Midterm for CS378: Natural Language Processing

Instructions:

- You will have 80 minutes to complete the exam.
- This exam is to be completed individually be each student.
- You are allowed one 8.5"x11" double-sided note sheet.
- You are **not** allowed calculators or other electronic devices.
- Partial credit will be given for short-answer and long-answer questions, so please show work in the exam.
- For short-answer and long-answer questions, **please box or circle your final answer** (unless it is an explanation).

Grading Sheet (for instructor use only)

Question	Points	Score
1	20	
2	25	
3	15	
4	8	
5	12	
Total:	80	

Name: _____

EID: _____

Part 1: Multiple Choice (20 points)

1. (20 points) Answer the following multiple choice questions (2 points each) by writing the answer in the provided blank.

   Suppose you have the following training data for Naïve Bayes:

```
I liked the movie [LABEL=+]
I hated the movie because it was an action movie [LABEL=-]
Really cool movie [LABEL=+]
```

_____ (1) What is the unsmoothed maximum likelihood estimate of $P(+)$ for this data?

A. 1/3 B. 1/2 C. 2/3 D. 1

C, 2/3 (two positive / one negative)

_____ (2) What is the unsmoothed maximum likelihood estimate of $P(\text{movie}|+)$ for this data?

A. 2/17 B. 1/5 C. 2/7 D. 1/2

C, 2/7 (7 words emitted in positive samples, 2 of them are movie)

_____ (3) Suppose we are given an unseen input sentence *the movie*. What is the joint probability $P(-, \text{the movie})$?

A. 2/300 B. 4/98 C. 1/12 D. 1/3

A, 2/300 (1/3 * 2/10 * 1/10)

_____ (4) What prediction will the model make on *the movie*?

A. Positive B. Negative

A, Positive, compare 4/98 to 2/300

_____ (5) Consider the following code snippet to define a neural network in PyTorch:

```
class MyNetwork(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        # Boilerplate not shown
        self.l1 = nn.Linear(input_size, hidden_size)
        self.nonlin = nn.Tanh()
        self.l2 = nn.Linear(hidden_size, output_size)
        self.log_sm = nn.LogSoftmax(dim=0)
        # Initialization, etc. not shown

    def forward(self, x):
        return self.log_sm(self.nonlin(self.l2(self.nonlin(self.l1(x)))))
```

Let b be a batch size > 1. Which of the following tensor shapes can be input to this network without causing a code crash?

I. [inp] II. [b, inp] III. [inp, b]

A. I B. I and II C. I and III D. II and III E. I, II, and III

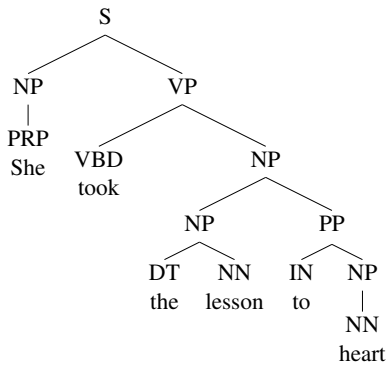
B. batch is the first dimension, not last

_____ (6) What is "wrong" with the above network compared to standard neural network models?

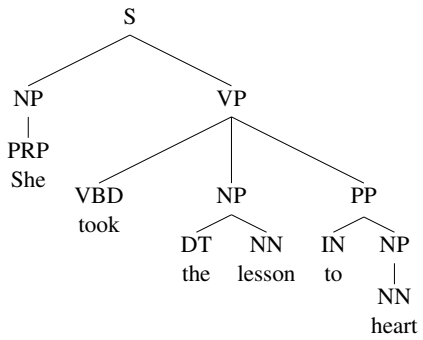
A. Tanh is not differentiable so backpropagation will not work
B. There is a dimension mismatch which will lead to a crash
C. Reusing the `self.nonlin` layer will cause a crash
D. The presence of the nonlinearity before the softmax results in a nonstandard model
D, you should never have a nonlinearity right before the softmax layer. All of the other answers are incorrect.

