Practice Exam Questions

Final Exam Date: Mon 05/04, 2026 7:00p - 9:00p @ PHYS 203

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?

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.

Q: Explain shortly what the n-gram assumption is.

The n-gram assumption is the (Markov) assumption that a given word's probability is only dependent on the $n-1$ previous words before it.

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: **True or false:** LoRA is a method used to approximate changes to weight matrices in Transformers to lower the memory cost during pre-training.

False. LoRA is used in *post-training* (e.g., supervised fine-tuning).

Q: What is the purpose of the causal attention mask in autoregressive (decoder-only) 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: (4 points) Explain the benefits and drawbacks of training a transformer using an extremely large and extremely small batch size.

A large batch size can better estimate the gradient and stabilize training at the cost of memory. A small batch size may underutilize hardware parallelism and yield a very noisy gradient estimate, but it may reduce memory cost.

Q: Let's say we train a language model with $N = 10^8$ parameters and a training data set of size $D = 10^9$ for $T = 10^5$ iterations of gradient descent, and we obtain cross-entropy loss $L = 3.3$ on validation data. Let's say we want to scale up the model so that it achieves a lower cross-entropy loss of $L' = 0.7$. **True or false:** If we increase the model size N, data set size D, and training iterations N to be sufficiently large, we can obtain a model with cross-entropy loss $L' = 0.7$. 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: Consider the following NLP methods: (1) bi-gram language model implemented using a feedforward neural network (assume a single hidden layer) (2) Word2Vec embedding. Identify one aspect in which the two methods are similar and one in which they are different.

Similar: both represent the relationship between pairs of words appearing in a local context using a dense vector representation (hidden layer). Different: Bigram models learn a directional relationship, while Word2Vec uses skip-grams.

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

Consider the following context-free grammar:

S → NP VP	V → 'ate'
VP → V	NN → 'John'
VP → V NP	NN → 'Mary'
VP → VP PP	NN → 'glasses'
NP → NN	NN → 'kitchen'
NP → NP PP	DT → 'the'
PP → IN NP	IN → 'with'
V → 'sees'	IN → 'in'

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: 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:

$S \rightarrow NP_{\text{sg}} \ VP_{\text{sg}}$
$S \rightarrow NP_{\text{pl}} \ VP_{\text{pl}}$
$NP_{\text{sg}} \rightarrow NN_{\text{sg}}$
$NP_{\text{pl}} \rightarrow NN_{\text{pl}}$

$NP_{sg} \rightarrow NP_{sg} \quad PP$
$NP_{pl} \rightarrow NP_{pl} \quad PP$
$NN_{sg} \rightarrow$ 'John'
$NN_{sg} \rightarrow$ 'Mary'
$NN_{pl} \rightarrow$ 'glasses'
$NN_{sg} \rightarrow$ 'kitchen'
$NN_{pl} \rightarrow$ 'We'
$VP_{sg} \rightarrow V_{sg}$
$VP_{pl} \rightarrow V_{pl}$
$V_{sg} \rightarrow$ 'sees'
$V_{pl} \rightarrow$ '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. Consider the example "The hawk couldn't catch up to the rabbit because it was too fast." 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 hawk couldn't catch up to the rabbit because the rabbit was too fast.' 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 comes close to human performance on this dataset. Does this mean that coreference resolution is a solved task? Explain.

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*." Or any task that has a large space of possible answers (e.g., the set of potential answers in math problems is the set of all numbers, which is much larger than the set of 3 classes in textual entailment).
(any one answer will suffice)

Long Question 4

You are building a medium-size transformer-based generative language model for checking the *truth*-value of mathematical statements. Each example contains a mathematical statement and a label indicating whether the statement is `true` or `false`. For example:

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

You are given:

- a pretrained language model as the target model, with access to its parameters for fine-tuning (e.g. `Phi-4`);
- train/dev/test data containing mathematical statements, each with a `true`/`false` label;
- API access to a powerful LLM (e.g. `GPT-5.5`) during training, but not during final inference;
- limited GPU memory for training and deployment.

Assume that you have tried directly fine-tuning the target model on the given labeled training data of mathematical statements, but the resulting model gives poor performance, because the problems require mathematical reasoning.

Q: (3 points) For a decoder-only Transformer model, what training objective is typically used during pre-training?

