$	$\{D\}_{4,5}$	$\{N\}_{5,6}$
	$\{\ \}_{0,2}$	$\{N\}_{1,3}$	$\{\ \}_{2,4}$	$\{\ \}_{3,5}$	$\{NP\}_{4,6}$	
	$\{NP\}_{0,3}$	$\{\ \}_{1,4}$	$\{\ \}_{2,5}$	$\{VP\}_{3,6}$		
	$\{\ \}_{0,4}$	$\{\ \}_{1,5}$	$\{\ \}_{2,6}$			
	$\{\ \}_{0,5}$	$\{\ \}_{1,6}$				
	$\{S\}_{0,6}$					

(b) Redraw the table with the necessary annotations (back pointers) that allow the extraction of the parse.

Solution:
The non-terminal is annotated with the rule number as a subscript.

0	1	2	3	4	5	6
	the	young	boy	saw	the	dragon
	$\{D_6\}_{0,1}$	$\{ADJ_{10}\}_{1,2}$	$\{N_9\}_{2,3}$	$\{VT_5\}_{3,4}$	$\{D_6\}_{4,5}$	$\{N_8\}_{5,6}$
	$\{\ \}_{0,2}$	$\{N_4\}_{1,3}$	$\{\ \}_{2,4}$	$\{\ \}_{3,5}$	$\{NP_3\}_{4,6}$	
	$\{NP_3\}_{0,3}$	$\{\ \}_{1,4}$	$\{\ \}_{2,5}$	$\{VP_2\}_{3,6}$		
	$\{\ \}_{0,4}$	$\{\ \}_{1,5}$	$\{\ \}_{2,6}$			
	$\{\ \}_{0,5}$	$\{\ \}_{1,6}$				
	$\{S_1\}_{0,6}$					

(c) Using the table above extract the parse in the form of a derivation of the sentence starting from the start symbol.

Solution:
The following derivation can be extracted from the table starting with $\{S_1\}_{0,6}$ where in each derivation step the left most non-terminal is expanded using the rule annotation in the table above.

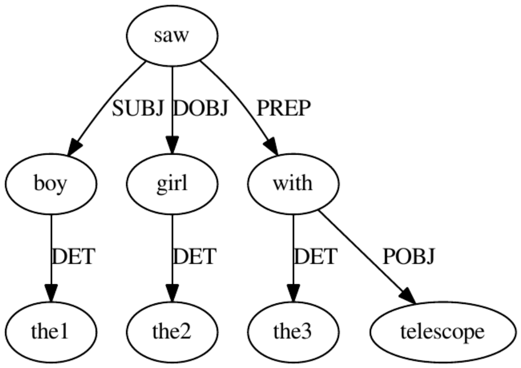
S→NP VP→D N VP→*the* N VP→*the* ADJ N VP→*the young* N VP→*the young boy* VP→*the young boy* VT NP→the young boy saw NP→*the young boy saw* D N→*the young boy saw the* N→*the young boy saw the dragon*

[8,4,3=15]

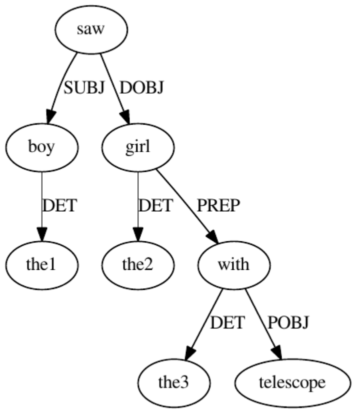
4. (a) You are given the following dependency relation tags:
`SUBJ-Subject`, `DOBJ-Direct object`, `PREP-Preposition`, `POBJ-Prepositional object`, `DET-Determiner`.

Draw all possible dependency parses (dependency trees with edges labelled with the dependency relation tags) for the sentence below.
`The boy saw the girl with the telescope.`

Solution:



The boy uses the telescope to see the girl.



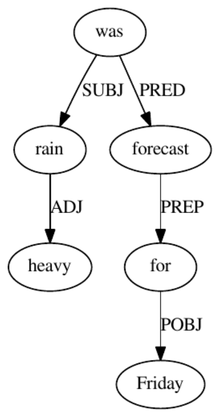
The boy saw the girl who had the telescope.

(b) For this part and the next (part (c)) the examples should only use words from the set below and use the tags in part (a) in addition if needed use `ADJ`, `PRED`:
{`the, was, rain, heavy, forecast, for, Friday, by, weatherman`}

Define a projective dependency tree and give an example sentence that has such a dependency tree. Draw the tree.

