# 【prompt 自动优化】TextGrad：借鉴DSPy 并融合梯度下降的新方法

原创 方方 方方的算法花园 2024年10月31日 09:02 北京

点击蓝字 关注我们

## 写在前面

前面，我们介绍了《【prompt 自动优化】DSPy：原理与实践全解析》，本文我们再来看一下TextGrad。

**TextGrad同是斯坦福发布（2024年6月），借鉴了 DSPy，融合了 PyTorch 的强大梯度反向传播功能，实现自动优化复杂 AI 系统。能够通过文本，执行自动“微分”，对大模型提供的文本反馈进行反向传播，以改进复合AI系统的各个组件。**

## TextGrad 核心思想

**TextGrad 把 LLM 的应用视为一个计算图，在这里，自然语言充当了实现不同组件间“梯度”传递的媒介。**它借助从语言模型的输出向所有可能的早期组件进行反向传播文本反馈的方式，来对各种系统内的各类变量进行优化。

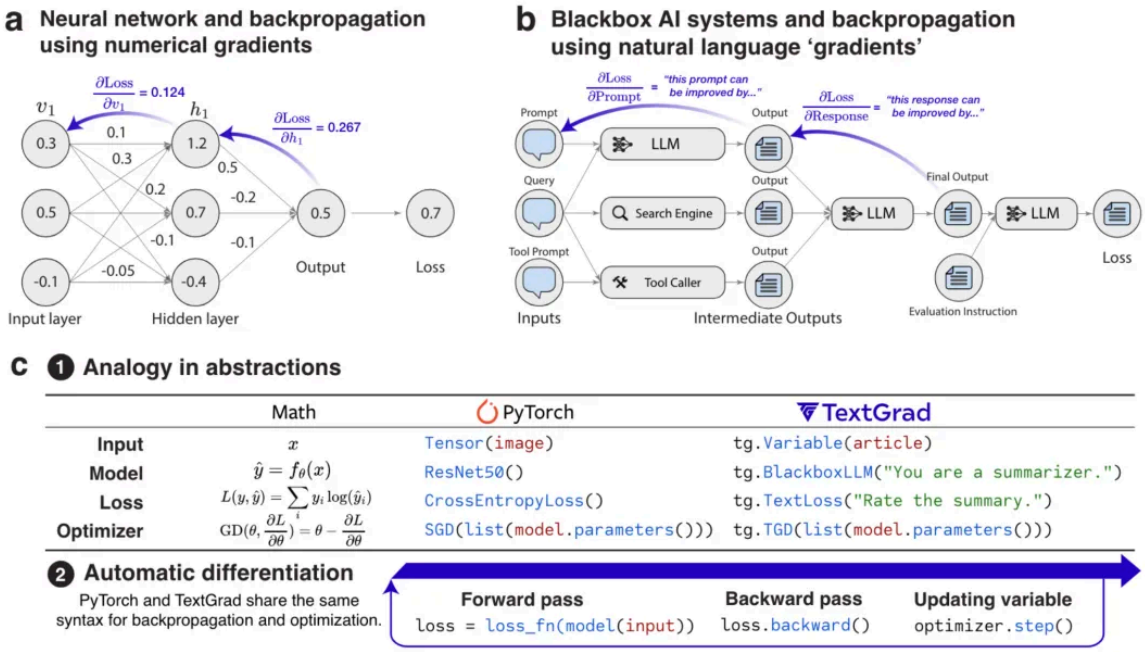
在 TextGrad 中，一切皆以文本呈现，这也就意味着我们利用语言模型来完成以下三个方面的任务：

**1）对输出进行评估；**

**2）对输出进行批评；**

**3）对输入进行更新。**

这种过程类似于 PyTorch 的反向传播，只不过此时传播的并非数值梯度，而是以文本形式呈现的反馈。




以两个 LLM 调用组成的系统为例，绿色代表需要优化的参数prompt，使用+表示两个字符串的串联，使用LLM(x)表示，将x作为提示提供给语言模型以收集响应，使用如下链式表示

法：

$$\text{Prediction} = \text{LLM}(\text{Prompt} + \text{Question}), \tag{1}$$

$$\text{Evaluation} = \text{LLM}(\text{Evaluation Instruction} + \text{Prediction}), \tag{2}$$

$$\text{Prompt} + \text{Question} \xrightarrow{\text{LLM}} \text{Evaluation Instruction} + \text{Prediction} \xrightarrow{\text{LLM}} \text{Evaluation}, \tag{3}$$

当进行一次 LLM 调用，利用 Prompt 来生成针对问题的预测后，再紧接着进行另一次 LLM 调用以对该预测进行评估。

这种统一的语言交互界面为 TextGrad 赋予了极强的普适性。TextGrad 将 prompt、question、output 等统统视为变量，且并不要求它们可微，展现出了极为强大的兼容性。TextGrad 能够与任意支持自然语言输入输出的 LLM 或其他 API 实现无缝协作，同时也不要求计算图中的其他函数可微。这一特性使它极为适合与检索、工具调用等即插即用的能力相融合，进而构建出灵活且变化多样的复合 AI 管道系统。

