Practice Exam Questions

Below are a sample of practice exam questions from previous years. Note that the set of covered topics can vary from year to year, and so the below list may contain questions on topics that are not covered this year (and therefore, will not be tested). The length of this document is **not** indicative of the length of the exam.

Short Questions

Q: **Define.** What is a language model?

A language model is a probability distribution over strings, consisting of tokens taken from a vocabulary V.

Q: "Teacher forcing" refers to the training approach where the model is trained with a loss function that only depends on the correct (i.e., ground truth) label. Why is teacher forcing used when pre-training a language model with the standard next-token language-modeling objective? Does teacher forcing lead to any issues during inference?

It feeds the *gold* previous token at every step, letting the model learn all conditional probabilities in parallel, which makes maximum-likelihood training (cross-entropy) faster and more stable. Teacher forcing enables large-scale unsupervised training on text data, where each subsequent token can be used a self-supervising example. Yes, during inference the model will not have access to the gold tokens, leading exposure bias. The model may not learn how to differentiate between alternatives (i.e., which "incorrect" answers are better/worse than others).

Q: Let's say we train a language model with some amount of compute C and obtain cross-entropy loss L on validation data. **True or false:** For any loss $L' < L$, there exists some amount of compute $C' > C$ such that the resulting model will achieve cross-entropy loss L'. Explain.

False. Natural language has non-zero entropy. That is, there will always be some uncertainty when predicting the next token, no matter how powerful the model is. In the limit of infinite compute $C' \to \infty$, an expressive model will achieve loss approaching the entropy but never lower.

Q: Assume an LSTM attention-based encoder decoder network that takes in as input an article (say, Wikipedia page) and a question about it and generates output tokens corresponding to the answer. Argue, in one sentence, why we can and why we can't think about the attention weights as explanations for the answer.

Pro: Attention weights capture the strength of relationship between an input and generated text. Sentences in the article with the highest attention weights are most prominent

when generating the output text, and can be viewed as explaining the parts of the input used to generate the answer. Con: (1) The attention weights could be uniform (or close to that) and as a result not informative (2) The generated answer can also rely on additional things (decoder parameters), so the attention does not fully explain the answer.

Q: What is the purpose of the residual connections in the transformer?

The residual connections help to alleviate the vanishing/exploding gradient problem. They allow gradients to be back-propagated without having their magnitudes increased or decreased.

Q: Give examples of two sentences that have high BLEU score but are semantically very different.

`'I really love these flowers'`, `'I really hate these flowers'`

Q: **Explain.** How is semantic parsing (mapping from input text to a logical form) different from semantic role labeling? In what sense are they similar?

Semantic role labeling is a "shallow" form of semantic parsing, where not all semantic information is extracted from the natural language text. In SRL, we only extract the events/actions and the semantic arguments of those events/actions: E.g., Who is the entity that is performing the action? What is the location of the event? What is the entity that is being used as an instrument in the action? Semantic parsing is similar to SRL in that semantic role label information is a subset of the information in the full logical form. However, the logical form also contains information about the relations between multiple events/actions, as well as truth-functional meaning, such as negation, quantifiers, etc.

Q: What is the purpose of the causal attention mask in autoregressive language models?

In autoregressive language modeling, the embedding of the token at position i is used to predict the $(i+1)^{\text{th}}$ token. The causal mask is necessary as it prevents the i^{th} token from attending to the $(i+1)^{\text{th}}$ token. Therefore, with the causal mask, each token is not allowed to "cheat" and simply copy the token from the next position. Instead, the model must rely only on the information in the tokens at position $j \leq i$ to predict the $(i+1)^{\text{th}}$ token.

Q: For a batch of b input sequences, each with length n, compare the per-forward-pass computational complexity of

(i) an RNN layer (assume a simple, vanilla RNN), and
(ii) a Transformer self-attention layer (i.e., no MLP).

Assume both models have model dimension d. How does forward-pass complexity change when you utilize parallel computation over the sequence during training?

