Introducing DSPy

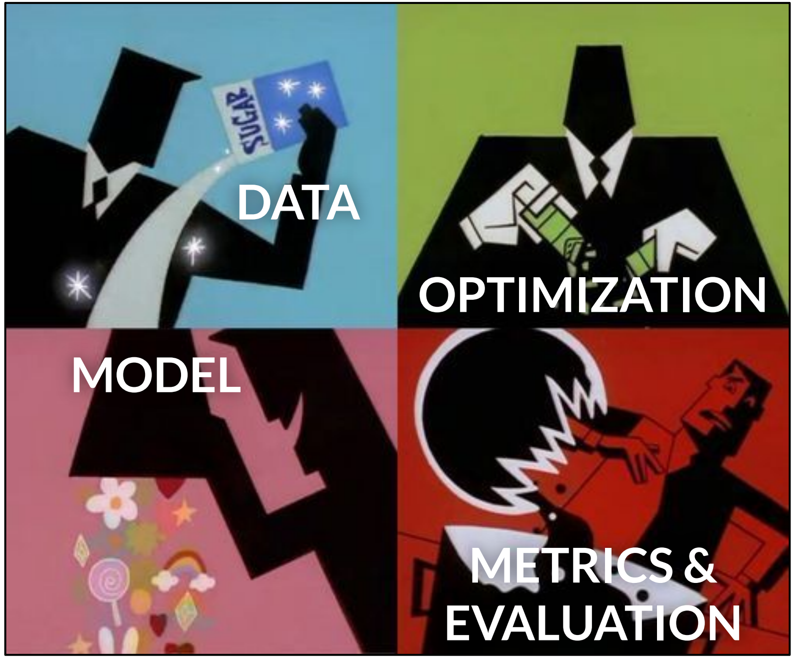
“*Programming*—not prompting—Foundation Models”

Riccardo Belluzzo

Agenda

- Introduction to DSPy
- Demo

The ML Recipe



The LLM Recipe (?)

PROMPT ENGINEER AT WORK

The LLM Recipe Expanded

1. Break the problem down into steps;
2. Prompt your LM well until each step works well in isolation;
3. Tweak the steps to work well together;
4. Get (or generate synthetic) examples to tune each step;
5. [Optional] Use these examples to finetune smaller LLMs to cut costs

Additionally, every time you change your pipeline, your LLM, or your data, all prompts (or fine-tuning steps) may need to change.

The DSPy Solution

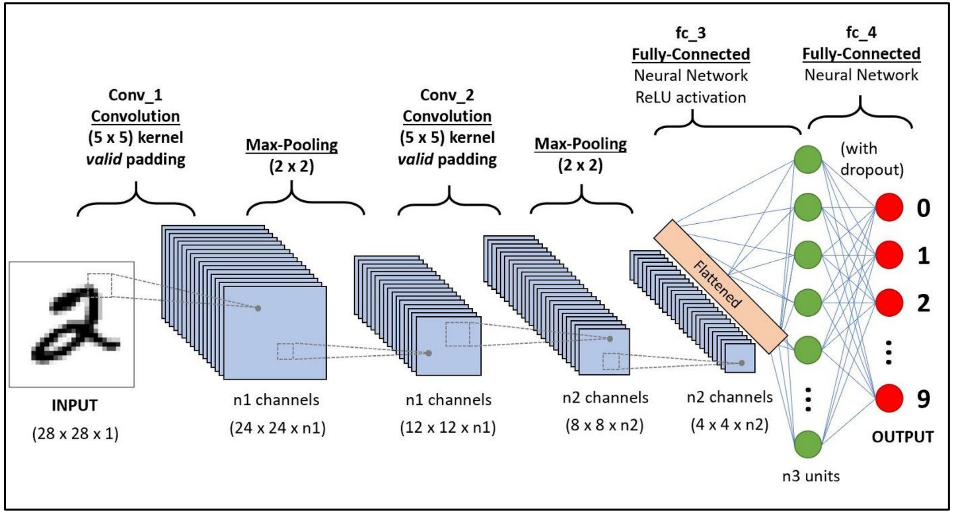
To make this more systematic and much more powerful, **DSPy** does two things:

1. It separates the flow of your program **(modules)** from the parameters (LM prompts and weights) of each step;

2. It introduces new **optimizers**, which are LM-driven algorithms that can tune the prompts and/or the weights of your LM calls, given a **metric** you want to maximize.

