TextGrad: Automatic ”Differentiation” via Text

Mert Yuksekgonul Federico Bianchi Joseph Boen Sheng Liu Zhi Huang
Carlos Guestrin James Zou

Stanford University

Presenter: Shukai Gong

July 12, 2024

Outline

Outline

Motivation

- The new generation of AI applications are **compound systems involving multiple sophisticated components**, where each component could be an LLM-based agent, a tool such as a simulator, or web search.
- We need to develop an principled way to **optimize these compound systems**.

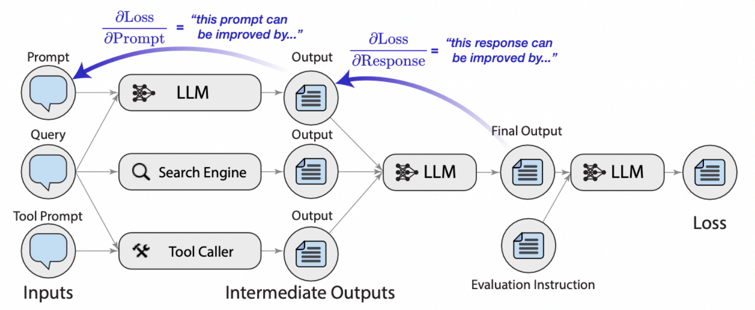

Figure 1: An exemplary AI system.

Motivation

TextGrad in a sentence:

- Automatic differentiation via text. Here differentiation/gradients are metaphors for **textual feedback** from LLMs.

AI systems under the framework of TextGrad:

- Each AI system is a `computation graph`.
- $\underbrace{\texttt{Variables}}_{\text{Prompt, Molecules, etc.}}$ are nodes in the `computation graph`, serving as inputs and outputs of $\underbrace{\text{complex function calls}}_{\text{LLM, Search Engine, etc.}}$.
- Gradients are natural language critcism to the `variables`, describing how a variable should be changed to improve the system.

Assumptions:

- The current state-of-the-art LLMs are able to reason about individual components and subtasks of the system that it tries to optimize.

Outline

Framework of TextGrad: System with 2 LLM calls

Consider a simple AI system with 2 LLM calls

$$\text{Prediction} = \text{LLM}(\text{Prompt} + \text{Question})$$
$$\text{Evaluation} = \text{LLM}(\text{Evaluation Instruction} + \text{Prediction})$$

Here the free parameter to optimize is the Prompt, $+$ denotes concatenation, and $\text{LLM}(x)$ means the response of LLM given input x.

Framework of TextGrad: Gradients

Essence of Gradients: the direction to adjust each parameter to improve a model.
Gradient for the simple graph: Prompt $\xrightarrow{\text{LLM}}$ Prediction $\xrightarrow{\text{LLM}}$ Evaluation

$$\frac{\partial \text{Evaluation}}{\partial \text{Prediction}} = \nabla_{\text{LLM}}(\text{Prediction}, \text{Evaluation})$$

$$\frac{\partial \text{Evaluation}}{\partial \text{Prompt}} = \frac{\partial \text{Evaluation}}{\partial \text{Prediction}} \cdot \frac{\partial \text{Prediction}}{\partial \text{Prompt}} = \nabla_{\text{LLM}}\left(\text{Prompt}, \text{Prediction}, \frac{\partial \text{Evaluation}}{\partial \text{Prediction}}\right)$$

Pseudo chain rule:

1. Collect the feedback on the prediction variable given the evaluation.
2. Collect the feedback on the prompt given this feedback and the [Prompt $\xrightarrow{\text{LLM}}$ Prediction] call.

Framework of TextGrad: Gradients

Gradient for generalized variable: $x \xrightarrow{\text{LLM}} y \xrightarrow{\text{LLM}} \mathcal{L}$

$$\frac{\partial \mathcal{L}}{\partial x} = \nabla_{\text{LLM}}(x, y, \frac{\partial \mathcal{L}}{\partial y}) \triangleq \text{LLM}(\text{Here is a conversation with an LLM: } \{x|y\}.$$

$$\text{Below are the critcism on } \{y\}\text{: } \{\frac{\partial L}{\partial y}\}$$

$$\text{Explain how to improve } \{x\})$$

Textual Gradient Descent:

$$x_{\text{new}} = \text{TGD.step}(x, \frac{\partial \mathcal{L}}{\partial x}) \triangleq \text{LLM}(\text{Below are the critcism on } \{x\} : \{\frac{\partial L}{\partial x}\}$$

$$\text{Incorporate the critcism and produce a new variable})$$

Two types of optimization goal:

- **Instance Optimization:** x is a solution to a problem. (Code Snipet, Molecule, etc.)
- **Prompt Optimization:** x is a prompt to an LLM.