- **RNN:** $O(b\,n\,d^2)$, but operations across time steps are *sequential* (depth n).
- **Transformer self-attention:** $O(b\,n^2\,d)$, with an $n \times n$ attention matrix, yet all n positions are computed *in parallel* in a few large matrix–matrix multiplies.

All n positions in a Transformer are processed *in parallel* using large batched matrix multiplies that utilize GPU cores, whereas an RNN must run n dependent steps sequentially, limiting parallelism.

Long Question 1

Suppose we implement an n-gram model using an MLP. The inputs are 1-hot encoded tokens, and so the input dimension of the MLP is nV where V is the size of the vocabulary. There is one hidden layer with dimension d, followed by the output layer. The MLP applies a ReLU on the activations in the hidden layer.

Q: What is the size of the output layer?

Since this is an n-gram model, the output is a distribution over the next token and so the output layer has dimension V.

Q: How many parameters are in this model?

There is a linear layer mapping from inputs of dimension nV into d-dimensional vectors, which requires an $nV \times d$-dimensional weight matrix (and optionally a d-dimensional bias vector). The ReLU does not have any parameters. The second linear layer maps d-dimensional activations into the V-dimensional output. Thus the second layer requires a $d \times V$-dimensional weight matrix (and optionally a V-dimensional bias). Thus the total number of parameters is $ndV + dV = (n+1)dV$ (plus $d + V$ bias parameters).

Q: How will the model behave if d is set too small?

The model will be very inexpressive and not very accurate in predicting the next token.

Q: How will the model behave if d is set too large?

If there is insufficient training data or compute, the model will be undertrained and easily overfit.

Q: How will the model behave if n is set too small?

If n is too small, the model will be unable to access information from tokens preceding the last n tokens. Thus, the model will have limited access to context.

Q: Suppose we have a pretrained model where n, V, and d are sufficiently large such that full fine-tuning is not feasible due to GPU memory constraints, but you have sufficient memory to load the model and run forward passes. Given a supervised fine-tuning dataset, how would you modify the model to enable you to fine-tune it? Be specific about which parameters you would add/modify/remove.

The original model can be written as $f(x) = R(xW_1 + b_1)W_2 + b_2$ where $x \in \mathbb{R}^{nV}$ is the concatenated input 1-hot embeddings. We can modify the model by adding terms to the first linear layer: $f(x) = R(xW_1 + xAB + b_1)W_2 + b_2$. Here, $A \in \mathbb{R}^{nV \times r}$ and $B \in \mathbb{R}^{r \times d}$ where r is much smaller than d or nV. During fine-tuning, we only learn A and B, and all other parameters are kept frozen. This is the basic idea behind LoRA.

(there are multiple acceptable answers here; for example, prefix tuning, or replacing W_1 and/or W_2 with low-rank approximations would also be acceptable)

Long Question 2

A probabilistic context-free grammar (PCFG) is a context-free grammar where each rule is assigned a probability. The probability of a derivation/parse tree is the product of the

probabilities of the rules in the tree (i.e., the distribution of each rule is i.i.d.). Consider the following PCFG:

Rule	Probability
S → NP VP	1.0
VP → V	0.2
VP → V NP	0.6
VP → VP PP	0.2
NP → NN	0.8
NP → NP PP	0.2
PP → IN NP	1.0
V → 'sees'	0.5

Rule	Probability
V → 'ate'	0.5
NN → 'John'	0.25
NN → 'Mary'	0.25
NN → 'glasses'	0.25
NN → 'kitchen'	0.25
DT → 'the'	1.0
IN → 'with'	0.5
IN → 'in'	0.5

Q: Given the sentence 'John sees Mary with glasses', draw **two** valid parse trees according to the grammar above.

There are two ways to parse this sentence, corresponding to the PP-attachment problem. The PP 'with glasses' can be attached to the verb phrase (containing 'sees'), indicating that John had glasses. The other option is that the PP is attached to the NP (containing Mary) and as a result—Mary had the glasses.