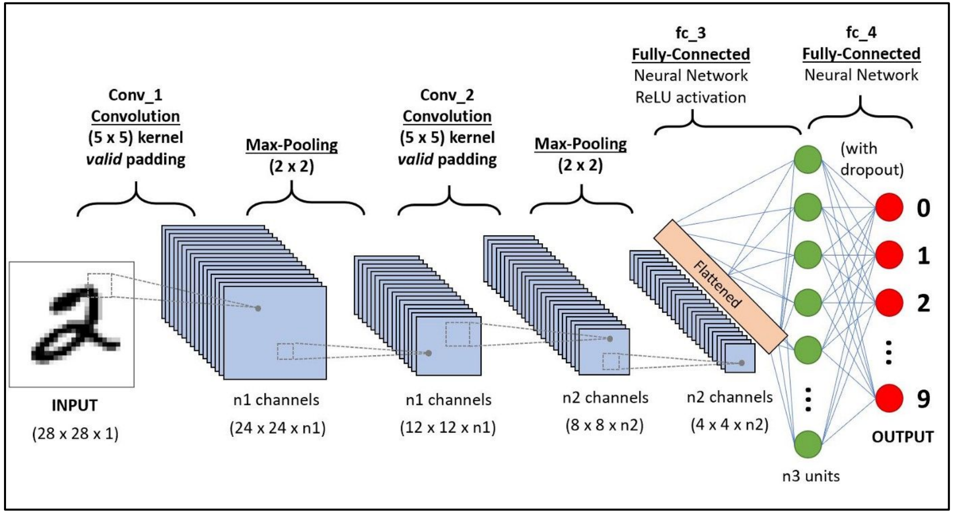
DSP: ***D****eclarative Language Model Calls into* ***S****elf-improving* ***P****ipelines*

A Deep Learning Program...



A Deep Learning Program…

… written in PyTorch

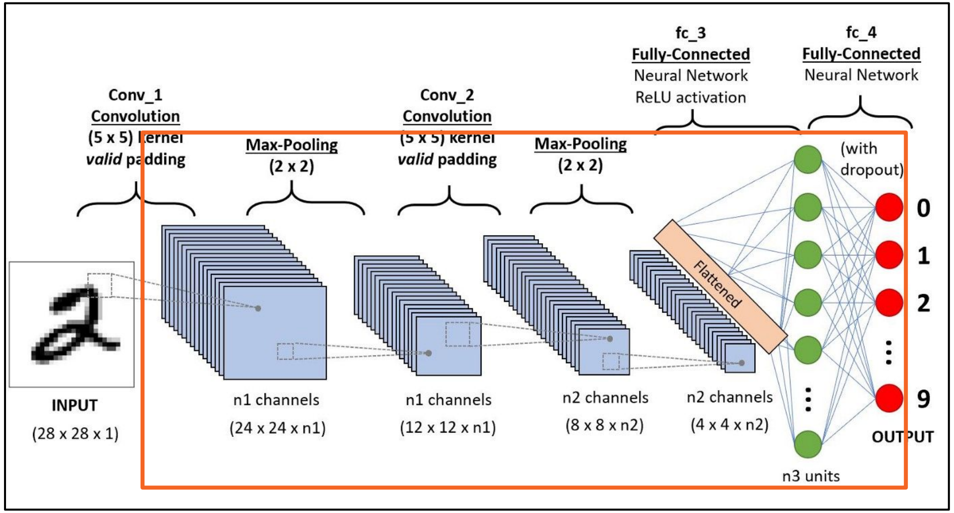


```
class ConvNet(nn.Module):
   def __init__(self):
     super().__init__()
     self.conv1 = nn.Conv2d(3, 6, 5)
     self.pool = nn.MaxPool2d(2, 2)
     self.conv2 = nn.Conv2d(6, 16, 5)
     self.fc1 = nn.Linear(16 * 5 * 5, 120)
     self.fc2 = nn.Linear(120, 84)
     self.fc3 = nn.Linear(84, 10)

   def forward(self, x) -> Tensor:
     x = self.pool(F.relu(self.conv1(x)))
     x = self.pool(F.relu(self.conv2(x)))
     x = torch.flatten(x, 1)
     x = F.relu(self.fc1(x))
     x = F.relu(self.fc2(x))
     x = self.fc3(x)
     return x
```

A Deep Learning Program...