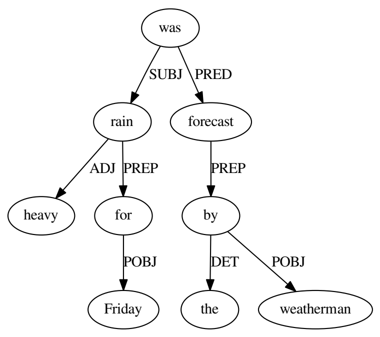
Solution:
A dependency tree is projective if there is a directed path from the head node to all nodes in the sentence between the head node and end point of the head node edge.

Example: `Heavy rain was forecast by the weatherman.`



(c) Give an example sentence that has a non-projective dependency tree and draw this tree.

Solution:
Example: `Heavy rain was forecast for Friday by the weatherman.`



[10,5,5=20]

5. (a) Give the AMR representation (Penman notation) for the sentence:
`Hari was charged with public incitement and resisting arrest.`

Solution:

```
(c / charge-01
   :ARG1 (h / Hari)
   :ARG2 (a / and
            :op1 (i / incite-01
                    :ARG0 h
                    :location (p / public)
                 )
            :op2 (r / resist-01
                    :ARG0 h
                    :ARG1 (a2 / arrest-01
                              :ARG1 h)
                 )
```

```
        )
)
```

(b) Give 3 possible text fragments for the AMR fragment below:

```
(t / thing
   :ARG1-of (o / opine-01
                :ARG0 (b / boy)
             )
)
```

Solution:

1. …the boy's opinion
2. …the opinion of the boy
3. …what the boy opined

Others are also possible.

(c) Given below are two AMR fragments that express the same meaning. What is the corresponding text fragment? Which AMR representation is preferred and why?

```
(o / organization                             (m / maker
   :ARG0-of (m / make-01                          :mod (c /chip)
                :ARG1 (c / chip)              )
             )
)
```

(d) What resource(s) do you need to do word sense disambiguation using Lesk's algorithm and how are those resource(s) used?

Solution:
Lesk's algorithm requires a dictionary that has example sentences for each different sense of a word - for example a resource like Wordnet. The dictionary (or any other resource) should also indicate the most frequent sense of a word.
By default Lesk's algorithm assigns the most frequent sense to a word. This is refined based on the degree of overlap between context words surrounding the target word in the test sentence and the context words for the same word in the example sentences in the dictionary. The sense with the maximum overlap is returned as the right sense.

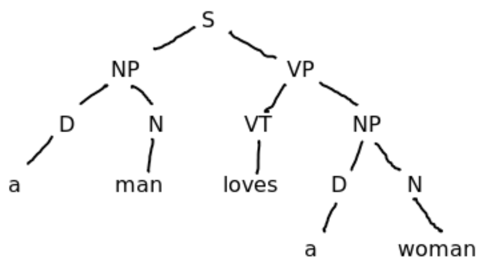
[8,3,(2,2),5=20]

6. You are given the grammar G below (where non-terminals start with capital letters and terminals with small ones) and the sentence S: `A man loves a woman.`

1.	S	→	NP VP
2.	NP	→	D N
3.	D	→	*a*
4.	VP	→	VT NP
5.	VT	→	*loves*
6.	N	→	*man*|*woman*

(a) Draw the parse tree for S using G.

Solution:



(b) Annotate grammar G with FoL+λ-calculus (First order logic+Lambda calculus) semantic annotations. Where needed, for terminal strings assume the semantic annotation is an equivalent primed string. So, if abc is a terminal string then abc' will be its semantic annotation.

Solution:

1.	S	$\rightarrow$	NP VP	[NP][VP]
2.	NP	$\rightarrow$	D N	[D][N]
3.	D	$\rightarrow$	*a*	$\lambda P(\lambda Q(\exists z(P(z) \wedge Q(z))))$
4.	VP	$\rightarrow$	VT NP	[VT][NP]
5.	VT	$\rightarrow$	*loves*	$\lambda R(\lambda U(R(\lambda V\, loves'(U,V))))$
6.	N	$\rightarrow$	*man*|*woman*	$man' \mid woman'$

(c) What is the FoL expression for the semantics of S?

Solution:
$\exists x(man'(x) \wedge \exists y(woman'(y) \wedge loves'(x,y)))$

(d) Using the syntactic tree of S (in (a) above) as a reference write the sequence of β-reductions that will compose the FoL semantic expression for the meaning of S.