Framework of TextGrad: General Case

For arbitarily complex systems, first we define a computation graph by

$$v = f_v(\mathsf{PredecessorsOf}(v)), \forall v \in \mathcal{V}$$

- v is a variable in the graph, $\mathcal{V}$ is the set of all variables in the graph.
- `SuccessorsOf` returns the SuccessorsOf
- `PredecessorsOf` returns the predecessors of a variable.
- f_v is a transformation that consumes a set of variables and produces the variable v, for example, LLM.

Gradient of variable (node) v:

$$\frac{\partial \mathcal{L}}{\partial v} = \bigcup_{\mathsf{SuccessorsOf}(v)} \nabla_{\mathsf{f}}(v, w, \frac{\partial \mathcal{L}}{\partial w})$$

Framework of TextGrad: General Case

Backpropagation:

Algorithm 1 Backpropagation in TEXTGRAD

Input: Variables $v \in \mathcal{V}$ in a graph, Loss variable $\mathcal{L}$, Backward Engine (LLM) $\mathcal{M}$ that will provide textual gradients

`# Initializing gradients`

for each $v \in \mathcal{V}$ **do**

  v.gradients $= \emptyset$

end for

`# Topological Sorting`

$Q \leftarrow \text{TopologicalSort}(G)$

`# Backpropagation`

for v in Q **do**

  `# Populate gradients in predecessors`

  for each $u \in \text{PredecessorsOf}(v)$ **do**

    # Here, we are omitting subscript v in f. Semantically, f is the function that generates v, and ∇_f is the backward operation for that function.

    # Semantically, this provides feedback to the variable u, given how v is produced, and the feedback we already collected for v.

    $u.\text{gradients.add}\left(\nabla_f\left(u, v, \frac{\partial \mathcal{L}}{\partial v}\right)\right)$

  end for

end for

Analogy between numerical gradients and TextGrad

Except that TextGrad shares the same syntax as Pytorch's autograd, it also has analogous optimization techniques:

- Batch Optimization: Use `tg.sum` to **concatenate** $\frac{\partial \mathcal{L}_1}{\partial x}, \cdots, \frac{\partial \mathcal{L}_n}{\partial x}$ in a batch.
- Constrained Optimization: Natural Language Prompt.
- Momentum: See earlier iterations of the variable when making the update.

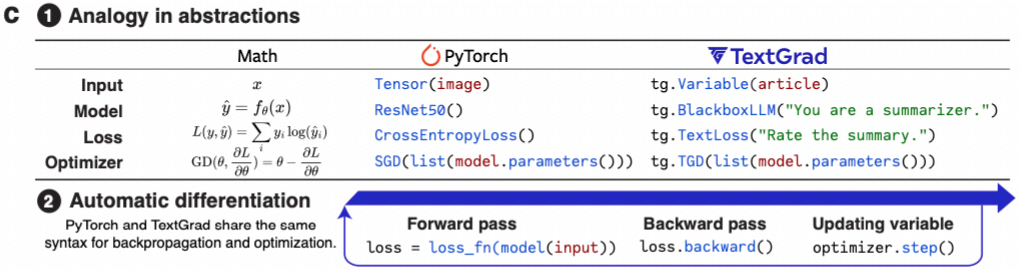

Figure 3: Analogy between numerical gradients and TextGrad.

Outline

Applications: Overview

- **Coding:** Optimizing solutions to difficult coding problems from LeetCode.
  - Boosted the performance of gpt-4o and best existing method by 20% relevant performance gain.
- **Problem Solving:**Optimizing solutions to complex scientific questions to improve the zero-shot performance of GPT-4o.
  - Improved the zero-shot accuracy from 51% to 55% by refining the solutions at test-time in Google-Proof Question Answering benchmark
- **Reasoning:** Optimize prompts to improve the LLM performance.
  - Push the performance of GPT-3.5 close to GPT-4 in several reasoning tasks.
- **Chemistry:** Designing new small molecules with desirable druglikeness and in silico binding affinity to drug targets.
- **Medicine:** Optimize radiation treatment plans for prostate cancer patients to achieve desirable target dosage and reduce side effects.