... written in PyTorch



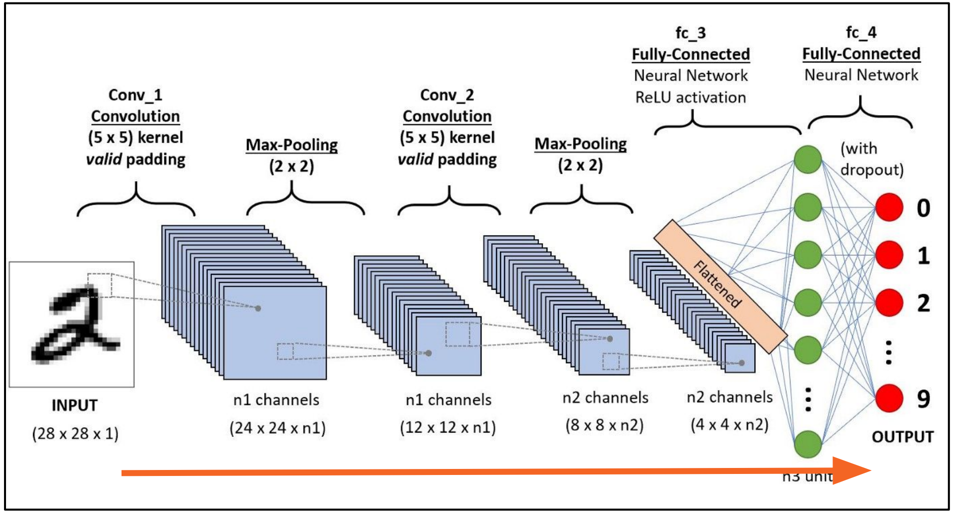
BUILDING BLOCKS

```
class ConvNet(nn.Module):
   def __init__(self):
     super().__init__()
     self.conv1 = nn.Conv2d(3, 6, 5)
     self.pool = nn.MaxPool2d(2, 2)
     self.conv2 = nn.Conv2d(6, 16, 5)
     self.fc1 = nn.Linear(16 * 5 * 5, 120)
     self.fc2 = nn.Linear(120, 84)
     self.fc3 = nn.Linear(84, 10)

   def forward(self, x) -> Tensor:
     x = self.pool(F.relu(self.conv1(x)))
     x = self.pool(F.relu(self.conv2(x)))
     x = torch.flatten(x, 1)
     x = F.relu(self.fc1(x))
     x = F.relu(self.fc2(x))
     x = self.fc3(x)
     return x
```

A Deep Learning Program...

... written in PyTorch



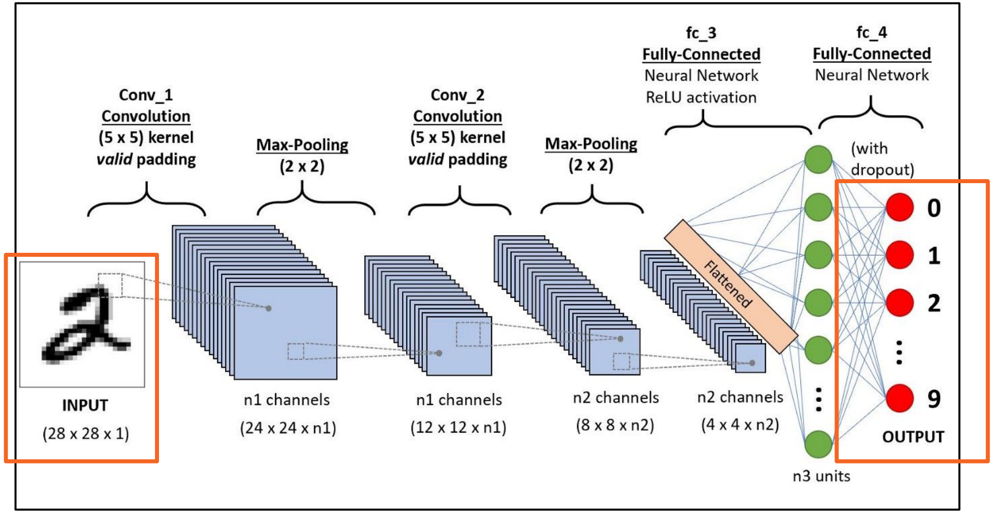
PROGRAM FLOW

```
class ConvNet(nn.Module):
  def __init__(self):
    super().__init__()
    self.conv1 = nn.Conv2d(3, 6, 5)
    self.pool = nn.MaxPool2d(2, 2)
    self.conv2 = nn.Conv2d(6, 16, 5)
    self.fc1 = nn.Linear(16 * 5 * 5, 120)
    self.fc2 = nn.Linear(120, 84)
    self.fc3 = nn.Linear(84, 10)

  def forward(self, x) -> Tensor:
    x = self.pool(F.relu(self.conv1(x)))
    x = self.pool(F.relu(self.conv2(x)))
    x = torch.flatten(x, 1)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)
    return x
```

A Deep Learning Program...

