CS 784: Final Exam Practice

Winter 2025
Exam Date: April 11, 2025
4:00pm-6:30pm, MC 2034

Instructions

- You are allowed to use any resources you like, including the internet, textbooks, course notes, calculators, and laptops.
- However, you are not allowed to communicate with anyone else about the exam.
- There are 5 questions on this exam.
- There are 8 pages in this exam (including cover page front and back, and three additional pages at the end for work that does not fit in the spaces provided).
- There are a total of 100 marks on the exam.
- You have 150 minutes (2.5 hours) to complete the exam.

Question	**1**	**2**	**3**	**4**	**5**	**TOTAL**
Marks	**5**	**30**	**20**	**20**	**25**	**100**
Score						

1 Honour Code (5 Points)

☐ By checking this box, I confirm that I will not communicate with any other persons about the exam during the exam time.

(There is actually no reliable way to detect and verify this, but you will feel guilty if you check the box and do not follow it.)

2 Muiti-Choice and Short Answers (35 Points)

Answer the questions by writing the answer in the provided blank. **There may be multiple correct answers for each question.** If so, please write all correct answers in the blank. You will receive full credit (5 pts) if you write down all correct answers and no incorrect answers, partial credit (3 pts) if you write down some correct answers and no incorrect answers, partial credit (1 pt) if you write nothing, and no credit (0 pts) if you write down any incorrect answers.

(Questions 2.1–2.2) When classifying a sentence with a ~~Deep~~ Shallow Averaging Network, we take the word embeddings as the input, use averaged word embeddings as the features, and predict the score for each class with a 2-layer perceptron (i.e., with one hidden layer). We are now interested in analyzing the complexity of this model for a single forward pass for one sentence. The size of the hidden layer is D, the number of words is S, the size of the embedding space is E, and the number of classes is K.

__C__ 2.1 (5 points) What is the time complexity of a single forward pass through it for one sentence, assuming there is no parallelization of computing?

A. $\mathcal{O}(S + ED + K)$
B. $\mathcal{O}(SEDK)$
C. $\mathcal{O}(SE + ED + DK)$
D. $\mathcal{O}(ED + DK)$
E. None of the above

__D__ 2.2. (5 points) What is the space complexity to save the **parameters of the 2-layer perceptron**?

A. $\mathcal{O}(S + ED + K)$
B. $\mathcal{O}(SEDK)$
C. $\mathcal{O}(SE + ED + DK)$
D. $\mathcal{O}(ED + DK)$
E. None of the above

(Questions 2.3–2.4) You built a sentiment analysis system that feeds word tokens into a uni-directional left-to-right recurrent neural network (RNN), then outputs the sentiment class by

putting the final hidden state through a linear + softmax layer. You observe that your model incorrectly predicts very positive sentiment for the following (negative sentiment) passage: *The play was terrible. The performances were lackluster, and the acting was unconvincing. Then, there was a long line to exit the theatre building. At least the dinner was excellent.*

D 2.3. (5 points) Why do you think the model make this decision?

A. RNNs are not good models for sequence classification tasks
B. RNNs do not model the unknown words like *lackluster* well
C. RNN training is very stable
D. An RNN's hidden state is heavily influenced by recent tokens
E. None of the above

ABCD, or BCD 2.4. (5 points) What other methods **might** work better than unidirectional RNNs for **this example** by addressing the identified issue?

