TextGrad: Automatic "Differentiation" via Text

August 6, 2024

TextGrad: Automatic "Differentiation" via Text

Mert Yuksekgonul et al., Stanford

https://arxiv.org/abs/2406.07496

arXiv:2406.07496v1 [cs.CL] 11 Jun 2024

TextGrad: Automatic "Differentiation" via Text

Mert Yuksekgonul[1*] MERTY@STANFORD.EDU
Federico Bianchi[1*] FEDE@STANFORD.EDU
Joseph Boen[2*] TBOEN@STANFORD.EDU
Sheng Liu[2*] SHENGL@STANFORD.EDU
Zhi Huang[2*] ZHIHUANG@STANFORD.EDU
Carlos Guestrin[1,3] GUESTRIN@STANFORD.EDU
James Zou[1,2,3] JAMESZ@STANFORD.EDU
[1]DEPARTMENT OF COMPUTER SCIENCE, STANFORD UNIVERSITY
[2]DEPARTMENT OF BIOMEDICAL DATA SCIENCE, STANFORD UNIVERSITY
[3]CHAN ZUCKERBERG BIOHUB
CORRESPONDENCE: MERTY@STANFORD.EDU AND JAMESZ@STANFORD.EDU

REPOSITORY AND TUTORIALS

Abstract

AI is undergoing a paradigm shift, with breakthroughs achieved by systems orchestrating multiple large language models (LLMs) and other complex components. As a result, developing principled and automated optimization methods for compound AI systems is one of the most important new challenges. Neural networks faced a similar challenge in its early days until backpropagation and automatic differentiation transformed the field by making optimization turn-key. Inspired by this, we introduce TEXTGRAD, a powerful framework performing automatic "differentiation" via text. TEXTGRAD backpropagates textual feedback provided by LLMs to improve individual components of a compound AI system. In our framework, LLMs provide rich, general, natural language suggestions to optimize variables in computation graphs, ranging from code snippets to molecular structures. TEXTGRAD follows PyTorch's syntax and abstraction and is flexible and easy-to-use. It works out-of-the-box for a variety of tasks, where the users only provide the objective function without tuning components or prompts of the framework. We showcase TEXTGRAD's effectiveness and generality across a diverse range of applications, from question answering and molecule optimization to radiotherapy treatment planning. Without modifying the framework, TEXTGRAD improves the zero-shot accuracy of GPT-4o in Google-Proof Question Answering from 51% to 55%, yields 20% relative performance gain in optimizing LeetCode-Hard coding problem solutions, improves prompts for reasoning, designs new druglike small molecules with desirable *in silico* binding, and designs radiation oncology treatment plans with high specificity. TEXTGRAD lays a foundation to accelerate the development of the next-generation of AI systems.

1 Introduction

There is an emerging paradigm shift in how AI systems are built, owing to the breakthroughs of Large Language Models (LLMs) [1–6]. The new generation of AI applications are increasingly compound systems involving multiple sophisticated components, where each component could be an LLM-based agent, a tool such as a simulator, or web search. For instance, a system of LLMs communicating with symbolic solvers can solve olympiad-level math problems [7]; a system of LLMs using search engines and code interpreter tools performs comparably to human competitive programmers [8] and are solving real-world

*Co-first authors.

