This exam has seven questions. They vary in difficulty and topic, so I recommend searching around for those you're more familiar with first. I don't expect many people will achieve a perfect score, so keep that in mind if you're struggling. I recommend putting some thoughts down for each question at least, as we will be giving partial credit. Please answer each question in the space below the question.

Problem 1 (2 points) I have a vocabulary $\mathcal{V}$. What is the maximum entropy I can have of a probability distribution over $\mathcal{V}$? What is the distribution that achieves this entropy? You can provide the answers with no derivations or arguments.

Solution:

Distribution with maximum entropy is the uniform distribution, i.e. $p(w) = \frac{1}{|\mathcal{V}|}$ for all $w \in \mathcal{V}$. Plugging this into the entropy formula:

$$\begin{aligned} H(w) &= -\sum_w p(w) \log p(w) \\ &= -\sum_w \frac{1}{|\mathcal{V}|} \log \frac{1}{|\mathcal{V}|} \\ &= -|\mathcal{V}| \cdot \frac{1}{|\mathcal{V}|} \log \frac{1}{|\mathcal{V}|} \\ &= -\log \frac{1}{|\mathcal{V}|} = \log |\mathcal{V}| \end{aligned}$$

Problem 2 (2 points) I have vectors $u \in \mathbb{R}^d$ and $v \in \mathbb{R}^d$. They are orthonormal; that is, $\|u\| = \|v\| = 1$, and $u \perp v$. I have a matrix $W = (uv^\top + vu^\top)$. What is WWu? What is WWv? What are the eigenvectors of W^2?

Solution:

Note that $Wu = v$ and $Wv = u$. This means $WWu = Wv = u$ and $WWv = Wu = v$. Since $WWu = W^2u = u = 1u$ and $WWv = W^2v = v = 1v$, u and v are eigenvectors of W^2 with eigenvalue 1. In fact, any vector in the span of u and v (i.e. of the form $au + bv$ for $a, b \in \mathbb{R}$) are eigenvectors of W^2 with eigenvalue 1.

Problem 3 (4 points) Assume the language modeling setup from lecture 1: I have a vocabulary $\mathcal{V}$, sequences $w_{1:T} \in \mathcal{V}^*$, embeddings $E_w \in \mathbb{R}^d$ for every $w \in \mathcal{V}$, and I make a probability distribution:

$$p(w \mid w_{<t}) = \frac{\exp(E_w^\top h_{<t})}{\sum_{w' \in \mathcal{V}} \exp(E_{w'}^\top h_{<t})} \tag{1}$$

$$h_{<t} = \frac{1}{t-1} \sum_{j=1}^{t-1} E_{w_j} \tag{2}$$

But oh no, I've initialized all of my E_w to zero. Still, I'm going to train all the E_w using gradient descent on the log-loss, as usual. (We've written this out again for your convenience, but it's identical to lecture 1.)

$$E \rightarrow E - \alpha \nabla_E \mathbb{E}_{(w_t, w_{<t}) \sim D} \left[-\log p(w_t \mid w_{<t}) \right]. \tag{3}$$

What is the value of this gradient? (You can write the value of a single gradient vector in $\mathbb{R}^d$ for an arbitrary word E_w.) Why does this make learning impossible? What is the probability distribution at initialization?

Solution:

Gradient (1 point): First, remember from lecture that the general gradient of the log-loss for a given embedding is:

$$\nabla_{E_w} - \log p(w \mid w_{<t}) = \nabla_{E_w} - \log \left(\frac{\exp(E_w^\top h_{<t})}{\sum_{w' \in \mathcal{V}} \exp(E_{w'}^\top h_{<t})} \right) \tag{4}$$

$$= -\left(1 - p(w \mid w_{<t})\right) h_{<t} \tag{5}$$

However, because all E_w are initialized to $\mathbf{0}_d$, we see that $h_{<t}$ is always $\mathbf{0}_d$ as well, making the full gradient $\mathbf{0}_d$:

$$h_{<t} = \frac{1}{t-1} \sum_{j=1}^{t-1} E_{w_j} = \frac{1}{t-1} \sum_{j=1}^{t-1} \mathbf{0}_d = \mathbf{0}_d \tag{6}$$

$$\nabla_{E_w} - \log p(w_t \mid w_{<t}) = -\left(1 - p(w \mid w_{<t})\right) (\mathbf{0}_d) = \mathbf{0}_d \tag{7}$$