The two valid parse trees are:

```
[S
  [NP [NN 'John']]
  [VP
    [VP
      [V 'sees']
      [NP [NN 'Mary']]
    ]
    [PP
      [IN 'with']
      [NP [NN 'glasses']]
    ]
  ]
]

[S
  [NP [NN 'John']]
  [VP
    [V 'sees']
    [NP
      [NP [NN 'Mary']]
      [PP
        [IN 'with']
        [NP [NN 'glasses']]
      ]
    ]
  ]
]
```

(you may also draw these as trees)

Q: What is the probability of each of the above parse trees? Which parse is more likely according to the PCFG given above? (note: if a question like this is on the exam, we will allow you to use a calculator)

$p(\text{first tree}) = p(S \rightarrow NP\ VP) \cdot p(NP \rightarrow NN) \cdot p(NN \rightarrow \text{`John'}) \cdot p(VP \rightarrow VP\ PP) \cdot p(VP \rightarrow V\ NP) \cdot p(V \rightarrow \text{`sees'}) \cdot p(NP \rightarrow NN) \cdot p(NN \rightarrow \text{`Mary'}) \cdot p(PP \rightarrow IN\ NP) \cdot p(IN \rightarrow \text{`with'}) \cdot p(NP \rightarrow NN) \cdot p(NN \rightarrow \text{`glasses'})$

$= (1)(\frac{4}{5})(\frac{1}{4})(\frac{1}{5})(\frac{3}{5})(\frac{1}{2})(\frac{4}{5})(\frac{1}{5})(1)(\frac{1}{2})(\frac{4}{5})(\frac{1}{4})$

$= \frac{192}{1000000}$.

$p(\text{second tree}) = p(S \rightarrow NP\ VP) \cdot p(NP \rightarrow NN) \cdot p(NN \rightarrow \text{`John'}) \cdot p(VP \rightarrow V\ NP) \cdot p(V \rightarrow \text{`sees'}) \cdot p(NP \rightarrow NP\ PP) \cdot p(NP \rightarrow NN) \cdot p(NN \rightarrow \text{`Mary'}) \cdot p(PP \rightarrow IN\ NP) \cdot p(IN \rightarrow \text{`with'}) \cdot p(NP \rightarrow NN) \cdot p(NN \rightarrow \text{`glasses'})$

$= (1)(\frac{4}{5})(\frac{1}{4})(\frac{3}{5})(\frac{1}{2})(\frac{1}{5})(\frac{4}{5})(\frac{1}{4})(1)(\frac{1}{2})(\frac{4}{5})(\frac{1}{4})$

$= \frac{192}{800000}$.

Therefore, the second parse is more likely.

Q: Suppose we add the rules $NN \rightarrow$ 'We' and $V \rightarrow$ 'see' to the grammar. Give an example of a sentence in the language of the resulting grammar that is not grammatical with respect to English.

`'We sees John with glasses.'`

Q: How could you modify the grammar so that the ungrammatical sentences in the above question are excluded from the language? (without excluding grammatical sentences such as 'We see John with glasses')

The NP, NN, VP, and V nonterminals can each be split into two, depending on the grammatical number (singular vs plural). The resulting grammar would have the following rules:

$S \rightarrow NP_{\text{sg}}\ \ VP_{\text{sg}}$
$S \rightarrow NP_{\text{pl}}\ \ VP_{\text{pl}}$
$NP_{\text{sg}} \rightarrow NN_{\text{sg}}$
$NP_{\text{pl}} \rightarrow NN_{\text{pl}}$
$NP_{\text{sg}} \rightarrow NP_{\text{sg}}\ \ PP$
$NP_{\text{pl}} \rightarrow NP_{\text{pl}}\ \ PP$
$NN_{\text{sg}} \rightarrow$ 'John'
$NN_{\text{sg}} \rightarrow$ 'Mary'
$NN_{\text{pl}} \rightarrow$ 'glasses'
$NN_{\text{sg}} \rightarrow$ 'kitchen'
$NN_{\text{pl}} \rightarrow$ 'We'
$VP_{\text{sg}} \rightarrow V_{\text{sg}}$
$VP_{\text{pl}} \rightarrow V_{\text{pl}}$
$V_{\text{sg}} \rightarrow$ 'sees'
$V_{\text{pl}} \rightarrow$ 'see'