此外，TextGrad 无需人工设计 prompt，而是通过自动搜索最优的任务描述来直接参与优化过程。这让开发者从繁琐的 prompt engineering 中解脱出来，并且有望自动寻找到更为出色的 in-context learning 模式。

## TextGrad VS DSPy

论文中进行了如下实验：

**对比任务**

Big Bench Hard 中的两个典型推理任务，即物体计数和单词排序，【示例被随机划分为 50 个（用于训练）、100 个（用于验证）和 100 个（用于测试）样本】；还有 GSM8k 中的小学数学问题解决任务【采用了 DSPy 的分割方式，具体为 200 个（用于训练）、300 个（用于验证）和 1319 个（用于测试）】。

**Example Query for Word Sorting**

Sort the following words alphabetically: List: oakland seaborg jacobi membrane trapezoidal allis marmot toggle anthology.

**Example Query for Object Counting**

I have a couch, a bed, a car, a fridge, two tables, an oven, a toaster, and a chair. How many objects do I have?

**Example Query for GSM8k**

Amber, Micah, and Ahito ran 52 miles in total. Amber ran 8 miles. Micah ran 3.5 times what Amber ran. How many miles did Ahito run?

**评估方法**

针对物体计数和 GSM8k 这两项任务，采用基于字符串的精确匹配度量方法，该方法会查看答案中所提供的最后一个数值，并将其与真实答案进行比较。而对于单词排序任务，则提示 gpt-4o 通过以下特定提示，将真实答案列表与答案中所提供的响应进行比较。

**Evaluation system prompt for Word Sorting evaluation**

**System Prompt:** Below is a question from a question-answering task, the ground truth answer, and reasoning with the final prediction. Is the final prediction correct, i.e. the same as the ground truth answer? Say only 1 (yes) or 0 (no). Return your response within <ACCURACY> </ACCURACY> tags. e.g.<ACCURACY> 0 </ACCURACY> or <ACCURACY> 1 </ACCURACY>.

**Example prompt:**
**Question for the task:** {question}
**Ground truth answer:** {answer}
**Reasoning and prediction from the language model:** {prediction}

**对比项**

- TEXTGRAD（仅指令，无示例演示）：利用 gpt-4o 在反向传播过程中提供反馈，以提升 gpt-3.5-turbo-0125 的性能；采用 3 的批量大小，并进行 12 次迭代，也就是说模型总共看到 36 个训练样本，这些样本是随机有放回地抽取的。在每次迭代后，使用数据集的验证集来运行验证循环，如果性能比上一次迭代更好，就更新提示。
- COT（零示例）：零样本的 COT 模式。
- DSPy（BFSR，8 个示例）：采用了 DSPy 的 BootstrappedFewShotRandomSearch（BFSR）优化器，配置了 10 个候选程序和 8 个少量示例。该优化器通过生成能够满足特定指标（如准确度）的语言模型输入输出跟踪，来确定哪些示例应包含在提示中。它还融合了链式推理（CoT）机制。之后，它会在这些示例的子集中进行随机搜索，每个子集最多包含 8 个示例。

**对比结果**

TEXTGRAD 明显提升了零样本提示的性能表现。它在单词排序和 GSM8k 任务上的表现与 DSPy 相当，在物体计数任务上相较于 DSPy 还提高了 7%。尽管上下文中的 8 个示例能够引导 LLM 的行为，但这可能会增加推理的成本。

**DSPy 优化器与 TEXTGRAD 做出了相互补充的调整——前者增加了上下文中的示范示例，而后者则对系统提示进行了优化。将 DSPy 所选择的示例添加到 TEXTGRAD 优化后的提示中能够进一步提升性能（对于 GSM8k，直接将 DSPy 的示例与 TEXTGRAD 的指令相结合，可将准确度提升至 82.1%），这表明将这两种方法结合起来是一个非常有益的发展方向。**

| 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** |

Example: TextGrad optimized prompt for `gpt-3.5-turbo-0125`

**Prompt at initialization (GSM8k Accuracy= 72.9%):**
*You will answer a mathematical reasoning question. Think step by step. Always conclude the last line of your response should be of the following format: 'Answer: $VALUE' where VALUE is a numerical value."*

**Prompt after 12 iterations with batch size 3 (GSM8k Accuracy= 81.1%):**
*You will answer a mathematical reasoning question. Restate the problem in your own words to ensure understanding. Break down the problem into smaller steps, explaining each calculation in detail. Verify each step and re-check your calculations for accuracy. Use proper mathematical notation and maintain consistency with the context of the question. Always conclude with the final answer in the following format: 'Answer: $VALUE' where VALUE is a numerical value.*