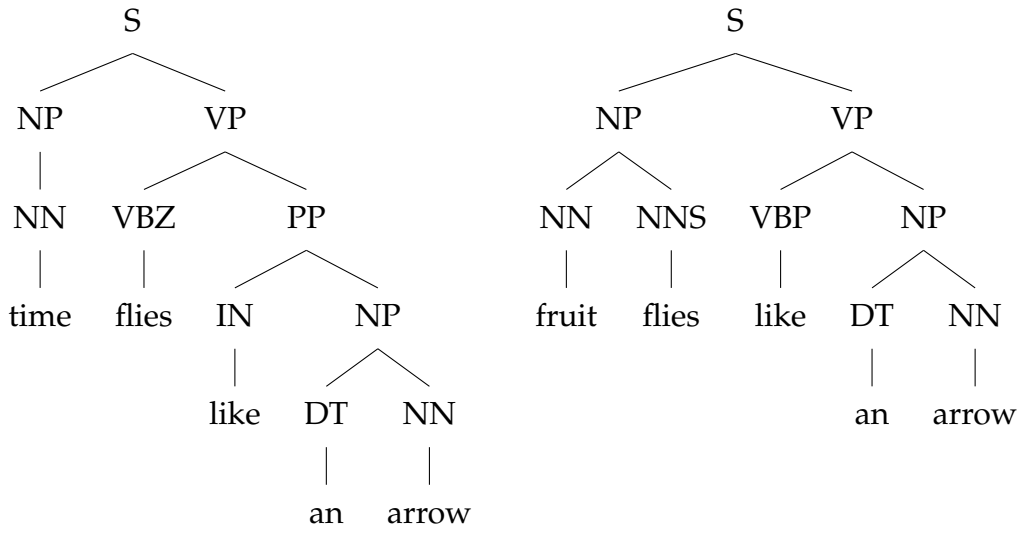
A. A bag-of-words model (i.e., average word embeddings as the sentence feature)
B. Weighted average with trainable attention mechanism as the sentence feature
C. A Transformer, using the first token (`[CLS]`) as the sentence feature
D. Bidirectional RNNs, using the concatenation of the first and last hidden states as the sentence feature
E. None of the above

ABD 2.5. (5 points) Suppose you train a model for a sentence-level classification task (such as sentiment analysis) over a dataset containing only singular nouns. You then apply it to a test set also containing plural nouns. Which of the following could help your model generalize better?

A. Stemming (i.e., change the word to its canonical form, for both training and testing data)
B. Use subword tokenization
C. Using a word-based LSTM
D. Using pre-trained word embeddings
E. Using a bag-of-words featurization

3 Constituency Parsing (20 Points)

Consider the corpus that contains two sentences and their constituency parse trees (shown below).



3.1 (6pts) Write down the probabilistic context-free grammar (PCFG) that maximizes the probability of the corpus. A PCFG defines a rewrite rule $A \rightarrow BC$ with a probability $P(A \rightarrow BC)$ or $P(A \rightarrow B)$, where A is a non-terminal symbol and B and C are either non-terminal or terminal symbols. The probabilities of rules that share the same left-hand side symbol A should sum to 1. Please include the lexical rules (i.e., those with the right-hand side containing only terminal symbols).

One example rule in the answer (you don't need to write it down again) is:
`(1.0) S -> NP VP`, where `(1.0)` denotes the corresponding probability.

To maximize the probability of the corpus, the principled way is to write down the log probability of the corpus in terms of the parameters being the optimization objective, and set the first-order gradient of this huge probability formula w.r.t. each parameter to zero. Additional constraints are the probability of rules that share the same left-hand side sum up to 1. Eventually, the solution to this constrained optimization problem is just counting based estimation.

(1.0) S → NP VP
(1/4) NP → NN
(1/4) NP → NN NNS
(1/2) NP → DT NN
(1/2) VP → VBZ PP
(1/2) VP → VBP NP
(1.0) PP → IN NP
(1/4) NN → time
(1/4) NN → fruit
(1/2) NN → arrow
(1.0) NNS → flies
(1.0) DT → an
(1.0) VBZ → flies
(1.0) VBP → like
(1.0) IN → like

3.2 (4pts) According to the PCFG you wrote in 3.1, what are the probabilities of the sentences shown in the trees?

