CS 4650: Natural Language Processing
Practice Midterm

Instructor: Wei Xu
TAs: Xiaofeng Wu, Tarek Naous, Jonathan Zheng, Yao Dou

March 7, 2025

As a **close-book close-note** exam, you are not expected to be able to answer all the questions correctly. You may get challenged on some of the questions. Try your best! We may curve (only up) for final letter grades, if it is needed.

Questions are not sequenced in order of difficulty. Make sure to look ahead if you are stuck on a particular question.

Write answers in the spaces provided. Write as clearly as possible. No credit will be given to unreadable handwriting or ambiguous answers.

Partial Credit: If you show your work and describe your approach, we will happily give partial credit wherever possible. Answers without supporting work will not be given credit. If you run out of space for work, there is an extra sheet of paper attached to the end of this exam you can use.

Q1. Logistic Regression	/3
Q2. HMM and Viterbi Algorithm	/3
Q3. Perception Algorithm	/3
Q4. Evaluation Metric + Precision/Recall/F1	/2
Q5. Feedforward Neural Network	/2
Q6. Attention	/2
Q7. Short Answers and Multi-Choice Questions	/5
Total	/20

Name: _____

GTID: _____

1 Logistic Regression

Assume you are given a training set of D documents, $d_1, \ldots, d_D$, with labels $y_1, \ldots, y_D$, where $y \in 0, 1$ (there are 2 categories, for example spam or not spam).

$f(\cdot)$ is a feature function, $f(d) = \langle f_1(d), \ldots, f_k(d), \ldots, f_V(d) \rangle \in \mathcal{R}^V$, where $f_k(d)$ is the number of times word k appears in document d and there are V total words in the vocabulary.

(1) **(1 point)** You are given a fixed set of parameters $\theta \in \mathcal{R}^V$. Write down the equation used by logistic regression to estimate $P(y = 1|d)$ for some document, d, using these parameters. *Hint: recall the logistic function:* $\frac{1}{1+e^{-x}}$

(2) **(1 point)** Now write down the objective function, $O(\theta)$, that is maximized by Logistic Regression, to choose parameters θ based on the training dataset described above. Note that we are only asking you to write the function that is to be maximized and not anything about how to choose θ that maximizes it at this point.

(3) **(1 point)** Write high-level pseudocode for how to optimize the Logistic Regression objective in part (b) using gradient ascent with learning rate η. *Hint: you do not need to write down the exact form of the gradient. You can just write* $\frac{\partial O(\theta)}{\partial \theta_k}$ *or* $\nabla O(\theta)$

2 HMM and Viterbi Algorithm

The Viterbi algorithm is a dynamic programming algorithm for finding the most likely sequence of hidden state. It can be used to solve Hidden Markov Models (HMMs) as well as many other problems.

Assume there is a two-word language, which namely consists of only two words: $fish$ and *sleep*. We have a small training corpus. In this training corpus, word $fish$ appears 8 times as a noun (NN) and 5 times as a verb (VB); word *sleep* appears twice as a noun and 5 times as a verb.

(1) **(1 point)** What are the emission probabilities?

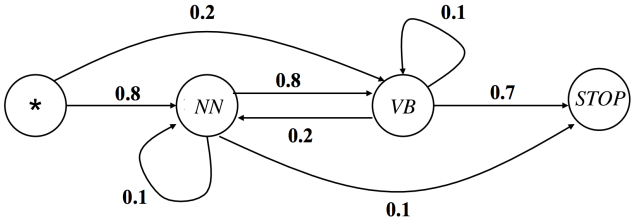
$e(fish|NN) =$

$e(sleep|NN) =$

$e(fish|VB) =$

$e(sleep|VB) =$

(2) **(2 points)** Also suppose we already have a simple Part-of-Speech HMM model, a bigram one:



The arrows in the diagram denote conditional probabilities, e.g. $q(NN|*) = 0.8$ and $q(NN|VB) = 0.2$.

Use the Viterbi algorithm to find the most likely POS tag sequence for a test sentence *"Fish sleep."* For simplicity, we ignore the punctuation in the calculation for this toy example.