... written in PyTorch



INPUT/OUTPUT DEFINITION

(i.e we don't care too much about what is happening in the middle)

```
class ConvNet(nn.Module):
   def __init__(self):
     super().__init__()
     self.conv1 = nn.Conv2d(3, 6, 5)
     self.pool = nn.MaxPool2d(2, 2)
     self.conv2 = nn.Conv2d(6, 16, 5)
     self.fc1 = nn.Linear(16 * 5 * 5, 120)
     self.fc2 = nn.Linear(120, 84)
     self.fc3 = nn.Linear(84, 10)

   def forward(self, x) -> Tensor:
     x = self.pool(F.relu(self.conv1(x)))
     x = self.pool(F.relu(self.conv2(x)))
     x = torch.flatten(x, 1)
     x = F.relu(self.fc1(x))
     x = F.relu(self.fc2(x))
     x = self.fc3(x)
     return x
```

DSPy ⟵⟶ PyTorch

- Example: program a multi-hop QA system
  - A QA system that answer to a question by iteratively increasing its context

```
class MultiHopQA(dspy.Module):
    def __init__(self):
        super().__init__()
        self.retrieve = dspy.Retrieve(k=3)
        self.query_generation = dspy.Predict("context, question -> query")
        self.answer_generation = dspy.ChainOfThought("context, question -> answer")

    def forward(self, question):
        context = []

        for hop in range(2):
            query = self.query_generation(context=context, question=question).query
            context += self.retrieve(query=query).passages

        return self.answer_generation(context=context, question=question)
```

DSPy ⟷ PyTorch

- Example: multi-hop QA system
  - A QA system that answer to a question by iteratively increasing its context

```
class MultiHopQA(dspy.Module):
    def __init__(self):
        super().__init__()
        self.retrieve = dspy.Retrieve(k=3)
        self.query_generation = dspy.Predict("context, question -> query")
        self.answer_generation = dspy.ChainOfThought("context, question -> answer")

    def forward(self, question):
        context = []

        for hop in range(2):
            query = self.query_generation(context=context, question=question).query
            context += self.retrieve(query=query).passages

        return self.answer_generation(context=context, question=question)
```

BUILDING BLOCKS

DSPy ⟷ PyTorch

- Example: multi-hop QA system
  - A QA system that answer to a question by iteratively increasing its context

```
class MultiHopQA(dspy.Module):
    def __init__(self):
        super().__init__()
        self.retrieve = dspy.Retrieve(k=3)
        self.query_generation = dspy.Predict("context, question -> query")
        self.answer_generation = dspy.ChainOfThought("context, question -> answer")

    def forward(self, question):
        context = []

        for hop in range(2):
            query = self.query_generation(context=context, question=question).query
            context += self.retrieve(query=query).passages

        return self.answer_generation(context=context, question=question)
```

PROGRAM FLOW

DSPy ⟷ PyTorch

- Example: multi-hop QA system
  - A QA system that answer to a question by iteratively increasing its context

```
class MultiHopQA(dspy.Module):
    def __init__(self):
        super().__init__()
        self.retrieve = dspy.Retrieve(k=3)
        self.query_generation = dspy.Predict("context, question -> query")
        self.answer_generation = dspy.ChainOfThought("context, question -> answer")

    def forward(self, question):
        context = []

        for hop in range(2):
            query = self.query_generation(context=context, question=question).query
            context += self.retrieve(query=query).passages

        return self.answer_generation(context=context, question=question)
```

INPUT/OUTPUT DEFINITION

DSPy is much more!

- Clean up your prompts and structure your input and output
  - DSPy Signatures

- Plug&Play specialised LLM modules ready to use:
  - DSPy Modules: `Predict`, `Retrieve`, `ChainOfThought`, `ReAct`…

- Optimizers to assist the user in LLM development
  - DSPy Optimzers: `BootstrapFewShot`, `SignatureOptimizer`…

- Control Flow in LLM programs
  - DSPy Assertions, standard python control flow (if, for, while loops…)

- Evaluation and Metrics back in the loop of LLM project life-cycle
  - DSPy Evaluate and metrics definition

DEMO

DSPy