Instruction-Tuning

Estimated Time: 45 minutes

Learning objectives

After completing this lab, you will be able to:

- Explain instruction masking and why it is used in training transformer models
- Explain how instruction masking improves model performance by focusing on response tokens
- Identify the role of loss function representation in instruction masking
- Describe how instruction masking is applied in a chatbot or other real-world scenarios
- Analyze the impact of cross-entropy loss in training models with instruction masking

Introduction

As transformer-based models evolve, fine-tuning techniques such as instruction-tuning have become crucial for improving response quality in AI systems. However, traditional loss functions often treat all tokens in a sequence equally, which can lead to inefficiencies in learning. Instruction masking addresses this by selectively focusing loss computation on response tokens while ignoring instruction tokens. This ensures that models prioritize generating contextually accurate and meaningful responses without overfitting to repetitive instruction patterns.

Instruction masking

Instruction masking is a sophisticated technique employed during instruction tuning to enhance the learning process of transformer models. By focusing the loss calculation on specific parts of the output sequence—such as response tokens—while ignoring instruction tokens, instruction masking ensures that the model prioritizes generating accurate and contextually relevant responses.

Why use instruction masking?

- **Focus on responses:** The model learns to generate precise and meaningful responses.
- **Efficient learning:** Reduces computational overhead by excluding irrelevant tokens from loss calculations.
- **Prevents overfitting:** This minimizes the risk of the model overfitting to instruction tokens, which are often repetitive or less informative.

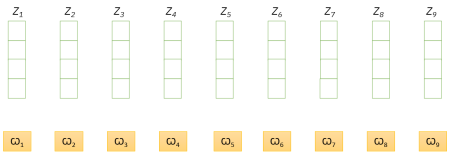
Instruction masking and loss function representation

Loss function representation

The total loss function for instruction masking is expressed as the sum of individual loss functions for each token pair:

```
L(ω1,z1) + L(ω2,z2) + L(ω3,z3) + ⋯ + L(ωn,zn)
```

$$\mathcal{L}(\omega_1, z_1) + \mathcal{L}(\omega_2, z_2) + \mathcal{L}(\omega_3, z_3) + \mathcal{L}(\omega_4, z_4) + \mathcal{L}(\omega_5, z_5) + \mathcal{L}(\omega_6, z_6) + \mathcal{L}(\omega_7, z_7) + \mathcal{L}(\omega_8, z_8) + \mathcal{L}(\omega_9, z_9)$$



The image represents elements involved in instruction masking, showing a structured relationship between
ω values (highlighted in yellow) and
Z values (depicted as vertical stacks). At the top, a summation of loss functions $L(\omega,Z)$ is displayed, indicating an optimization process or learning objective tied to these elements.

```
L(ω1 , z1) + L(ω2 , z2) + L(ω3 , z3) + L(ω4 , z4) + L(ω5 , z5) + L(ω6 , z6) + L(ω7 , z7) + L(ω8 , z8) + L(ω9 , z9)
```

Where:

- L: Loss function
- ω (omega): Input tokens/instructions
- z: Output tokens/predictions

Visual components

`Vertical bars` (z_1–z_9): Masked positions.

`Yellow boxes` (ω_1–ω_9): Input context.

`Combined pairs`: Position-specific loss calculations.

How does the instruction masking work?

To understand how instruction masking works in a real-time scenario, let's break it down step by step using a practical example.

Real-time scenario: Chatbot for answering questions

A chatbot is trained to answer user queries using instruction-tuning. Instruction masking ensures that the model prioritizes response generation over memorizing instructions.

Example: Input sequence

```
Instruction: Which is the largest ocean?
Response: The Pacific Ocean</s>
```

Here, the input consists of two parts:

- **Instruction:** The user's question or command (`Which is the largest ocean?`).
- **Response:** The expected answer generated by the model. (`The Pacific Ocean</s>`).

Masking process:

In instruction masking, the goal is to ensure that the model learns only from the response tokens while ignoring the instruction tokens. Here's how it works:

- **Instruction tokens are masked (loss ignored):**
  - Tokens from the instruction (e.g., `Which`, `is`, `the`, `largest`, `ocean`, `?`) are ignored during loss calculation.
  - Prevents overfitting to instruction formats.
- **Response tokens remain unmasked (loss calculated):**
  - Tokens from the answer (e.g., `The`, `Pacific`, `Ocean`, `</s>`) are included in loss calculation.
  - Trains the model to generate accurate, context-aware responses.

Processing steps:

Let's walk through the steps involved in processing the input sequence with instruction masking:

- **Tokenization:**

The input sequence is split into tokens (subwords or words) using a tokenizer.

Example:

Instruction: `["Which", "is", "the", "largest", "ocean", "?"]`

Response: `["The", "Pacific", "Ocean", "</s>"]`

- **Encoding:**

Each token is converted into a numerical representation (embedding) that the model can process.

Example:

"Which" → `[0.23, 0.45, ...]`

"Pacific" → `[0.67, 0.89, ...]`

- **Position embedding:**

Positional information is added to the tokens to preserve the order of the sequence.

Example:

"Which" (Position 1) → `[0.23, 0.45, ...] + [0.1, 0.2, ...]`

"Pacific" (Position 7) → `[0.67, 0.89, ...] + [0.8, 0.9, ...]`

- **Context preservation:**
  The model processes the tokens through its layers (e.g., transformer layers) to capture contextual relationships.

Example:

The model understands that "largest" is related to "ocean" and "Pacific".

- **Token prediction:**
  The model predicts the next token in the sequence.

Example:

After processing the instruction, the model predicts "The Pacific Ocean" as the next tokens.