Solution:
Note that a λ application can be written in multiple ways. If e_1 and e_2 are λ-expressions then an application can be written as:
$e_1\ e_2$ or $(e_1\ e_2)$ or $[e_1\ e_2]$ or $e_1(e_2)$
Using different forms helps with clarity.

a woman	$\equiv$	$[\lambda P(\lambda Q(\exists z(P(z) \wedge Q(z))))]\ woman'$
	$\xrightarrow{\beta}$	$\lambda Q(\exists z(woman'(z) \wedge Q(z)))$
Similarly *a man*	$\xrightarrow{\beta}$	$\lambda Q(\exists w(man'(w) \wedge Q(w)))$
[loves] [a woman]	$\equiv$	$[\lambda R(\lambda U(R(\lambda V(loves'(U,V)))))][\lambda Q(\exists z(woman'(z) \wedge Q(z)))]$
	$\xrightarrow{\beta}$	$\lambda U((\lambda Q(\exists z(woman'(z) \wedge Q(z))))(\lambda V(loves'(U,V))))$
	$\xrightarrow{\beta}$	$\lambda U(\exists z(woman'(z) \wedge (\lambda V(loves(U,V)))(z)))$
	$\xrightarrow{\beta}$	$\lambda U(\exists z(woman'(z) \wedge loves(U,z)))$
[a man][loves a woman]	$\equiv$	$[\lambda Q(\exists w(man'(w) \wedge Q(w)))][\lambda U(\exists z(woman'(z) \wedge loves(U,z)))]$
	$\xrightarrow{\beta}$	$(\exists w(man'(w) \wedge \lambda U(\exists z(woman'(z) \wedge loves(U,z)))(w)))$
	$\xrightarrow{\beta}$	$(\exists w(man'(w) \wedge (\exists z(woman'(z) \wedge loves(w,z)))))$

So, through β-reduction we have composed the meaning of the sentence as given in (c).

[Hint: It may help to first write the FoL expression. Then use the syntax tree and the β-reduction operation to come up with the appropriate annotations.]

[2,10,3,5=20]

7. (a) Calculate number of parameters required by a seq2seq neural machine translation system with the following characteristics:

- Source vocabulary size: 1000
- Target vocabulary size: 2000
- Encoder hidden state size: 100
- Decoder hidden state size: 200
- Attention mechanism is implemented by a one layer feedforward neural network (as discussed in class).
- RNN is implemented using an LSTM.
- There is no embeding layer.
- Assume no bias.

Divide your answer into the following components:
Encoder (All four gates are connected to input and previous state):

- Cell state (W_{hc})
- input gate (W_{hi})
- Forget gate (W_{hf})
- Output gate (W_{ho})

Attention:

- Encoder hidden state to attention (W_{ha})
- Decoder hidden state to attention (W_{za})

Decoder (All four gates are connected to encoder hidden state, previous state and previous output):

- Decoder hidden state to output (W_{zo})
- Cell state (W_{zc})
- input gate (W_{zi})
- Forget gate (W_{zf})
- Output gate (W_{zog})

You need to provide dimensionality of all matrices or vectors. Use same notation for names of matrices and vectors as given above.

Solution:

- Encoder: $W_{hc} = W_{hi} = W_{hf} = W_{ho} = 100 \times 1100 = 100 \times (1000 + 100) = 100 \times 1000 + 100 \times 100 = 110000$, $Total = 4 \times 110000 = 440000$
- Attention: $W_{ha} = 100$, $W_{za} = 200$, $Total = 100 + 200 = 300$
- Decoder: $W_{zo} = 2000 \times 200$
  $W_{zc} = W_{zi} = W_{zf} = W_{zog} = 200 \times 2300 = 200 \times (2000 + 200 + 100) = 200 \times 2000 + 200 \times 200 + 100 \times 200 = 460000$, $Total = 4 \times 460000 = 1840000$

(b) When do you think a seq2seq neural machine translation model will give inadequate or poor translations? Try to answer by using examples. How do you think the problems can be addressed. Keep your answers concise and to the point.

Solution:

Seq2Seq models work well only when it has seen enough examples of words and contexts in the training corpus to be able to generalize reasonably. Such models don't do well when translation involves significant use of context and background knowledge or constructions that may never or rarely occur in the training corpus like similies, metaphors, proverbs etc. which often cannot be translated literally. Named entities and garden path sentences can also create problems.

Some examples below using Google translate (the second language is Hindi):
a stitch in time saves nine : samay mein ek silaee nau bachaata hai
gulabi thand : Pink cold
pathar se Hari ne Shyam ko mara : Green with stone hit black
The old man the boat : boodha aadamee naav

[11,4=15]