The typical training objective used for transformer models is the cross-entropy loss between the predicted token and the ground truth token (next token prediction).

Q: (3 points) During training, you try to train the target model entirely using `bfloat16` precision to save memory. The training loss initially decreases, but later suddenly increases, even though the learning rate is small enough that the spike in the loss is not caused by taking steps that are too large. Conceptually, what can cause this instability? Why does a small learning rate not fully solve the issue?

The instability is caused by numerical precision limits. In low precision, small gradients, small weight updates, or accumulated optimizer statistics may be rounded away or represented inaccurately. As a result, the optimizer may stop applying useful small updates, or may apply noisy/inaccurate updates. This is different from the usual instability caused by a learning rate that is too large: even if the intended update is small, the low-precision representation may not be able to represent it accurately.

Q: (3 points) Suggest one way to improve training stability without reducing the number of model parameters. Explain the basic idea and the memory tradeoff.

Use a higher-precision format, such as `float32`, during training. Higher precision can represent very small gradients and updates more accurately, so optimization is less likely to become unstable due to rounding or underflow.

The number of model parameters is unchanged, but each parameter and intermediate value uses more memory. Therefore, training is more stable, but it requires more GPU memory and may be slower.

Q: (3 points) After training, you only need to run inference. Compare quantizing the model to `int8` versus `int4`/`float4`. Assume the GPU supports accelerated matrix operations for `int8`, but does **not** support accelerated matrix operations for `int4` or `float4`.

Quantizing to `int8` can reduce model size and memory usage, and because the GPU supports accelerated `int8` operations, inference can also become faster. The disadvantage is that quantization is an approximation, so model quality may drop compared to the original higher-precision model.

Quantizing to `int4` or `float4` can reduce the model size even further, allowing the model to fit on smaller hardware. However, because the GPU does not accelerate `int4` or `float4` matrix operations, inference may not become faster and could even require extra dequantization overhead. The approximation error is also larger, so the model's accuracy or reasoning quality may degrade more than with `int8`.

Q: (4 points) Since the math problems require reasoning, you decide to use the powerful LLM as a teacher during training. Describe a fine-tuning strategy for the target model. Your answer should specify:

- what you ask the powerful LLM to produce;
- what the target model receives as input;
- what the target model is trained to output;
- how you make sure the final `true`/`false` label can be extracted during inference.

Use the powerful LLM to generate reasoning traces or short proofs/disproofs for the training examples (i.e., *distillation*). Since the training label is known, the teacher can be prompted to generate reasoning that supports the correct label.

For example:

`Teacher LLM input:` The following mathematical statement is labeled `true`/`false`. Provide a concise proof or disproof that justifies this label. Statement: ⟨statement⟩

`Teacher LLM output:` ⟨proof or disproof⟩ followed by a standardized final line such as `Final answer: true` or `Final answer: false`.

Then train the target model with:

`Target model input:` ⟨statement⟩

`Target model output:` ⟨teacher-generated reasoning⟩ + `Final answer: true/false`.

At inference time, the target model is given only the statement. It generates reasoning and ends with the standardized final answer line. The label can then be extracted by parsing the final line. The standardized output format is important because otherwise the model may produce a correct explanation but an ambiguous final answer.

Q: (4 points) API calls to the powerful LLM are expensive. Give one way to reduce the number of teacher calls while still improving the target model. Also explain how access to the teacher model's output probabilities/logits could make training more efficient.

Approach 1: Active selection of examples. Instead of calling the powerful LLM for every training example, first train the target model on a smaller teacher-labeled subset. Then evaluate the target model on the training or development set and call the teacher mainly for examples the target model gets wrong or is uncertain about. This focuses expensive teacher calls on examples that are most useful for improving the model.

Approach 2: Clustering. If examples can be clustered by mathematical topic, such as algebra, probability, or set theory, then we can sample representative examples from each cluster rather than calling the teacher on every example. Later, we can spend more teacher calls on clusters where the target model performs poorly.

If the API provides logits or output probabilities, the target model can learn from the teacher's full probability distribution rather than only the single token that the teacher sampled. This is knowledge distillation. The distribution tells the student not only the teacher's chosen answer, but also which alternative tokens or continuations the teacher considered plausible. This gives a richer training signal per API call and can reduce the number of examples or training iterations needed to reach good performance.