- **Loss calculation:**

The model calculates the loss for each predicted token against the valid token.

Example:

For the token "Pacific", if predicted with 0.7 probability: $L(\omega, z) = -\log(0.7) = -(-0.357) = 0.357$

For an incorrect prediction (e.g., "Atlantic" with 0.2 probability): $L(\omega, z) = -\log(0.2) = -(-1.609) = 1.609$

- **Masking application:**

The model applies instruction masking to focus loss calculation on response tokens only.

Example:

Masked tokens: `["Which", "is", "the", "largest", "ocean", "?"]`

Unmasked tokens: `["The", "Pacific", "Ocean", "</s>"]`

Cross-entropy loss in instruction masking

Cross-entropy loss is commonly used for training models with instruction masking.
It measures how well the model predicts the correct output by comparing the predicted probability distribution with the proper labels.
The masked tokens (ignored parts of the input) are not included in the loss computation, preventing them from affecting learning.

Cross-entropy loss (Mathematical formulation)

The cross-entropy loss measures how well the model's predicted probabilities match the true distribution. For a single token, it is defined as:

$$\text{Cross-Entropy Loss} = -\sum_{i=1}^{V} y_i \log(p_i)$$

Where:

y_i: True probability distribution (one-hot encoded; 1 for the correct token, 0 for others).

p_i: Predicted probability for token i.

V: Vocabulary size.

Illustrative example: Question answering

We have a sequence of ground truth tokens (correct words in a sentence) and predicted tokens (what the model generates).

Example sentence:

"Which is the largest ocean? The Pacific Ocean."

Let's break down the concepts with the same example:

- **Tokens:** The input and the expected answer are sequences of tokens (words or sub-words). For instance:
  - Input (Instruction Tokens): "Which", "is", "the", "largest", "ocean?"
  - Expected Answer (Answer Tokens): "The", "Pacific", "Ocean", "."
- **Model predictions:** The model generates predictions (ω_i) for each token in the expected answer. These predictions are probability distributions over the vocabulary.
- **Ground truth:** The ground truth (z_i) is a one-hot vector indicating the correct token at each position in the answer.

This sentence consists of both an instruction and an answer.

Index	Ground Truth Token (x_i)	Model Prediction (o_i)
x_1	Which	o_1
x_2	is	o_2
x_3	the	o_3
x_4	largest	o_4
x_5	ocean?	o_5
x_6	The	o_6
x_7	Pacific	o_7
x_8	Ocean	o_8
x_9		o_9

- The first five tokens (x_1 to x_5) are **instruction tokens** (question part).
- The last four tokens (x_6 to x_9) are **answer tokens** (response part).
- The model generates predictions o_1, o_2, ..., o_9, corresponding to each token.

Example calculation

We are dealing with a sequence prediction problem, where a model predicts words or tokens, and we compute how good or bad its predictions are using a loss function.

Step 1: Understanding the input and predictions

Input:

- "Which is the largest ocean?" (this is o_5)

Possible answer probabilities (as given by the model):

"Pacific Ocean" → 0.7 ✅ (correct answer)
"Atlantic Ocean" → 0.2
"Indian Ocean" → 0.1

Since the correct answer is "Pacific Ocean", the model should have assigned a probability as close to 1.0. However, it gave 0.7.

Step 2: Understanding the function

We use the negative log-likelihood (NLL) loss function, defined as:

$$L(a_i, z_i) = -\log P(z_i \mid a_i)$$

This function measures how "wrong" the model's predicted probability is for the correct answer. The logarithm function is used because:

- It penalizes lower probabilities more.
- It ensures that probabilities close to 1 have a low loss (indicating a good prediction).
- It is a standard choice in classification problems (such as cross-entropy loss).

Step 3: Applying the formula

Since the model predicted P("Pacific Ocean") = 0.7, we substitute:

$$L(a_i, z_i) = -\log(0.7)$$

Using the natural logarithm

$$\log(0.7) \approx -0.3567$$

So,

$$L(a_i, z_i) = -(-0.3567) = 0.3567$$

Interpretation

- A lower cross-entropy loss means the model is confident about the correct answer.
- If the "Pacific Ocean" probability had been higher, the loss would be lower.
- If the "Pacific Ocean" probability were lower (e.g., 0.1 instead of 0.7), the loss would be higher, indicating the model is unsure or incorrect.

Summary table

Prediction Probability	Log Calculation	Final Loss
1.0 (Perfect)	$\log(1) = 0$	**0.0000**
0.7 (Good)	$\log(0.7) \approx -0.3567$	**0.3567**
0.2 (Bad)	$\log(0.2) \approx -1.6094$	**1.6094**
0.1 (Very Bad)	$\log(0.1) \approx -2.3026$	**2.3026**

This table shows that higher certainty in correct predictions results in lower loss, while lower certainty leads to higher loss.

Conclusion

Instruction masking enhances transformer-based model fine-tuning by ensuring loss computation focuses only on response tokens, improving accuracy and contextual relevance. This technique prevents models from overfitting to instruction patterns, optimizes computational efficiency, and enhances user interactions in AI-driven applications such as chatbots and Q&A systems. As AI evolves, instruction masking will remain crucial in refining models for real-world usability.

Key takeaways

- Focused learning – Loss computation only applies to response tokens, improving model accuracy and efficiency.
- Better generalization – Prevents overfitting to instruction patterns, making models more adaptable.
- Optimized computation – Reduces training overhead while enhancing response quality.
- Context awareness – Maintains contextual understanding while refining response generation.
- Real-world impact – Improves AI applications like chatbots and Q&A systems.

Author(s)

- Sowmyaa Gurusamy

Other Contributor(s)

- Malika Singla
- Lakshmi Holla