## TextGrad实践

### 1 环境安装

```
pip install textgrad
或
conda install -c conda-forge textgrad
或
pip install git+https://github.com/zou-group/textgrad.git
或
pip install textgrad[vllm]
```

### 2 入门

```
import textgrad as tg

tg.set_backward_engine("gpt-4o", override=True)

# Step 1: 获取LLM的初始回复
model = tg.BlackboxLLM("gpt-4o")
question_string = ("如果在阳光下晾干 25 件衬衫需要 1 小时，"
                   "在阳光下晾干 30 件衬衫需要多长时间？"
                   "一步一步地推理")

question = tg.Variable(question_string,
                       role_description="question to the LLM",
                       requires_grad=False)

answer = model(question)

# answer (初始回复错误)
"""
⚠ answer: 要确定在阳光下晾干 30 件衬衫需要多长时间，我们可以根据给定的信息使用比例关
 这是一步一步的推理：[.....]
```

```
因此，在阳光下晾干 30 件衬衫需要 1.2 小时（或 1小时12分钟）。
"""


answer.set_role_description("简洁准确回答问题")

# Step 2: 像pytorch一样定义损失函数和优化器
# 这里没有SGD，但有TGD（文本梯度下降）
optimizer = tg.TGD(parameters=[answer])
evaluation_instruction = (f"这是问题描述: {question_string}. "
                          "评估此问题的给定答案, "
                          "聪明、合乎逻辑且非常具有批判性. "
                          "只需提供简洁的反馈.")


# TextLoss 是一个自然语言指定的损失函数，它描述了我们想要如何评估推理。
loss_fn = tg.TextLoss(evaluation_instruction)

#loss print
"""
[...] 您的逐步推理清晰且合乎逻辑，但它在假设干燥时间与衬衫数量成正比方面包含一个严重缺
"""

# Step 3: 执行损失计算、反向传播
loss = loss_fn(answer)
loss.backward()
optimizer.step()
answer

#answer print
"""
假设30件衬衫都布置得当以接收相等的阳光，在阳光下晾干 30 件衬衫仍然需要 1 小时。
"""
```

### 3 示例：最小实例优化

```
tg.set_backward_engine("gpt-4o")

initial_solution = """为了解决这个方程 3x^2 - 7x + 2 = 0, 我们使用一元二次方程求根
x = (-b ± √(b^2 - 4ac)) / 2a
a = 3, b = -7, c = 2
x = (7 ± √((-7)^2 - 4 * 3(2))) / 6
x = (7 ± √(7^3) / 6
结果是:
x1 = (7 + √73)
```

```
x2 = (7 - √73)"""

# 定义要优化的变量，让 requires_grad=True 启用梯度计算
solution = tg.Variable(initial_solution,
                       requires_grad=True,
                       role_description="解决这个数学问题")

# 定义优化器，让优化器知道要优化哪些变量，然后运行损失函数
loss_fn = tg.TextLoss("你将评估数学问题的解决方案。不要试图自己解决，不要给出解决方案

optimizer = tg.TGD(parameters=[solution])
loss = loss_fn(solution)

#loss print
"""
Variable(value=Errors:
1.判别式计算中的符号不正确：应该是 b^2 - 4ac，而不是 b^2 + 4ac。
2.二次公式的简化不正确：分母应该是 2a，而不是 6。
3.最终解决方案缺少除以 2a。, role=response from the language model, grads=)
"""

loss.backward()
optimizer.step()
print(solution.value)

#answer print
"""
为了求解方程 3x^2 - 7x + 2 = 0，我们使用二次公式：x = （-b ± √（b^2 - 4ac））/ 2

给定：a = 3， b = -7， c = 2

将值代入公式：x = （7 ± √（（-7）^2 - 4（3）（2））） / 6 x = （7 ± √（49 - 24））

解决方案是：x1 = （7 + 5） / 6 = 12 / 6 = 2 x2 = （7 - 5） / 6 = 2 / 6 = 1/3
"""
```