Multiplying the probabilities of the rules in the parse tree, we have:

$$\begin{aligned} P(\mathcal{T}_1) =& P(\text{S} \to \text{NP VP}) \cdot P(\text{NP} \to \text{NN}) \cdot P(\text{NN} \to \text{time}) \cdot P(\text{VP} \to \text{VBZ PP}) \\ & \cdot P(\text{VBZ} \to \text{flies}) \cdot P(\text{PP} \to \text{IN NP}) \cdot P(\text{IN} \to \text{like}) \\ & \cdot P(\text{NP} \to \text{DT NN}) \cdot P(\text{DT} \to an) \cdot P(\text{NN} \to arrow) \\ =& 1 \cdot \frac{1}{4} \cdot \frac{1}{4} \cdot \frac{1}{2} \cdot \cdot 1 \cdot 1 \cdot 1 \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{128} \end{aligned}$$

$$\begin{aligned} P(\mathcal{T}_2) =& P(\text{S} \to \text{NP VP}) \cdot P(\text{NP} \to \text{NN NNS}) \cdot P(\text{NN} \to \text{fruit}) \cdot P(\text{NNS} \to \text{flies}) \\ & \cdot P(\text{VP} \to \text{VBP NP}) \cdot P(\text{VBP} \to like) \cdot P(\text{NP} \to DTNN) \cdot P(\text{DT} \to an) \\ & \cdot P(\text{NN} \to arrow) \\ =& 1 \cdot \frac{1}{4} \cdot \frac{1}{4} \cdot 1 \cdot \frac{1}{2} \cdot 1 \cdot \frac{1}{2} \cdot 1 \cdot \frac{1}{2} = \frac{1}{128} \end{aligned}$$

3.3 (3pts) What is (are) the sentence(s) with the highest probability according to the PCFG you wrote down in 3.1? What is the probability of this sentence/these sentences?

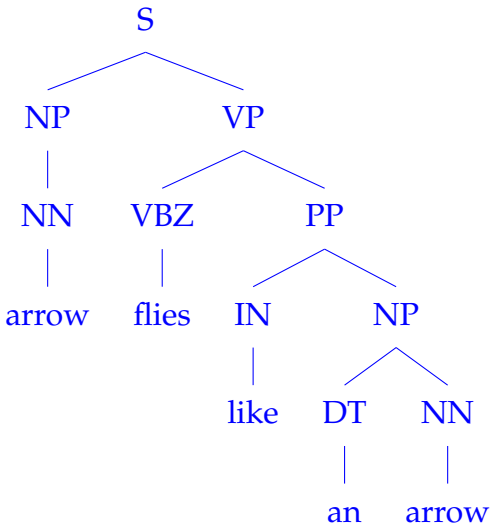
Let $f[x]$ denote the highest-probability string(s) derived from non-terminal or pre-terminal x, and $p[x]$ denote the corresponding probability. We can use a bottom-up process to obtain all $f[x]$ for all x.

$f[\text{IN}] = \text{like}$	$p[\text{IN}] = 1$	
$f[\text{VBP}] = \text{like}$	$p[\text{VBP}] = 1$	
$f[\text{VBZ}] = \text{flies}$	$p[\text{VBZ}] = 1$	
$f[\text{DT}] = \text{an}$	$p[\text{DT}] = 1$	
$f[\text{NNS}] = \text{flies}$	$p[\text{NNS}] = 1$	
$f[\text{NN}] = \text{arrow}$	$p[\text{NN}] = \frac{1}{2}$	compare across all NN rules
$f[\text{NP}] = \text{an arrow}$	$p[\text{NP}] = \frac{1}{4}$	$= p(\text{NP} \to \text{DT NN}) \times p[\text{NN}]$
		compare across all NP rules
$f[\text{PP}] = \text{like an arrow}$	$p[\text{PP}] = \frac{1}{4}$	$= p(\text{PP} \to \text{IN NP}) \times p[\text{IN}] \times p[\text{NP}]$
$f[\text{VP}] = \text{flies like arrow}$	$p[\text{VP}] = \frac{1}{8}$	$= p(\text{VP} \to \text{VBZ PP}) \times p[\text{VBZ}] \times p[\text{PP}]$
or like an arrow	$p[\text{VP}] = \frac{1}{8}$	$= p(\text{VP} \to \text{VBP NP}) \times p[\text{VBP}] \times p[\text{NP}]$
$f[\text{S}] = \text{an arrow flies like an arrow}$	$p[\text{S}] = \frac{1}{32}$	$= p(\text{S} \to \text{NP VP}) \times p[\text{NP}] \times p[\text{VP}]$
or an arrow like an arrow	$p[\text{S}] = \frac{1}{32}$	$= p(\text{S} \to \text{NP VP}) \times p[\text{NP}] \times p[\text{VP}]$

3.3 (7pts) What is (are) the most likely parse tree(s) of the "sentence" *arrow flies like an arrow*? What is the probability?