_____ (7) Indicate the correct tree for the most natural interpretation of the following English sentence:

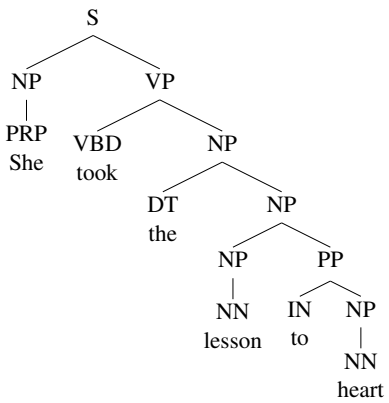
A.



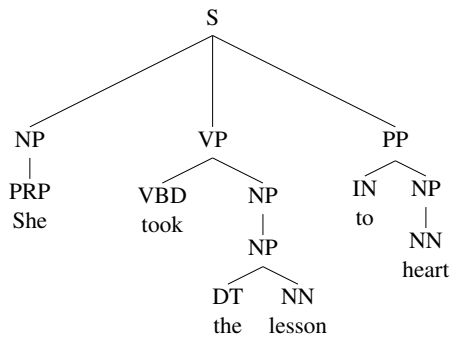
B.



C.



D.



B. *to heart* is a property of the *took* event. A is the other semi-reasonable parse, but this posits that there is a *the lesson to heart* concept, similar to the parse for the phrase *the chief of police*, but this makes no sense.

_____ (8) Suppose we binarize tree (D) from the previous question with $h = 0$ according to the technique discussed in class. What new constituent will be introduced?

A. S B. VP C. NP D. PP
A

_____ (9) Suppose you are running a shift-reduce dependency parser with the arc-standard system on a sentence of length n. How many **shift** operations are needed?

A. Cannot be determined B. $n-1$ C. n D. $n+1$ D. $2n-1$ E. $2n$
C

_____ (10) Suppose we have the following **word** vectors: *swam* = [1, 0, 0]; *ran* = [1, 1, 0]; *ate* = [0, 0, 2]. Now we get two new examples of a new word *wug*: *wug swam*, *wug ate*. We are going to learn parameters just for *wug* without changing the rest of our vector space. Which **context** embedding for *wug* gives the best value of the skip-gram objective on these two examples? Recall that skip-gram model is defined by the following probability distribution over contexts given a word: $P(c|w) = \frac{e^{c \cdot w}}{\sum_{c' \in V} e^{c' \cdot w}}$

A. [10, 0, 5] B. [10, -10, 0] C. [10, -10, 5] D. [0, -10, 5]

C; Should be as close to 0.5 / 0 / 0.5 as possible. The C vector achieves this roughly with 10 / 0 / 10 before softmaxing

Part 2: Short Answer (25 points)

2. (25 points) Answer the following short-answer questions.

a. (5 points) Suppose you are doing bag-of-words text classification on a document. The raw input is a single string containing the text of the entire document. Describe in one or two sentences the pipeline to go from the raw input to a feature vector.

You should've said something about tokenization, indexing/word counting, and building of the feature vector. -1 if you did not mention tokenization explicitly. More points taken off if you said something about counting and normalizing, which is for training a Naive Bayes model and not feature extraction.

b. (3 points) Suppose you have a neural network that is overfitting to the training data. Describe **two ways** to fix this situation.

Several possible answers: regularization, dropout, train less / early stopping, decrease model capacity

c. (3 points) You are training a neural network with Adam and watching the negative log likelihood of the training set over epochs. Rather than decreasing, it seems to fluctuate around where it started. What is one change you could make to your training procedure that could fix this?

The best answer is to lower learning rate.

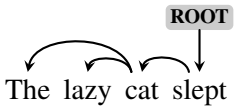
d. (3 points) Describe one way a neural network could be initialized poorly and what effect that could have.

Initializing to 0 means learning is impossible. Initializing too large can saturate units. Initializing the same across different layer sizes can lead to different scales. Very generic answers ("initializing poorly means that learning doesn't work") did not receive credit.

e. (3 points) Describe the reason why skip-gram with negative sampling is faster to train than the original skip-gram model.