Molecule Optimization

Preliminaries:

- **Binding affinity:** The strength of the interaction between a molecule and a target protein. We use Vina score for evaluation. The more negative the better.
- **Druglikeness:** Estimation of how the molecule will behave in vivo, with respect to solubility, permeability, metabolic stability and transporter effects. We use Quantative Estimate of Druglikeness (QED, $\in [0, 1]$) score for evaluation. The higher the better.
- **SMILES:** A string encoding of a molecule.

The competing tradeoffs between these two metrics makes the optimization task realistic and challenging.

- Vina scores tend to prefer larger molecules with many functional groups that maximize interactions with a binding site.
- QED scores tend to prefer lighter, simpler molecules that are more likely to be absorbed.

Molecule Optimization

- Category: Instance Optimization
- Loss/Evaluation Function: use `gpt-4o` as the LLM,

$$\mathcal{L} = \text{LLM}(\text{Affinity}(\text{SMILES}_i, \text{Target}), \text{Druglikeness}(\text{SMILES}_i))$$

- Optimization: $\text{SMILES}_{i+1} = \text{TGD.step}(\text{SMILES}_i, \frac{\partial \mathcal{L}}{\partial \text{SMILES}_i})$

Experimental Setup:

- Apply TextGrad to all 58 targets in the DOCKSTRING molecule evaluation benchmark.
  - Consist of clinically relevant proteins sampled from a variety of structural classes, 29 of which have clinically approved drugs.
- For each target, optimize a starting fragment using TextGrad for 10 iterations, for 3 unique initial fragments.
- Compare the characteristics of the molecules generated by TextGrad to clinically approved drugs for the respective protein to evaluate the performance.

Molecule Optimization

This prompt allows us to specify both the target name (`protein_name`) as well as a prioritization (`vina_qed_ratio`) between these two objectives.

- `vina_qed_ratio`$= 10$

Molecule Optimization Prompt

Given a docking and a druglikeness score, and a molecule as a SMILES string provide a short criticism to improve the druglikeness of this molecule and its binding affinity with the protein {`protein_name`}. For docking, lower is better (less than -10 is considered good) and for druglikeness, 1 is the best and 0 is the worst (greater than 0.8 is considered good). In terms of prioritization, the docking score is {`vina_qed_ratio`} times as important as the druglikeness score. Make sure your criticism is very succinct and to the point.

`# if smiles_string is valid`
SMILES: {`smiles_string`}, Docking: {`Vina`}, Druglikeness: {`QED`}

`# if smiles_string is invalid`
SMILES: {`smiles_string`}, This molecule is invalid.

Figure 4: Molecule Optimization Prompt

Post-hoc Selection:

$$s_{\text{overall}}(\text{molecule}, \text{protein}) = \text{Vina}(\text{molecule}, \text{protein}) + (1 - \text{QED}(\text{molecule}))$$

Molecule Optimization

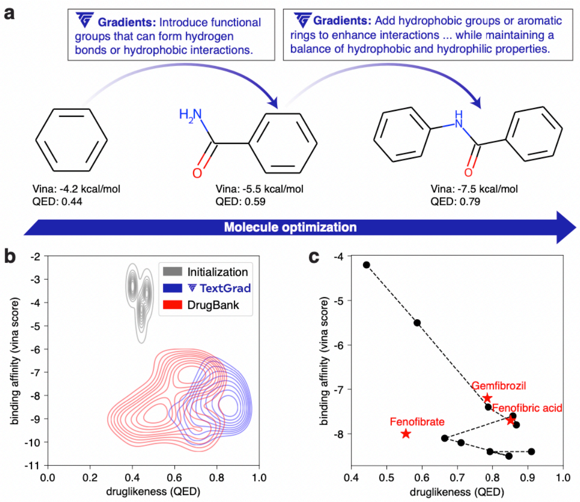

Figure 5: Molecule Optimization Results

Molecule Optimization

Drug Discovery by TextGrad: The molecule at the final iteration in Figure 5 (c) has low structural similarity with its most similar clinically approved counterpart, and **better QED and Vina scores** (d) with a highly **plausible pose geometry** shown in (e).