We consider all possible parse trees that can be derived using the PCFG for the sentence *arrow flies like an arrow*.

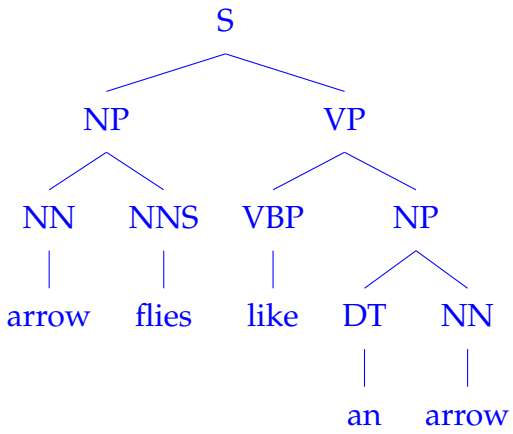
Candidate Parse 1:



Total Probability:

$$\begin{aligned} P_1 &= P(\text{S} \to \text{NP VP}) \cdot P(\text{NP} \to \text{NN}) \cdot P(\text{NN} \to \text{arrow}) \\ &\quad \cdot P(\text{VP} \to \text{VBZ PP}) \cdot P(\text{VBZ} \to \text{flies}) \cdot P(\text{PP} \to \text{IN NP}) \\ &\quad \cdot P(\text{IN} \to \text{like}) \cdot P(\text{NP} \to \text{DT NN}) \cdot P(\text{DT} \to \text{an}) \cdot P(\text{NN} \to \text{arrow}) \\ &= 1 \cdot \frac{1}{4} \cdot \frac{1}{2} \cdot \frac{1}{2} \cdot 1 \cdot 1 \cdot 1 \cdot \frac{1}{2} \cdot 1 \cdot \frac{1}{2} = \frac{1}{64} \end{aligned}$$

Candidate Parse 2 (Tree B):



Total Probability (Tree B):

$$\begin{aligned} P_2 &= P(\text{S} \to \text{NP VP}) \cdot P(\text{NP} \to \text{NN NNS}) \cdot P(\text{NN} \to \text{arrow}) \cdot P(\text{NNS} \to \text{flies}) \\ &\quad \cdot P(\text{VP} \to \text{VBP NP}) \cdot P(\text{VBP} \to \text{like}) \cdot P(\text{NP} \to \text{DT NN}) \cdot P(\text{DT} \to \text{an}) \cdot P(\text{NN} \to \text{arrow}) \\ &= 1 \cdot \frac{1}{4} \cdot \frac{1}{2} \cdot 1 \cdot \frac{1}{2} \cdot 1 \cdot \frac{1}{2} \cdot 1 \cdot \frac{1}{2} = \frac{1}{64} \end{aligned}$$

You are not required to write the CKY algorithm out for this question, however, this problem is solved by the following CKY algorithm. Let $f[\text{NT}, \ell, r]$ denote the probability of the best (i.e., highest-probability) parse tree of starting with NT and deriving the substring of the desired sentence from ℓ to r. To calculate each value of $f[\text{NT}, \ell, r]$, we enumerate the rules with NT as the left-hand side, and for each rule with two right-hand-side nodes, we enumerate all possible splits of the substring into two parts.

```
Initialize all f[NT, l, r] to 0
for length in 1..5:
  for l in 1..5-length + 1:
    r = l + length - 1:
      for nt in all_nt:
        for rule in rules[nt]:
          if rule has one RHS node:
            f[nt, l, r] = max(f[nt, l, r], p[rule] * f[rule.rhs[0], l, r])
          else:
            assert rule has two RHS nodes
            for mid in l..r-1:
              f[nt, l, r] = max(f[nt, l, r], p[rule] * f[rule.rhs[0], l, mid] *
                  f[rule.rhs[1], mid+1, r])
return f[S, 1, 5] and backtrace to get the parse tree(s)
```

4 Probing and Classifier (20 Points)

Probing is a technique used to analyze the representation of models, including but not necessarily limited to language models, through a lens of task-specific performance. Given a task T and a model M, the way to conduct probing M on T is to train a classifier C on the hidden states of M, using the training set for task T. A development set for task T is used to prevent over-fitting to the training set. We then evaluate the performance of the trained classifier C on the hold-out test set of task T, the obtained result R is referred to as the probing accuracy.