Long Question 3

The textual entailment is a general language understanding problem, in the sense that many other semantic understanding tasks can be reduced to it. In textual entailment, we are given a premise P and a hypothesis H. The task is to determine whether P entails/implies H, P contradicts H, or neither. Let's think about the coreference resolution problem. "The mission that **Mary** set out to do after talking to John, failed, it was just too hard, and **she** was very upset about it" All the bold words and all the underlined words are coreferent (i.e., mentions of the same entity).

Q: Can you rewrite this problem as a textual entailment task? You can use this example to explain your work.

Consider the sentence where the anaphora are replaced with the referent noun phrases: 'The mission that Mary set out to do after talking to John, failed, the mission was just too hard, and Mary was very upset about it.' If the anaphora are substituted with the correct referents, then the original sentence should entail the new sentence. In fact, the two sentences should be semantically equivalent, and so the new sentence should also entail the original sentence.

Q: We have trained our textual entailment system over a large collection of examples, such as the SNLI dataset. The data contains examples such as

```
{'premise': 'The sisters are hugging goodbye while holding to-go
             packages after just eating lunch.'
 'hypothesis': 'Two women are embracing while holding to-go
                packages.'
 'label': entailment}
```

Since this is a large collection (>500K pairs) we were able to train a very strong entailment system, that come close to human performance on this dataset. Explain. Does this mean that coreference resolution is a solved task?

No, as even if the model performs well on this dataset, there is no guarantee that it would perform well on out-of-distribution data, especially as compared to humans. In addition, matching human performance does not necessarily imply the task is "*solved*."

Q: Consider the problem of resolving prepositional attachment ambiguity, such as in the sentences "Sally saw Alex with binoculars." Can we write a textual entailment example that would resolve this ambiguity? If so, how?

Yes, we can write the sentence as the premise and write a hypothesis such as "Sally uses binoculars". The textual entailment label will be `entails` if "with binoculars" attaches to "saw". However, it may still be possible that both Sally and Alex have binoculars. Therefore, a better example would be to let the hypothesis be "Sally does not use binoculars" which would be a `contradiction` if "with binoculars" attaches to "saw." Similarly, the hypothesis "Alex does not have binoculars" would be a `contradiction` if "with binoculars" attaches to "Alex".

Q: Give an example of an NLP task that can not be easily reduced to textual entailment.

Language modeling. Any task that does not rely on semantic information, such as "How many vowel characters are in the word *epithelialization*."

(any one answer will suffice)

Long Question 4

Imagine training a large transformer-based model, but we have insufficient GPU memory to train the model using `float32` precision. Instead we attempt to train the model with `bfloat16` precision but we find that as training progresses, the training loss initially decreases, but after some time, it begins increasing suddenly. We are using gradient descent with a learning rate that is sufficiently small to prevent any instability due to the step size being too large.

Q: (2 points) What is the cause of this instability during training?

Instability is caused by very small gradients being rounded to 0 due to lack of precision in `bfloat16`.

Q: (2 points) Provide a suggestion to improve stability without reducing the number of parameters.

We can use mixed-precision training. Store the weights in `float32` but store all other variables (e.g., gradients, activations) in `bfloat16`. We scale the gradients by a constant factor $\gamma > 1$ such that the values very close to zero are shifted such that they are not rounded to 0 in `bfloat16`. When computing the gradient update, we undo the scaling: $w_{\text{new}} = w_{\text{old}} - \frac{1}{\gamma} g$ where g are the scaled gradients.

Q: (2 points) Suppose we have a pretrained model and we only wish to perform inference. What are the advantages and disadvantages of quantizing the model into `int8` precision? Suppose the GPU supports accelerated matrix operations using `int8`.