## 4 示例：最小prompt优化

```
import textgrad as tg
llm_engine = tg.get_engine("gpt-3.5-turbo")
tg.set_backward_engine("gpt-4o")
_, val_set, _, eval_fn = load_task("BBH_object_counting", llm_engine)
question_str, answer_str = val_set[0]
question = tg.Variable(question_str, role_description="question to the LLM",
answer = tg.Variable(answer_str, role_description="answer to the question", r
```

```
"""
Question:
我有两根芹菜茎、两颗大蒜、一个土豆、三头西兰花、一根胡萝卜和一根山药。我有多少蔬菜？
Ground Truth Answer:
10
"""
system_prompt = tg.Variable("你是一个简洁的LLM，逐步思考",
                            requires_grad=True,
                            role_description="system prompt to guide the LLM"
model = tg.BlackboxLLM(llm_engine, system_prompt=system_prompt)
optimizer = tg.TGD(parameters=list(model.parameters()))
prediction = model(question)
"""
Prediction:
你总共有七种蔬菜：两根芹菜茎、两根大蒜、一个土豆、三头西兰花、一根胡萝卜和一根山药。
"""
loss = eval_fn(inputs=dict(prediction=prediction, ground_truth_answer=answer)
"""
Loss denoting accuracy:
Variable(value=0, grads=)
"""
loss.backward()
"""
System prompt gradients:
... 2. 鼓励显式求和:
- 提示应鼓励模型明确说明求和过程。这有助于验证计数的准确性。
例如，“清楚地解释您的计算并验证总计。....
"""
optimizer.step()
"""
New system prompt value:
你是一个简洁的LLM。逐步思考。在计算中优先考虑准确性。单独识别并计算每个项目。
清楚地解释您的计算并验证总数。计算后，查看您的步骤以确保总数正确。
如果您发现计数存在差异，请重新评估列表并更正任何错误。
"""
prediction = model(question)
"""
New prediction:
L新预测：
让我们数一数每种蔬菜的数量：
芹菜茎：2
大蒜：2
马铃薯：1
西兰花头：3
胡萝卜：1
山药：1
```

```
现在，让我们把蔬菜的总数相加起来：2 + 2 + 1 + 3 + 1 + 1 = 10
"""
```

## 5 自定义本地模型

参考https://github.com/zou-group/textgrad/issues/13 中的回复，可以自建openai接口后调用。

**1. 封装本地模型的openai标准接口，以便TextGrad调用。**

（1）拉取api-for-open-llm代码并安装项目依赖

```
git clone https://github.com/xusenlinzy/api-for-open-llm.git
cd api-for-open-llm
pip install -r requirements.txt
```

（2）在api-for-open-llm项目中创建.env文件，并配置以下内容：

```
vim .env

# 配置以下内容：

# 启动端口
PORT=8000
# model 命名
MODEL_NAME=Qwen2-7B-Instruct
# 将MODEL_PATH改为我们的模型所在的文件夹路径
MODEL_PATH=../models/Qwen2-7B-Instruct
# device related
# GPU设备并行化策略
DEVICE_MAP=auto
# GPU数量
NUM_GPUs=1
# 开启半精度，可以加快运行速度、减少GPU占用
DTYPE=half
# api related
# API前缀
API_PREFIX=/v1
# API_KEY，此处随意填一个字符串即可
OPENAI_API_KEY=0
```

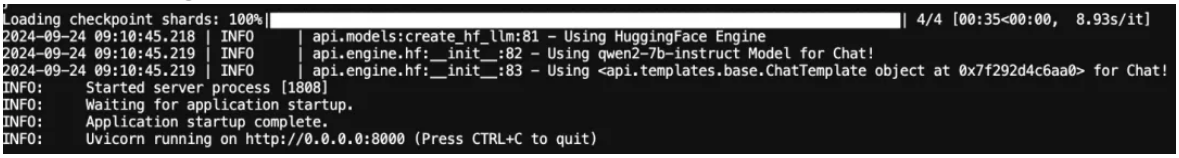
（3）测试接口

```
cp api-for-open-llm/api/server.py api-for-open-llm/
python api-for-open-llm/server.py > server.log 2>&1 &
```

cat server.log 如果出现如下报错：File "xx/api-for-open-llm/api/server.py", line 1, in
from api.config import SETTINGS

ModuleNotFoundError: No module named 'api.config'; 'api' is not a package
是由于api-for-open-llm github的api文件夹缺少__init__.py 文件，执行 touch __init__.py 创建即可

cat server.log 继续查看日志，出现以下日志代表openai接口创建成功。




### 2. 加载模型

```python
import os
from textgrad.engine.openai import ChatOpenAI
os.environ['OPENAI_API_KEY'] = "0"
os.environ['OPENAI_BASE_URL'] = "http://localhost:8000/v1"
engine = ChatOpenAI(model_string='Qwen2___5-3B-Instruct')
print(engine.generate(max_tokens=40, content="你好",system_prompt="You are a
## 输出
"""
你好！有什么问题我可以帮助你吗？
"""
```

## 参考链接

> **论文**：TextGrad: Automatic "Differentiation" via Text（https://arxiv.org/pdf/2406.07496）
> **Github**：https://github.com/zou-group/textgrad

Hello. END

#LLM学习 12 语言模型 5

#LLM学习 · 目录

上一篇
【LLM论文阅读】MoRE：LLM通过多视角反思与迭代增强序列推荐

下一篇
【prompt 自动优化】DSPy：原理与实践全解析