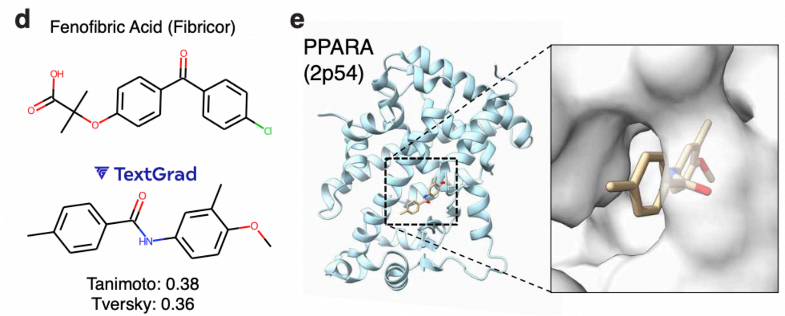

Figure 6: Drug Discovery

Molecule Optimization: Sanity Checks

LLM hallucinations could manifest by TextGrad proposing invalid, toxic, or otherwise undesirable molecules in order to optimize its objective function.

Evaluation: ADMET-AI model is used to detect mutagenisis and clinical toxicity. 1.0 indicates a highly likelihood for harm and 0.0 a low likelihood.

- No harmfulness indicators are directly encoded into TextGrad objective function, but it implicitly avoids proposing harmful molecules!

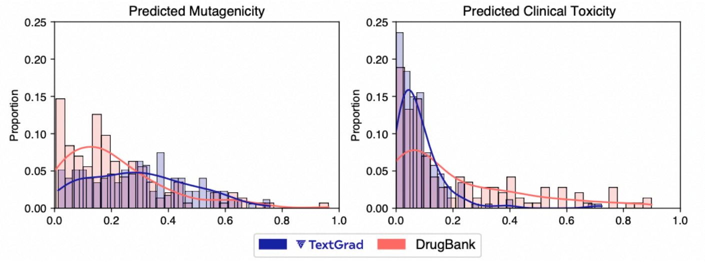

Figure 7: Safety Properties of Molecules generated by TextGrad

Prompt optimization for reasoning

- Category: Prompt Optimization
- Answer $=$ `gpt-3.5-turbo-0125`(Prompt, Question). Loss function:

$$\mathcal{L} = \text{Evaluator}(\text{Answer}, \text{Ground Truth})$$

- Optimization: $\text{Prompt}_{i+1} = \text{TGD.step}(\text{Prompt}_i, \frac{\partial \mathcal{L}}{\partial \text{Prompt}_i})$ (`gpt-4o`)
- Tasks: 2 standard reasoning tasks (Object Counting and Word Sorting) from Big Bench Hard, and GSM8k grade-school math problem solving.
- Evaluator:
  - For object counting: string-based exact match metric.
  - For word sorting: LLM.

Prompt optimization for reasoning

- Baselines:
  - Zero-shot CoT
  - DSPy: a state-of-the-art language model programming and prompt optimization framework. (Few-shots)

Table 3: Prompt optimization for reasoning tasks. With TEXTGRAD, we optimize a system prompt for `gpt-3.5-turbo` using `gpt-4o` as the gradient engine that provides the feedback during backpropagation.

Dataset	Method	Accuracy (%)
Object Counting [50, 51]	CoT (0-shot) [46, 47]	77.8
	DSPy (BFSR, 8 demonstrations) [10]	84.9
	TEXTGRAD (instruction-only, 0 demonstrations)	**91.9**
Word Sorting [50, 51]	CoT (0-shot) [46, 47]	76.7
	DSPy (BFSR, 8 demonstrations) [10]	**79.8**
	TEXTGRAD (instruction-only, 0 demonstrations)	**79.8**
GSM8k [52]	CoT (0-shot) [46, 47]	72.9
	DSPy (BFSR, 8 demonstrations) [10]	**81.1**
	TEXTGRAD (instruction-only, 0 demonstrations)	**81.1**

Figure 8: Prompt Optimization for Reasoning

Prompt optimization for reasoning

Results:

- Across all three tasks, TextGrad improves the performance of the 0-shot prompt significantly.
- It performs similarly to DSPy for Word Sorting and GSM8k, and improves over DSPy by 7% for Object Counting.