Computing the normalization constant for skip-gram during learning requires normalizing over the vocabulary, whereas skip-gram with negative sampling does not. You can also see this as only reducing the scores of certain sampled contexts rather than all negative contexts.

f. (3 points) Consider the following sentence:



Give a correct sequence of shift-reduce operations that will lead to the correct parse. Recall that the start state is Stack=[ROOT] and Buffer=[the lazy cat slept].

SSSLLSLR

g. (5 points) Fill in the following table about algorithms discussed in this class. Give the best single answer for each cell by filling in the blank or circling a value (for the Model Type column). Possible values for the blank cells are given below. You can use the acronyms for SGD and MLE.

Options: SGD (stochastic gradient decent), tagging, MLE (maximum likelihood estimation), classification

Method	Task Type	Model Type	Training
Logistic regression		Generative / Discriminative	
Naive Bayes		Generative / Discriminative	
HMMs		Generative / Discriminative	
Feedforward networks		Generative / Discriminative	

classification, discriminative, MLE/SGD (both accepted)
classification, generative, MLE
tagging, generative, MLE
classification, discriminative, SGD (MLE also accepted)

Part 3: Long Answer (36 points)

3. (15 points) Below is the Viterbi algorithm. Assume that the tags are indexed into the set of integers from 1 to t.

Algorithm 1 Viterbi Algorithm

1: **function** VITERBI(x, S, T, E) ▷ x: sentence of length n, S: initial log probs, T: transition log probs, E: emission log probs
2: Initialize v, a $n \times t$ matrix
3: **for** $y = 1$ to t **do** ▷ Handle the initial state
4: $v[1, y] = S[y] + E[y, x_1]$
5: **end for**
6: **for** $i = 2$ to n **do**
7: **for** $y = 1$ to t **do**
8: $v[i, y] = E[y, x_i] + \max_{y_{\text{prev}}} \left(T[y, y_{\text{prev}}] + v[i-1, y_{\text{prev}}]\right)$
9: **end for**
10: **end for**
11: Best final state = $\arg\max_y (v[n, y] + T[\text{STOP}, y])$
12: Reconstruct answer with tracked argmaxes (not shown)
13: **end function**

a. (2 points) What is the runtime of the Viterbi algorithm?

O(nt^2)

b. (4 points) Suppose certain state transitions are completely illegal; that is, $P(y|y_{\text{prev}}) = 0$ for those state pairs. Modify the above algorithm to take advantage of this information. You should write your answer as a description of which line numbers to modify and either a rewriting of the appropriate lines or a clear, precise textual description of what needs to change.

Two possibly useful notions we will define for you: $N(y) =$ the set of tags that can legally precede y. $B(y) =$ set of tags that can legally follow y.

The main change is that line 8 should be modified to max over $y_{\text{prev}} \in N(y)$. Other modifications to the initialization and final state computation are good as well and would technically be required but we did not take points off if these were missing.

c. (2 points) Suppose each tag has at most $k < t$ legal transitions it can make. What is the runtime of this modified version of Viterbi?

O(nkt)

d. (5 points) Suppose we are building a trigram tagger. The normal transition probabilities $P(y|y_{i-1})$ are replaced with probabilities that depend on the previous two tags $P(y|y_{i-2}, y_{i-1})$. Emissions are unchanged. Describe a modification to the algorithm to implement this.

Two solutions, which amount to the same thing: (1) Redefine v to be a 3-dimensional array. Loop over all possible pairs of tags prior to the current one. (2) Redefine v to be 2-dimensional and each "tag" is now a pair of tags. Only certain transitions in this expanded state space are legal. The modal answer from students was to loop over $v[i-1, y_{\text{prev}}]$ and $v[i-1, y_{\text{prevprev}}]$. This does not compute the right thing and significantly double counts paths through the state space.

e. (2 points) Given the algorithm in part (d), suppose we limit the legal tag transitions as in part (c). What is the runtime of your new Viterbi algorithm? Give the best runtime you can achieve.

O(nk^2t). Most people got this right for the wrong reason. Only $O(kt)$ tag pairs are possible, and from this only $O(k)$ transitions to the next tag are valid. So you populate a $O(nkt)$ matrix with size $O(k)$ for each operation. Most people were instead filling in a $O(nt)$ matrix with time $O(k^2)$ for each operation.