4.1 (10 points) True or False: for two models M_1 and M_2, denote R_1 and R_2 as the probing accuracy of M_1 and M_2, respectively, while keeping other factors fixed. If $R_1 > R_2$, then it is always the case that the hidden states of M_1 contain **more information** about the task T than those of M_2. Please explain your answer.

No, because the conclusion depends on the linking hypothesis that the probing accuracy is a good measure of the amount of information contained in the hidden states, which is not necessarily true.

4.2 (10 points) Suppose we now have a perfect probe classifier C that can perfectly predict the non-terminal label of a textual constituent (i.e., a span of words). Describe how you would use this probe classifier to determine which word is the head.

You may assume all inputs are valid constituents.

Suppose we can backpropagate the gradients of the probe classifier C to the hidden states of the model M. We can then compute the gradients of input tokens embeddings with respect to the output of the probe loss classifier $\nabla_{\mathbf{X}}\mathcal{L}(\mathbf{X}, \mathbf{\Theta})$, where $\mathbf{X}$ denotes the input token (embeddings) of the constituent and $\mathbf{\Theta}$ denotes the parameters of the model and the probe classifier, $\mathcal{L}$ denotes the loss function that we used to train the probe classifier.

Let $\nabla_{\mathbf{x}_i}\mathcal{L}(\mathbf{X}, \mathbf{\Theta})$ denote the gradient of the probe classifier with respect to the i-th token embedding $\mathbf{x}_i$. Since the gradient norm generally indicates "how much the loss will change if we change the input a little," we can use the gradient norm to determine which token is the head of the constituent (larger means more likely).

This process intuitively aligns the definition of the head of a constituent, i.e., the most important word in the constituent that determines the constituent category. Other proposals that use this property of constituent head will also be accepted.

5 Model Editing (25 Points)

Model editing [1, *inter alia*] is a technique to modify the behaviour of a pretrained model without retraining it from scratch. Particularly, it requires a set of training sequence pairs $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, where $\boldsymbol{x}_i$ is the input and $\boldsymbol{y}_i$ is the desired output. The goal is to modify the model's parameters so that it produces the desired output for the training examples while maintaining its performance on other examples.

Consider a simplified 1-layer Transformer model as follows, where we would like to modify the model's parameter so that it produces a single desired output token y given an input sequence $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ with sufficiently high probability. The model first computes the embedding $\boldsymbol{E}$ of the input sequence using a learned embedding matrix, then computes the intermediate representations $\boldsymbol{H}$, and finally produces the output token y using a linear layer.

$$
\begin{aligned}
\boldsymbol{E} &= \text{Embedding}(\boldsymbol{x}) && \in \mathbb{R}^{n \times d} && \text{(word embeddings)} \\
\boldsymbol{K}, \boldsymbol{Q}, \boldsymbol{V} &= \boldsymbol{W}_k \boldsymbol{E}, \boldsymbol{W}_q \boldsymbol{E}, \boldsymbol{W}_v \boldsymbol{E} && \in \mathbb{R}^{n \times d} && \text{(key, query, and value tensors)} \\
\boldsymbol{H} &= \texttt{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\intercal}{\sqrt{d}}\right)\boldsymbol{V} && \in \mathbb{R}^{n \times d} && \text{(hidden states)} \\
\boldsymbol{h} &= \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{H}_i && \in \mathbb{R}^{d} && \text{(mean-pooling hidden state)} \\
\boldsymbol{u} &= \underline{\qquad \boldsymbol{W}_o \boldsymbol{h} \qquad} && \in \underline{\mathbb{R}^{V}} && \text{(distribution logits with parameter } \boldsymbol{W}_o \in \underline{\mathbb{R}^{V \times d}}\text{)} \\
\boldsymbol{p}_i &= \underline{\qquad \frac{\exp(u_i)}{\sum_j \exp(u_j)} \qquad} && \boldsymbol{p} \in \underline{\mathbb{R}^{V}} && \text{(distribution over the vocabulary)}
\end{aligned}
$$