As we did not ask you to show steps, *a full point was received if your solution indicated that the gradient was a 0 vector.*

Why learning is impossible? (2 points): A 0 vector gradient means that *no update will ever be made to these embeddings.* Given the update rule during gradient descent: $E \rightarrow E - \alpha(\mathbf{0}_d)$, so $E \rightarrow E$. Another way to think about this is that the update rule attempts to increase the magnitude of the similarity (e.g., dot product) between a given embedding and prefix representation. Because all E and $h_{<t}$ are 0, no changes can increase the product.
While a uniform initial probability distribution generally prevents learning for this model, the argument is more involved as to why this would prevent learning [a]. Additionally, this is not the same as a "vanishing gradient"; in this case, it is precisely 0. *Solutions received 1 point if they attributed lack of learning to the 0 gradient without explanation, and 2 points for indicating that it is because the embeddings will never be modified.*

Initial probability distribution (1 point): Finally, with embeddings initialized to all 0, we can determine the initial probability distribution $p(w \mid w_{<t})$. Note that the question is asking about the full probability distribution output by the model, not the distribution that weights are sampled from:

$$p(w \mid w_{<t}) = \frac{\exp(E_w^\top h_{<t})}{\sum_{w' \in \mathcal{V}} \exp(E_{w'}^\top h_{<t})} \tag{8}$$

$$= \frac{\exp(0)}{\sum_{w' \in \mathcal{V}} \exp(0)} = \frac{1}{\sum_{w' \in \mathcal{V}} 1} = 1/|\mathcal{V}| \tag{9}$$

Therefore, *each word in the vocabulary receives equal,* $\frac{1}{|\mathcal{V}|}$ *probability,* forming a *uniform distribution* over the vocabulary. Importantly, the probability distribution is not 0 for each word or undefined, as the $exp(0)$ (e^0) in the numerator and denominator become 1 (softmax will always maintain a probability distribution that sums to 1). *Full credit was received for stating the probability is* $\frac{1}{|\mathcal{V}|}$ *for all words, and half credit for stating that the distribution is uniform without correctly specifying the probability of each token, uniform over the vocabulary, or minor errors in the stated distribution. Solutions that said the probability was 0, 1, or undefinde do not form probability distributions and did not receive credit.*

[a] For non-zero, identical embeddings, the gradient is non-zero and updates can be made. The updates to each word embedding, however, are identical in *expectation*, preventing learning.

Problem 4 (4 points) You're training a tokenizer using the byte-pair encoding algorithm we saw in class. At one point, here's the set of counts of token pairs:

```
{
  "('pizza', 'pie')": 7,
  "('pie', 'ce')": 4,
  "('ce', 'pizza')": 3,
  "('pie', 'pizza')": 2,
  "('run', 'ning')": 2,
  "('pie', 'run')": 1,
  "('ning', 'run')": 1,
  "('ning', 'I')": 1,
  "('I', 'roh')": 1
}
```

This is all of the token pair counts. Given this information, what are the next two tokens to be added to the vocabulary? Consider this carefully, knowing that the counts will change after the first token is added.

Solution:

The new tokens are `pizzapie`, and then `pizzapiece`.
The key here is that adding the first token-pair means that the token counts for the second-most (pie, ce) will go down to zero because all of the `pie` tokens will be gone. Where the `pie` tokens were now are `pizzapie` tokens, so the pair {`pizzapie, ce`} will have the highest count.

