Fundamental Determinism of Neural Networks and Security

Cherkas Ruslan

2025-12-23

Contents

Notes

1. 10.5281/zenodo.18038515 `https://doi.org/10.5281/zenodo.18038515`
2. Original text [en] `https://github.com/johnthesmith/scraps/blob/main/en/nn-determinism.md`
3. The article is published under the license: `https://creativecommons.org/licenses/by-sa/4.0/` CC BY-SA 4.0

Abstract

The article proposes viewing AI safety as an engineering process, emphasizing that neural networks should be deterministic under fixed inputs, weights, and execution conditions. Any observed variability is considered a defect in hardware, software, or algorithms, highlighting the need for strict control and reproducibility in AI systems.

1 Introduction

By the nature of my work, I often encounter claims about the **fundamental non-determinism** of neural networks given identical input data.
It can be stated that for any neural network of the form

$$r = f(a, w)$$

of any type (LLM, convolutional, etc.), with fixed arguments a, fixed weights w, and identical execution conditions, the result r is deterministic. Any variation in the result without changes to a, w, or execution conditions is an implementation error.
Separately, recurrent networks of the form

$$r_1 = f(a, w, r_0)$$

must be noted, where the result r_1 explicitly depends on the previous state r_0. However, even in this case, all else being equal (including r_0), the result r_1 must be deterministic.
This assertion also applies to the training process

$$w = l(a, p)$$

where w are the resulting weights, a the training arguments, and p the training data.
Let us examine the common objections typically cited as causes of non-determinism.

2 Non-determinism of Input Data

Objection: the network is fundamentally non-deterministic due to random values during initialization.
Refutation: all numerical values, including weights, are fixed; there is no internal randomness without changing the arguments. Any violation of identical arguments for repeated computations is an implementation error.

2.1 Hardware Non-determinism

Objection: results may vary due to hardware non-determinism.
Refutation: with identical hardware and precision, hardware non-determinism is excluded. The presence of non-determinism indicates hardware faults and invalidates the result.

2.2 Software Non-determinism

Objection: results may vary due to software implementation details, including library versions, internal caching, or optimizations.
Refutation: with an identical software environment and library versions, the result must be deterministic; otherwise, it must be classified as a software defect.

2.3 Algorithmic Non-determinism

Objection: results may vary due to algorithmic non-determinism, including parallel execution, race conditions, or operation ordering.
Refutation: non-determinism without changing inputs equals a bug. An algorithm must produce a single result for fixed data and a fixed sequence of steps. If the order "floats," producing for

$$x + y + z$$

the result

$$(x + y) + z \neq x + (y + z),$$

this must be considered an incorrect implementation.

2.4 External Factors

Objection: a network may produce different results due to fuzzy operations (quantum factors) or external world parameters (measurements).
Refutation: any external-world parameter, including quantum effects or other factors, can affect the result only via explicit model arguments. If a factor directly influences the result without being explicitly included in the arguments, this is an implementation error.

3 Actions

I will outline the actions that should be taken when non-determinism is detected in a model.

3.1 Acknowledging the Problem

The presence of non-determinism is an indicator of a problem. The problem must be acknowledged and explicitly stated.

3.2 Problem Analysis

When non-determinism is detected, its source must be identified and classified into one of the categories discussed above:

1. Arguments — randomness in inputs or weights, untracked state changes, etc.

2. Hardware non-determinism — differing operation order, hardware faults.

3. Software non-determinism — library versions, optimizations.

4. Algorithmic non-determinism — implementation errors, parallelism, violation of execution order.

3.3 Problem Resolution

Based on the analysis, measures must be taken to eliminate the problem, with a demonstration of deterministic results.

3.4 Intentional Introduction of Non-determinism

I allow cases where "randomness" is required. However, it must be explicitly included in the model arguments, with the ability to repeatedly test identical arguments. The model must still demonstrate determinism.

4 Conclusions

Based on the above, I propose the following conclusions:

1. Claims about the **fundamental** non-determinism of mathematical neural networks indicate either a lack of understanding of what is happening or an attempt to conceal errors at the hardware, software, or algorithmic levels.
2. The inability of a neural network to demonstrate deterministic results should be considered a dangerous problem due to unpredictability, especially in business-critical and mission-critical systems.
3. The use of any models in business-critical and mission-critical systems must be based on an understanding of the sources of non-determinism, their control, and their elimination.
4. Implementing a deterministic execution pipeline for neural networks should be a conscious, high-priority goal.

5 Final Statement

In summary: discussions about the safety of neural network development, and AI in general, must be grounded in achieving guaranteed repeatability and verifiability of AI results. This is achievable.

Remark. Non-determinism in the execution pipeline should be treated as an implementation defect rather than a feature.

References

[1] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D., *Deep Reinforcement Learning That Matters*, Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, no. 1, pp. 3247–3254, 2018. URL: `https://ojs.aaai.org/index.php/AAAI/article/view/11694`

[2] Pineau, J., Sinha, K., Fried, G., Ke, R. N., Larochelle, H., *Improving Reproducibility in Machine Learning Research*, Journal of Machine Learning Research, vol. 22, no. 164, pp. 1–20, 2021. URL: `https://jmlr.org/papers/v22/20-303.html`