(DSPy adds in-context demonstration examples and TextGrad optimizes the system prompt)

- For GSM8k, directly **combining** the demonstrations from DSPy with the instruction from TextGrad increases the accuracy to 82.1%

Outline

Related Work and Similar Insights

- **DSPy:** Viewing complex LLM-based systems as programs with potentially many layers, and proposes ways to build and optimize them in a programmatic fashion.
- **Prompt Optimization with Textual Gradients (ProTeGi):** Defining the Textual Gradients in the context of prompt optimization, where gradients are natural language feedback from LLMs given to the mistakes made during the task.
- **Verbalized Machine Learning: Revisiting Machine Learning with Language Models: (arXiv:2406.04344v1):** Adopt LLM-parameterized learner (*gradient*) and optimizer (*textual gradient descent*).

Related Work and Similar Insights

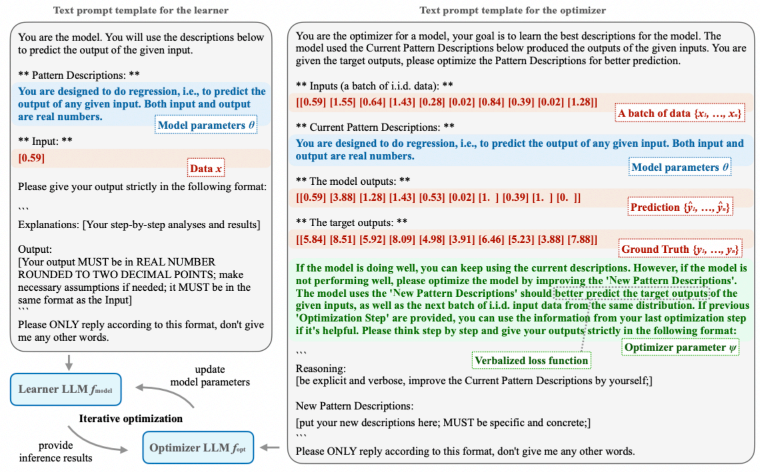

Figure 2: An overview of iterative optimization and text prompt templates of the learner and the optimizer in the regression example.

Figure 9: Verbalized Machine Learning (arXiv:2406.04344v1)

Related Work and Similar Insights

$$\nabla_{\boldsymbol{\theta}} \ell_{\text{regression}} = \frac{1}{N} \sum_{i=1}^{N} \left(y_n - f_{\text{model}}(\boldsymbol{x}_n; \boldsymbol{\theta}) \right) \cdot \frac{\partial f_{\text{model}}(\boldsymbol{x}_n; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \quad \text{s.t. } \boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}} \ell_{\text{regression}} \in \Theta_{\text{language}} \tag{2}$$

Figure 10: Update of parameters

Algorithm 1 Training in VML

Initialize model parameters $\boldsymbol{\theta}_0$, iteration number T, batch size M and optimizer parameters $\boldsymbol{\psi}$;
for $i = 1, \cdots, T$ **do**
  Sample M training examples $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_M$;
  for $m = 1, 2, \cdots, M$ **do**
    $\hat{y}_m = f_{\text{model}}(\boldsymbol{x}_m; \boldsymbol{\theta}_{i-1})$;
  end
  $\boldsymbol{\theta}_i = f_{\text{opt}}\left(\{\boldsymbol{x}_m, \hat{y}_m, y_m\}_{m=1}^{M}, \boldsymbol{\theta}_{i-1}; \boldsymbol{\psi}\right)$;
end

Figure 11: Algorithm of Gradient Descent

Discuss

Suspicion about text-based gradient

- Computations of gradients in TextGrad are completely done by LLMs. Hallucinations of LLMs may severely affect the correctness/direction of the gradients.
- No guarantee of convergence of the optimization process especially when the conversation windows is long.
- Higher computational cost and volatility for traditional tasks like regression / classification / $\cdots$.

Thank you!