# Minimal Working Example

In this post, we walk you through a minimal working example using the DSPy library.

We make use of the GSM8K dataset and the OpenAI GPT-3.5-turbo model to simulate prompting tasks within DSPy.

## Setup

Before we jump into the example, let's ensure our environment is properly configured. We'll start by importing the necessary modules and configuring our language model:

```
import dspy
from dspy.datasets.gsm8k import GSM8K, gsm8k_metric

# Set up the LM.
turbo = dspy.OpenAI(model='gpt-3.5-turbo-instruct', max_tokens=250)
dspy.settings.configure(lm=turbo)

# Load math questions from the GSM8K dataset.
gsm8k = GSM8K()
gsm8k_trainset, gsm8k_devset = gsm8k.train[:10], gsm8k.dev[:10]
```



Let's take a look at what `gsm8k_trainset` and `gsm8k_devset` are:

```
print(gsm8k_trainset)
```

The `gsm8k_trainset` and `gsm8k_devset` datasets contain lists of `dspy.Examples`, with each example having `question` and `answer` fields.

# Define the Module #

With our environment set up, let's define a custom program that utilizes the `ChainOfThought` module to perform step-by-step reasoning to generate answers:

```
class CoT(dspy.Module):
    def __init__(self):
        super().__init__()
        self.prog = dspy.ChainOfThought("question -> answer")

    def forward(self, question):
        return self.prog(question=question)
```

# Compile and Evaluate the Model

With our simple program in place, let's move on to compiling it with the `BootstrapFewShot` teleprompter:



```
from dspy.teleprompt import BootstrapFewShot

# Set up the optimizer: we want to "bootstrap" (i.e., self-generate) 4-shot examples of our CoT program.
```

```
config = dict(max_bootstrapped_demos=4, max_labeled_demos=4)

# Optimize! Use the `gsm8k_metric` here. In general, the metric is going to tell the optimizer how well
it's doing.
teleprompter = BootstrapFewShot(metric=gsm8k_metric, **config)
optimized_cot = teleprompter.compile(CoT(), trainset=gsm8k_trainset)
```

Note that `BootstrapFewShot` is not an optimizing teleprompter, i.e. it simply creates and validates examples for steps of the pipeline (in this case, chain-of-thought reasoning) but does not optimize the metric. Other teleprompters like `BootstrapFewShotWithRandomSearch` and `MIPRO` will apply direct optimization.

# Evaluate

Now that we have a compiled (optimized) DSPy program, let's move to evaluating its performance on the dev dataset.

```
from dspy.evaluate import Evaluate

# Set up the evaluator, which can be used multiple times.
evaluate = Evaluate(devset=gsm8k_devset, metric=gsm8k_metric, num_threads=4, display_progress=True,
display_table=0)

# Evaluate our `optimized_cot` program.
evaluate(optimized_cot)
```



# Inspect the Model's History

For a deeper understanding of the model's interactions, we can review the most recent generations through inspecting the model's history:

```
turbo.inspect_history(n=1)
```

And there you have it! You've successfully created a working example using the DSPy library.

This example showcases how to set up your environment, define a custom module, compile a model, and rigorously evaluate its performance using the provided dataset and teleprompter configurations.

Feel free to adapt and expand upon this example to suit your specific use case while exploring the extensive capabilities of DSPy.

If you want to try what you just built, run `optimized_cot(question='Your Question Here')`.

**Written By:** Herumb Shandilya