The Viterbi algorithm for a bigram HMM

Input: a sentence $x_1 \dots x_n$, parameters $q(s|v)$ and $e(x|s)$
Initialization: Set $\pi(0, *) = 1$
Definition: $S_0 = *$, $S_k = S$ for $k \in \{1 \dots n\}$
Algorithm:

For $k = 1 \dots n$,
    For $v \in S_k$,

$$\pi(k, v) = \max_{w \in S_{k-1}} \big(\pi(k-1, w) \times q(v|w) \times e(x_k|v) \big)$$

Return $\max_{v \in S_n} \big(\pi(n, v) \times q(STOP|v) \big)$

3 Perception Algorithm

$\mathbf{x}$: vector of n features for a single instance; $\mathbf{w}$: vector of n weights; b: bias; y: class label for this instance;
.

Activation a is the outcome score, used in both training and testing. It is about making prediction for a single instance (online learning) with the current set of weights:

$$a = \sum_n w_n x_n = \mathbf{w}^T \mathbf{x} + b \qquad \hat{y} = SIGN(a)$$

Training:

Start with some initial weight vector $\mathbf{w}$ and bias term b (such as 0)
Loop for K iterations
    For each training instance, compute activation a
        if $ya > 0$, do nothing
        if $ya \leq 0$, update the weights:
            $\mathbf{w} = \mathbf{w} + y\mathbf{x}$
            $b = b + y$

(1) **(1 point)** What does checking $ya \leq 0$ do? Would $ya < 0$ work as well?

(2) **(1 point)** Let's try an example. Suppose we have the following data points, and no bias term (draw 2D plot):

$x_1 = (1, 2), y_1 = 1$
$x_2 = (0, -1), y_2 = -1$
$x_3 = (2, 1), y_3 = -1$

(3) **(1 point)** Show, mathematically, why the parameter updates will make it do better on the same training instance next time around?

4 Evaluation Metric – F1 Score

(2 point) The F1-score is a metric that balances Precision and Recall to measure a model's performance. It is often preferable to accuracy in classification tasks where the true classes are not evenly distributed. In the task of Named Entity Recognition (NER), F1 scores help mitigate the often unbalanced distribution of tags. Suppose we have the following sentence with three named entities (organization, person, location) with the gold labels and predictions from two NER systems.

Sentence	`Angela`	`Merkel`	`negotiates`	`European`	`Union`	`deal`	`in`	`Brussels`	`.`
Gold Labels	B-PER	I-PER	O	B-ORG	I-ORG	O	O	B-LOC	.
System #1	B-PER	O	O	O	B-ORG	O	O	B-LOC	.
System #2	B-PER	I-PER	O	B-ORG	I-ORG	B-PER	O	B-LOC	.

What is the **entity-level** Precision and Recall of each system's performance? **Don't give any credit to partially matched entities.** Show your work along with the final numerical value.

(a) Precision of System #1 = _____

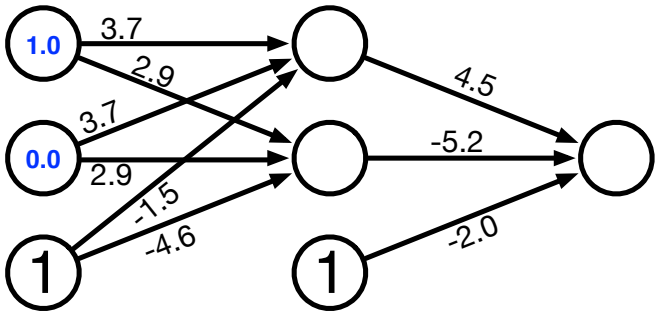
Recall of System #1 = _____

(b) Precision of System #2 = _____

Recall of System #2 = _____

5 Feedforward Neural Network

(2 point) Below is a simple neural network with one hidden layer:



$$\mathbf{W}_0 = \begin{bmatrix} 3.7 & 3.7 \\ 2.9 & 2.9 \end{bmatrix}, \mathbf{b}_0 = [-1.5, -4.6], \mathbf{W}_1 = \begin{bmatrix} 4.5 \\ -5.2 \end{bmatrix}, \mathbf{b}_1 = [-2.0]$$

Assume we use sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$ as activation function. Let's try out two input values $\mathbf{x} = [1.0, 0.0]^T$ and calculate the hidden layer $\mathbf{h}$ and outputs y of the network.