TextGrad overview

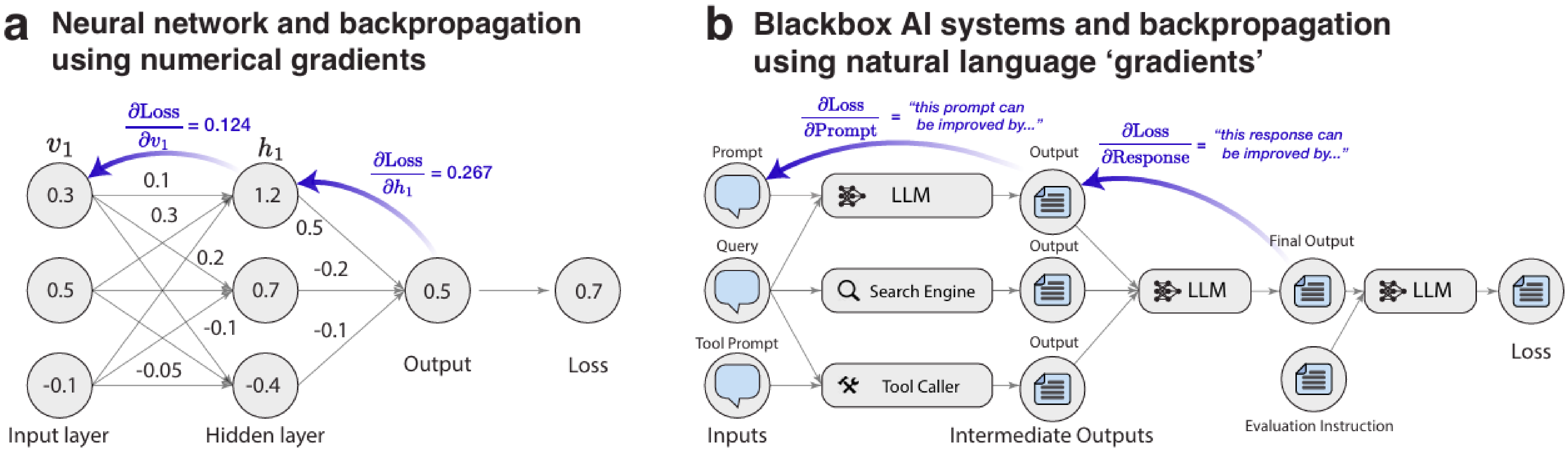
- The authors develop a technique where an LLM is used to generate a gradient-like signal for a model or system of models
  - The gradient comes from asking the LLM to provide a critique in the form of a specific recommended change
- They apply this technique to several use cases:
  - LLM generating code
  - LLM for science question problem solving
  - LLM performing reasoning
  - System designing molecules for drug targets
  - System recommending radiation treatments for prostate cancer
- Code uses syntax almost identical to PyTorch backpropagation

TextGrad gradients

- The key idea is to mimic how gradients are used to backpropagate in regular neural networks
  - When an input goes through $f_\theta(x)$, then $\partial L/\partial\theta$ is mathematically well defined
- For the gradient, they prompt an LLM with a long string containing
  - The input
  - The output
  - A prompt similar to, “The output can be improved by…”
- The equivalent of a gradient descent update step looks like
  - Given the input, output, and this criticism {text gradient}, incorporate the criticism(s) and produce a better {variable}

Backpropagation analogy

- Numerical backpropagation in figure a on the left
- Text gradients in figure b on the right



What’s needed

- The TextGrad system can work on as little as one generator LLM and one critic LLM to create text gradients
- It can also work on any system of processes with an arbitrary DAG dataflow that is the computation graph
  - An intermediate variable can have multiple predecessors and/or multiple successors. The equivalent of summing inputs/gradients is string concatenation
- The critic LLM must have enough world knowledge to make useful feedback about the processes
- The initial process(es) in the DAG probably have to be LLMs, because they need to be able to understand the criticism and do an update step
- Subsequent processes can be anything

Prompt and instance optimization

- Leveraging the gradient analogy, they do more than find good prompts
- Prompt optimization is analogous to numeric optimization where you backpropagate the loss with respect to the an input (the prompt)
  - This setting usually involves averaging the gradients of batches of sample data
  - Gradient examples include prompt tuning and p-tuning an LLM
- Instance optimization is analogous to numeric optimization where you backpropagate the loss with respect to the input/weights for a specific example
  - This setting involves only one data example at a time
  - Gradient examples include NeRF and AlphaFold

TextGrad conclusion

- Previous work has laid the foundation for using a second LLM to provide a gradient-like feedback mechanism for improving an LLM prompt
- TextGrad leans heavily on the gradient descent analogy to create a process that is clean and efficient
  - Textual Gradient Descent (TGD) iterates over multiple steps, like SGD
  - TGD works for complex systems of processes, and processes other than the initial one(s) are not limited to being LLMs
  - TGD can be used for prompt or instance optimization
- They demonstrate good results with applications involving coding, problem solving, reasoning, chemistry, and medicine

References

- Code: https://github.com/zou-group/textgrad

- Automatic Prompt Optimization with “Gradient Descent” and Beam Search
  Reid Pryzant et al. (2023)
  https://arxiv.org/abs/2305.03495
- DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines
  Omar Khattab et al. (2023)
  https://arxiv.org/abs/2310.03714