5.1 (10 points) Suppose the vocabulary size is V, complete the model pipeline by filling in the following equations. Please do not write softmax operator in the last equation, instead, use exp and $\sum$ operators. (Erratum: The original question missed a subscript i in the last equation, which is now corrected.)

There are other possible ways in calculating $\boldsymbol{u}$—anything that is equivalent to the solution above is acceptable.

5.2 (5 points) Suppose you would like to edit the model so that it produces the output token y given the input sequence $\boldsymbol{x}$ with sufficiently high probability. You are only allowed to modify one matrix among $\boldsymbol{W}_k$, $\boldsymbol{W}_q$, and $\boldsymbol{W}_v$. Which one would you choose and why?

By modifying $\boldsymbol{W}_v$, we can directly change the content of the hidden states, whereas there needs more indirect steps for modifying $\boldsymbol{W}_k$ and $\boldsymbol{W}_q$ to change the content of the hidden states.

In nonlinear setups, modifying $\boldsymbol{W}_v$ will also possibly change the space spanned by the hidden states, which is not the case for $\boldsymbol{W}_k$ and $\boldsymbol{W}_q$.

5.3 (5 points) You are allowed to only modify $\boldsymbol{W}_o$ by adding a small perturbation $\boldsymbol{W}_\Delta$. You are asked to have a loss function, and get $\boldsymbol{W}_\Delta$ by gradient descent—that is, $\boldsymbol{W}_\Delta = \frac{\partial L}{\partial \boldsymbol{W}_o}$. Please write down the loss function you would use.

Taking $\boldsymbol{x}$ as the input to the neural network and $\boldsymbol{y}$ as the desired output, we can use the cross-entropy loss function as follows:

$$L = -\log \boldsymbol{p}_{\boldsymbol{y}}$$

Any loss function that is suitable for the task, e.g., the L_2 loss, is acceptable.

5.4 (5 points) What is the rank of $\boldsymbol{W}_\Delta$? Please prove.

$$\text{rank}(\boldsymbol{W}_\Delta) = \text{rank}(\frac{\partial L}{\partial \boldsymbol{W}_o}) = 1.$$

$$\text{Proof:} \quad \boldsymbol{W}_\Delta = \frac{\partial L}{\partial \boldsymbol{W}_o} = \frac{\partial L}{\partial \boldsymbol{u}} \cdot \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{W}_o}.$$

The gradient of the loss L with respect to $\boldsymbol{u}$ is a vector of size V, denoted as $\frac{\partial L}{\partial \boldsymbol{u}} \in \mathbb{R}^V$. The gradient of $\boldsymbol{u}$ with respect to $\boldsymbol{W}_o$ is:

$$\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{W}_o} = \boldsymbol{h}^\intercal,$$

where $\boldsymbol{h} \in \mathbb{R}^d$ is the mean-pooled hidden state.

Thus, $\boldsymbol{W}_\Delta$ can be expressed as:

$$\boldsymbol{W}_\Delta = \frac{\partial L}{\partial \boldsymbol{u}} \cdot \boldsymbol{h}^\intercal,$$

which is the outer product of a vector of size V and a vector of size d. The result is a matrix of size $V \times d$.

Since the outer product of two vectors always has rank 1 (assuming neither vector is the zero vector), the rank of $\boldsymbol{W}_\Delta$ is:

$$\text{rank}(\boldsymbol{W}_\Delta) = 1.$$

□

References

[1] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. Advances in Neural Information Processing Systems, 35:17359–17372, 2022.

[2] Allyson Ettinger, Ahmed Elgohary, and Philip Resnik. Probing for semantic evidence of composition by means of simple classification tasks Proceedings of the 1st workshop on evaluating vector-space representations for NLP

Extra Page

Extra Page

Extra Page