Note: In-class midterm will not involve as much real-number multiplication as in this question, but simple single-digit multiplications. Calculator is optional; in general, it would not be needed.

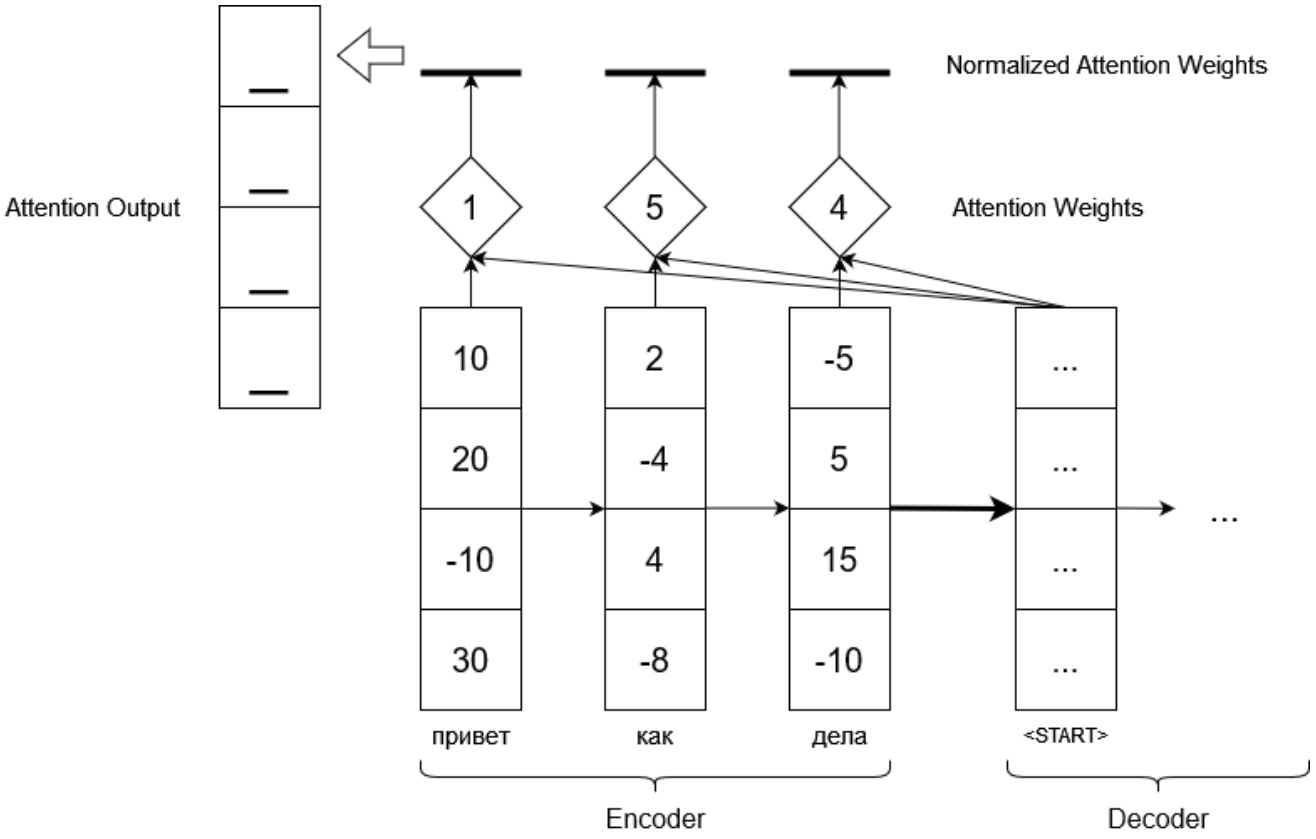
6 Attention

(2 points) Consider the basic encoder-decoder set-up below, which is attempting to machine translate "hello how are you" from Russian back to English. As you can see, we are currently in the first time step of the decoder (decoding from the start token) and have already calculated the attention score for each of the three words in Russian. All that is left is to **normalize those scores** and **calculate the attention output**. We leave that task to you! Fill in the blanks corresponding to the normalized attention weights of each Russian word, as well as the final attention output.