4. (8 points) We are trying to classify names that we see as either the names of people or the names of places using binary classification. Let PERSON be the positive class in this scenario. Here are the examples in our dataset:

   1. Hampshire [Label=PLACE $(-)$]
   2. Waterton [Label=PLACE $(-)$]
   3. Washington [Label=PERSON $(+)$]
   4. Newton [Label=PLACE $(-)$]

   Our feature representation consists of 3-gram suffixes; for example, the first example above is represented by the single feature [ire] and the second example by the single feature [ton]. We define a weight vector $\mathbf{w}$ over this feature space. Our decision rule is $\mathbf{w}^\top \mathbf{x} \geq 0$, so we predict PERSON when the score is zero or greater.

   a. (5 points) Suppose we initialize $\mathbf{w} = \mathbf{0}$ (the zero vector) and go through one epoch of perceptron training with step size (α) of 1 on the examples in the order 1, 2, 3, 4. What is the final perceptron weight vector?

   The weight vector is [suffix=ire, suffix=ton]. The model gets every single example wrong. Updating on all of these gives [0,0] $\rightarrow$ [-1,0] $\rightarrow$ [-1,-1] $\rightarrow$ [-1, 0] $\rightarrow$ [-1,-1], for a final answer of [-1,-1]

   c. (3 points) Can perceptron achieve perfect accuracy on this training data with this feature representation? If yes, give a weight vector that does this. If no, name an additional feature that would allow us to achieve perfect classification accuracy on this dataset.

   No, more or less any additional feature works that separates *Washington* including word length, prefixes ≥ 3, suffixes ≥ 4, etc.

5. (12 points) Consider the following grammar:

VP $\rightarrow$ V N [0.25]
VP $\rightarrow$ V NP [0.25]
VP $\rightarrow$ V PP [0.25]
VP $\rightarrow$ VP PP [0.25]
NP $\rightarrow$ N PP [1.0]
PP $\rightarrow$ P N [1.0]

N $\rightarrow$ reports [0.5]
N $\rightarrow$ Mars [0.5]
P $\rightarrow$ on [1.0]
V $\rightarrow$ wrote [0.5]
V $\rightarrow$ reports [0.5]

Define a PCFG with this grammar including the nonterminals {VP, NP, PP, N, P, V}, the terminals {reports, Mars, on, wrote}, the start symbol VP, and the probabilities as given above. That is, this is a grammar to generate verb phrases. **For the purposes of this problem, you can assume** $\log 0.5 = -1$ **(i.e., use base 2 for the logarithm).**

a. (3 points) Consider the new verb phrase ***reports on Mars***. Parse this sentence with the grammar. Write **both** the highest-probability parse of the sentence **and** the joint probability of that parse and the words $P(T, x)$.

CKY chart below with scores. The best possible parse is the one rooted in VP. However, many students failed to observe that the root symbol was VP and gave the NP parse. Since the importance of starting with the root symbol wasn't really emphasized in class, we decided not to take off points for this.

```
       VP-4/NP-2
      NONE     PP-1
V-1/N-1   P0   N-1
```

b. (5 points) Consider the new sentence ***wrote reports on Mars***. Parse this sentence with the grammar. Write **both** the highest-probability parse of the sentence **and** the joint probability of that parse and the words $P(T, x)$.

CKY chart below. The best possible parse uses VP → V NP at the top level.

```
      VP-5 from right one, -7 from wrong one
   NONE    NP-2/VP-4
 VP-4    NONE   PP-1
V-1   N/V-1   P 0    N-1
```

c. (2 points) How many parses of ***wrote reports on Mars*** can we build with this grammar?

2

d. (2 points) When parsing ***wrote reports on Mars***, what is the largest constituent (in terms of number of words) that we can build that is not part of a valid complete parse?

3. VP over (reports on Mars)

[BONUS] (2 points) What is Google's dependency parser called? Either name is acceptable. Parsey McParseFace or SyntaxNet