Inference is much faster using `int8` as compared to `bfloat16` or `float32`. The quantized model would also be much smaller and can be run on hardware with less GPU memory. However, there is an approximation cost and the performance of the quantized model will not be as good as that of the original model.

Q: (2 points) What are the advantages and disadvantages of quantizing the model into `int4` or `float4` precision? Suppose the GPU **does not** support accelerated matrix operations using `int4` or `float4`.

Due to lack of hardware acceleration support, inference will not be faster. But the quantized model will still be much smaller and can be run on hardware with less GPU memory. The approximation cost would be greater and the performance of the quantized model will deteriorate even further.

Q: (2 points) Consider altering the model to use mixture-of-experts (MoE) in the feedforward blocks of each transformer layer. Suppose we set the size of each expert such that the total number of parameters does not change. Does the resulting model require more, less, or the same amount of memory during *training*?

The number of parameters is unchanged, but for each input, a large portion of the activations in the FF block will be unused, and so less memory is required to store the (useful) activations and (non-zero) gradients.

Q: (2 points) Is the MoE model more expensive **during inference**? Less expensive? Or the same? (in terms of memory and compute)

Inference is faster due to the fact that only a small number of experts are needed for each forward pass. The memory cost in terms of the number of parameters is unchanged, since we still need to load all experts into memory, as different experts may be needed for different forward passes (and we can't tell which ones are needed a priori). However, since the number of useful activations in the FF block per input is much smaller as compared to a dense model, less memory is required to store these useful activations.

Long Question 5

You are tasked with building a medium-size generative language model (target model) for *truth*-value checking of mathematical statements. An example:

$$\text{Let } A \in \mathbb{R}^{n \times n}. \text{ If } A^2 \text{ is diagonalizable over } \mathbb{R}, \text{ then is } A \text{ diagonalizable over } \mathbb{R}?$$

You are provided with a pre-trained language model as a starting point (e.g., `GPT-2`), with access to its model parameters for fine-tuning. You are also provided with train/dev/test sets each containing a statement and an associated label indicating whether the statement is `true` or `false`. You also have API access to a powerful LLM (e.g., `gpt5-thinking`) during the training phase.

Let's assume that the problems are challenging, such that merely training the target model on the statement and the label yields poor performance during inference.

Q: (3 points) How would you fine-tune the target model, by briefly explaining the input and output template for the powerful LLM and the target model? How would you ensure that, during inference, the `true`/`false` label can be extracted from the target model?

We can distill the smaller model using the larger one: For each example, use the label to transform the problem to theorem proving and prompt the LLM to generate the proof to train the target model.

`LLM prompt:` Prove/disprove (based on the train label) the following statement: ⟨problem statement⟩

`LLM output:` ⟨proof⟩

`target model input` ⟨problem statement⟩

`target model output` ⟨LLM-generated proof⟩ + "Hence this statement is True/False"

Q: (2 points) Given API calls to an LLM can be expensive, how can you minimize the number of calls to the powerful LLM during training? Briefly discuss the high-level training procedure.

Perform initial few rounds of training using the above approach and as training progresses and then infer on the training set after each round, and only call the powerful LLM for the problems for which the target model incorrectly predicts the labels.

Q: (2 points) How can your above approach be made more efficient (in terms of number of LLM forward passes/API calls and training iterations) if you have an access to a clustering model – which is sufficiently capable to accurately classify mathematical sentences into topics corresponding to subfields (e.g., Probability Theory, Algebra, Set Theory, etc)?

Cluster the training examples and only call powerful LLM for few examples or centroids to train the target model and then infer on the remaining examples. Repeat this example selection process for the different examples from the clusters on which the target model performs most poorly.

Q: (2 points) Can you further improve the efficiency of fine-tuning if the API provided access to the logits (i.e., the log probabilities) over the full vocabulary for each output token of the LLM? If so, how?

Rather than distilling the smaller model on each observed token produced by the larger model, we can use the LLM's predicted distribution for each token. Using this approach, fewer training iterations are required to reach the same accuracy.