Problem 5 (4 points) John has had what he thinks is a brilliant research idea: the Simple Language Model ™. It's written as follows. I have a vocabulary $\mathcal{V}$, sequences $w_{1:T} \in \mathcal{V}^*$, embeddings $E_w \in \mathbb{R}^d$ for every $w \in \mathcal{V}$, and I make a probability distribution:

$$p(w \mid w_{<t}) = \exp(E_w^\top h_{w_{<t}}) \qquad h_{w_{<t}} = \frac{1}{t-1} \sum_{j=1}^{t-1} E_{w_j} \tag{10}$$

First, let $w_{<t}$ be a prefix and $g \in \mathcal{V}$ be some word such that $g \notin w_{<t}$ and compute:

$$\nabla_{E_g} - \log p(g \mid w_{<t}) \tag{11}$$

Unfortunately, this is **not** a language model. Second, describe what requirement of a language model it's missing, and minimally change it to solve this problem. (Note: we're **not** talking about a lack of expressivity in the computation of $h_{w_{<t}}$).

Solution:

First to calculate the gradient:

$$\begin{aligned}
\nabla_{E_g} - \log p(g \mid w_{<t}) &= -\nabla_{E_g} \log p(g \mid w_{<t}) \\
&= \nabla_{E_g} - \log \exp(E_w^\top h_{w_{<t}}) \\
&= -\nabla_{E_g} E_w^\top h_{w_{<t}} && \text{(Not by the chain rule.)} \\
&= -h_{w<t}
\end{aligned}$$

This gradient calculation is much simpler than ones in the note precisely because our model is missing normalization.
It is completely possible that this model could output probabilities greater than one. Thus it is not a language model. In order to fix this, we need to normalize the probabilities of all possible next words to sum to one (and to bound by 1). This could be done by completing the softmax in our formula for $p(w \mid w_{<t})$, which would yield:

$$p(w \mid w_{<t}) = \frac{\exp(E_w^\top h_{w_{<t}})}{\sum_{i \in \mathcal{V}} \exp\left(E_i^\top h_{w_{<t}}\right)}$$

As a note, many people attempted to apply the chain rule in order to calculate this gradient. This is an unideal first step as it's much easier to cancel the $\log(\exp(\cdot))$. However nearly all who start this way end with the wrong result, so here is a demonstration of what the calculation via the chain rule would look like:

$$\begin{aligned}
\nabla_{E_g} - \log p(g \mid w_{<t}) &= -\nabla_{E_g} \log p(g \mid w_{<t}) \\
&= \frac{-1}{p(g \mid h_{w<t})} \left(\nabla_{E_g} p(g \mid h_{w<t})\right) && (1) \\
&= \frac{-1}{p(g \mid h_{w<t})} \left(\nabla_{E_g} \exp(E_w^\top h_{w_{<t}})\right) \\
&= \frac{-1}{p(g \mid h_{w<t})} \exp(E_w^\top h_{w_{<t}}) \left(\nabla_{E_g} E_w^\top h_{w_{<t}}\right) && (2) \\
&= \frac{-1}{p(g \mid h_{w<t})} p(g \mid h_{w<t}) \left(\nabla_{E_g} E_w^\top h_{w_{<t}}\right) \\
&= -\nabla_{E_g} E_w^\top h_{w_{<t}} \\
&= -h_{w<t}
\end{aligned}$$

Where (1) is the result of applying the chain rule to log and (2) is the result of applying the chain rule to exp. In the intervening lines, we are just applying the definition of $p(g \mid h_{w<t})$.

Problem 6 (4 points) In the *Tasks and evaluation* lecture, we learned about the BLEU score. Informally, it took a gold sequence $y \in \mathcal{V}^*$ and a model-predicted sequence $\hat{y} \in \mathcal{V}^*$ and it computed precisions for various n-grams (tuples of words of length n):

$$p_n = \frac{\text{count of } n\text{-grams that occur in } \textbf{both } y \text{ and } \hat{y}}{\text{count of } n\text{-grams that occur in } \hat{y}} \tag{12}$$

We computed n-grams for $n \in \{1, 2, 3, 4\}$, and then reported a score:

$$\frac{|\hat{y}|}{|y|} \sqrt[4]{p_1 p_2 p_3 p_4} \tag{13}$$

Intuitively, having only p_1 wasn't sufficient because a translation $\hat{y}$ could have shuffled all the words of y and be called a perfect translation. Why don't we use p_5, p_6, p_7, and on?