Note: To simplify calculations, instead of using softmax, you should use a normalized weight. The formula for calculating the normalized weights of a vector is shown below.

$$NW([w_1, \ldots, w_n]) = \left[\frac{w_1}{\sum_{i=1}^{n} w_i}, \ldots, \frac{w_n}{\sum_{i=1}^{n} w_i}\right]$$

Hint: These calculations should be quite simple and do not require the use of a calculator.



7 Short Answers and Multi-Choice Questions

(1) **(1 point)** Consider the following PyTorch code:

```
class CvNet(nn.Module):
    def __init__(self):
        super(CvNet, self).__init__()
        self.embedding = nn.Embedding(1000, 50)
        self.conv = nn.Conv1d(50, 100, 3)
        self.pool = nn.MaxPool1d(38)

    def forward(self, s):
        x = self.embedding(s)
        x = x.transpose(1, 2)
        x = self.conv(x)
        x = self.pool(x)
        return x

s = torch.randint(0, 1000, (1, 40)) #[Batch,Len]
model = CvNet()
```

What is the dimension of the output after the pooling layer?

(2) **(1 point)** Which of the following best describes the BLEU score (select the best answer below):

(a) The objective function used for training neural machine translation systems.

(b) A metric that is used to evaluate translation quality in which human judges rate translations on a scale from 1-5.

(c) A metric that is used to automatically evaluate machine translation systems.

(d) A way to measure the difficulty of translation between a given language pair (e.g., English/French).

(3) **(1 point)** What problem do LSTMs address that vanilla recurrent neural networks (e.g., Elman Networks) suffer from? (select the best answer from the choices below)

(a) Elman Networks suffer from slow training times

(b) Elman Networks suffer from the problem of vanishing gradients

(c) LSTMs have fewer parameters, so they are likely to overfit

(d) Elman Networks are more difficult to parallelize

(e) Elman Networks have lower accuracy

(4) **(1 point)** Consider the skip-gram model, which computes the probability of a context word w' given a center word w as follows:

$$P(w' \mid w) = \text{softmax}(W \cdot \text{embedding}(w)), \quad \text{where } W \in \mathbb{R}^{|V| \times d}$$

What is the big-O runtime of computing a single probability $P(w' \mid w)$ under this model? Express your answer in terms of dimensionality d and vocabulary size $|V|$.

(5) **(1 point)** Neural language models generate text by sampling from probability distributions $P(x_t|x_{<t})$. Key sampling strategies include:

- **Greedy Decoding**: Always selects the highest probability token:

$$x_t = \arg\max_x P(x|x_{<t})$$

- **Beam Search**: Maintains B most probable sequences during generation.
- **Temperature Sampling**: Controls randomness by scaling logits:

$$P_T(x_t|x_{<t}) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

  Higher T flattens the distribution, increasing diversity.

- **Top-k Sampling**: Restricts sampling to the k most likely tokens:

$$V^{(k)} = \{x_1, x_2, ..., x_k\}$$

  where tokens are sorted by decreasing probability: $P(x_1|x_{<t}) \geq P(x_2|x_{<t}) \geq ... \geq P(x_k|x_{<t})$.

- **Top-p Sampling** (Nucleus Sampling): Samples from the smallest subset of vocabulary $V^{(p)}$ whose cumulative probability exceeds threshold p:

$$V^{(p)} = \min\{V' \subset V : \sum_{x_i \in V'} P(x_i|x_{<t}) \geq p\}$$

  where tokens are sorted by decreasing probability.

When generating text with a language model, you observe that outputs are becoming repetitive and predictable. Which modification would MOST effectively increase the diversity of outputs?

(a) Switching from temperature sampling (T=0.7) to greedy decoding

(b) Keeping temperature sampling but decreasing T from 0.7 to 0.3

(c) Switching from top-p sampling (p=0.9) to top-p sampling (p=0.5)

(d) Switching from greedy decoding to temperature sampling (T=1.2)

(e) Implementing beam search with beam width = 10