(This question has a lot of space below it not because it requires a long answer, but because the question afterward needs its own page.)

Solution:

As n increases, the number of n-grams in the predicted sequence $\hat{y}$ and the reference sequence y decreases. This leads to a much lower probability of exact n-gram matches between $\hat{y}$ and y. Consequently, for large n, the precision p_n often becomes 0.
Because the BLEU score uses the geometric mean of the n-gram precisions, if any $p_n = 0$, then the entire product becomes 0 and the BLEU score collapses to 0. (To receive full credit, you needed to clearly state this point.) This would make the BLEU score extremely sensitive to higher-order mismatches and less useful for evaluation.
(We gave partial credit if you mentioned the sparsity issue/diminishing returns but did not explicitly state that the BLEU score would collapse to 0.)

Problem 7 (3 points) I have a neural network of the following form. I have an input $x_0 \in \mathbb{R}^d$. I have parameters $\{\theta_1, \ldots, \theta_m\}$ that I've initialized randomly as $\theta_t \sim U[-0.01, 0.01]^d$, where each $\theta_t \in \mathbb{R}^d$, and I compute:

$$g_t = \begin{cases} 1 & \theta_t^\top x_{t-1} > 0 \\ -1 & \text{otherwise} \end{cases} \tag{14}$$

$$x_t = \sigma(x_{t-1} g_t) \tag{15}$$

where note that g_t is a scalar ($\mathbb{R}^1$), and where $\sigma(v)$ is the ReLU as discussed in class: an elementwise nonlinearity that takes the maximum of each element v_i and 0:

$$\sigma(v)_i = \max(0, v_i) \; \forall_{i=1}^d \tag{16}$$

As m, becomes large (that is, as we get more randomly sampled θ_t), provide an argument that x_m gets closer to the zero vector.

(This problem is more difficult than the others, and intended to show you how with deeply nonlinear systems at initialization, even with nonlinearities that don't *themselves* cause zero gradients, you can still get networks with zero gradients.)

(Extra note: as a reminder, the notation $U[-0.01, 0.01]^d$ means we sample a random value between -0.01 and 0.01 independently d times to form our vector.)

Solution:

The key components to know about this problem were:

- First, x_0 potentially has some (or zero) positive elements, some (or zero) negative elements, and the rest zero elements.
- No matter what if x is not the zero vector, there is a 50-50 chance that $\theta_t^\top x_{t-1} r 0$, so a 50-50 chance of g_t negating x_{t-1}.
- In the first step, x_0 to x_1, the elements of x_0 are either negated or not by g_t. If they're negated, then the ReLU sets all originally positive elements to zero. Otherwise, the ReLU sets all originally negative elements to zero.
- Now, x_t, $t \geq 1$ only has positive values.
- All values set to 0 by the ReLU stay zero.
- For all iterations $t > 1$, the gate g_t has a 50-50 chance of turning the (all positive) values of x_t to all negative. When this happens, they all get set to zero, so x_t is equal to the zero vector.
- So, as m grows, the probability that you *didn't* hit another negative g_t by the time you hit x_m decreases exponentially.

We still gave full points if you didn't realize that the decay of x to zero happens only in two chunks, and just becomes more likely over time, and instead only realized that there waw a 50-50 chance of flipping x each round and that some elements would be zeroed with each gate flip.
We assigned partial credit if you didn't state that the dot product of $\theta_t^\top x_{t-1}$ would be 50-50 positive or negative each round.
Missing various aspects of the key points above, but stating others, also led to partial credit.