
DSPy
Proogrammmin ing
— not proming
} Foundation
} Moodes. {.

Introducing DSPy

“*Programming* - not prompting - Foundation Models”

10.12.2024 | Riccardo Belluzzo
Poznański Horyzont Danych Meetup @ Allegro

Agenda

- Motivation
- Introduction to DSPy
- Demo

Have LLMs changed the ML job?

- My personal experience as a Research Engineer in Allegro

Have LLMs changed the ML job?

- **My personal experience as a Research Engineer in Allegro**
  - Several “GenAI adoption” PoCs conducted in 2 years time

Have LLMs changed the ML job?

- **My personal experience as a Research Engineer in Allegro**
  - Several “GenAI adoption” PoCs conducted in 2 years time
  - Development of an open-source library for querying LLMs
    - https://github.com/allegro/allms

Have LLMs changed the ML job?

- **My personal experience as a Research Engineer in Allegro**
  - Several “GenAI adoption” PoCs conducted in 2 years time
  - Development of an open-source library for querying LLMs
    - https://github.com/allegro/allms
  - Starting using Github Copilot at daily basis
  - …

What has mostly changed?

- **More interaction with the business unit** 🤝
  - “Business-oriented” Prompt Engineering

What has mostly changed?

- **More interaction with the business unit** 🤝
  - “Business-oriented” Prompt Engineering
- **More time spent on defining pipelines rather than modeling** 🔗
  - Making sure the LLM works nicely in the ecosystem

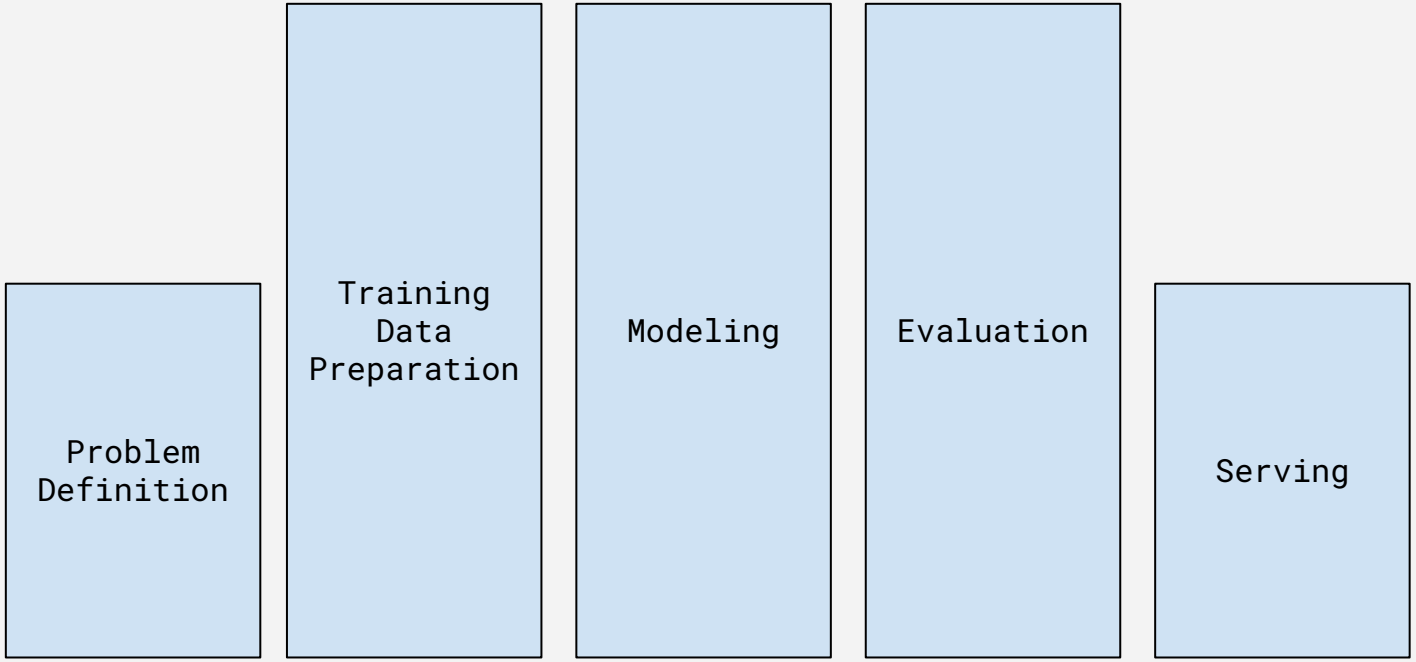
What has mostly changed?

- **More interaction with the business unit** 🤝
  - “Business-oriented” Prompt Engineering
- **More time spent on defining pipelines rather than modeling** 🔗
  - Making sure the LLM works nicely in the ecosystem
- **More focus on evaluation rather than training** 📊
  - LLM evaluation is still an open problem

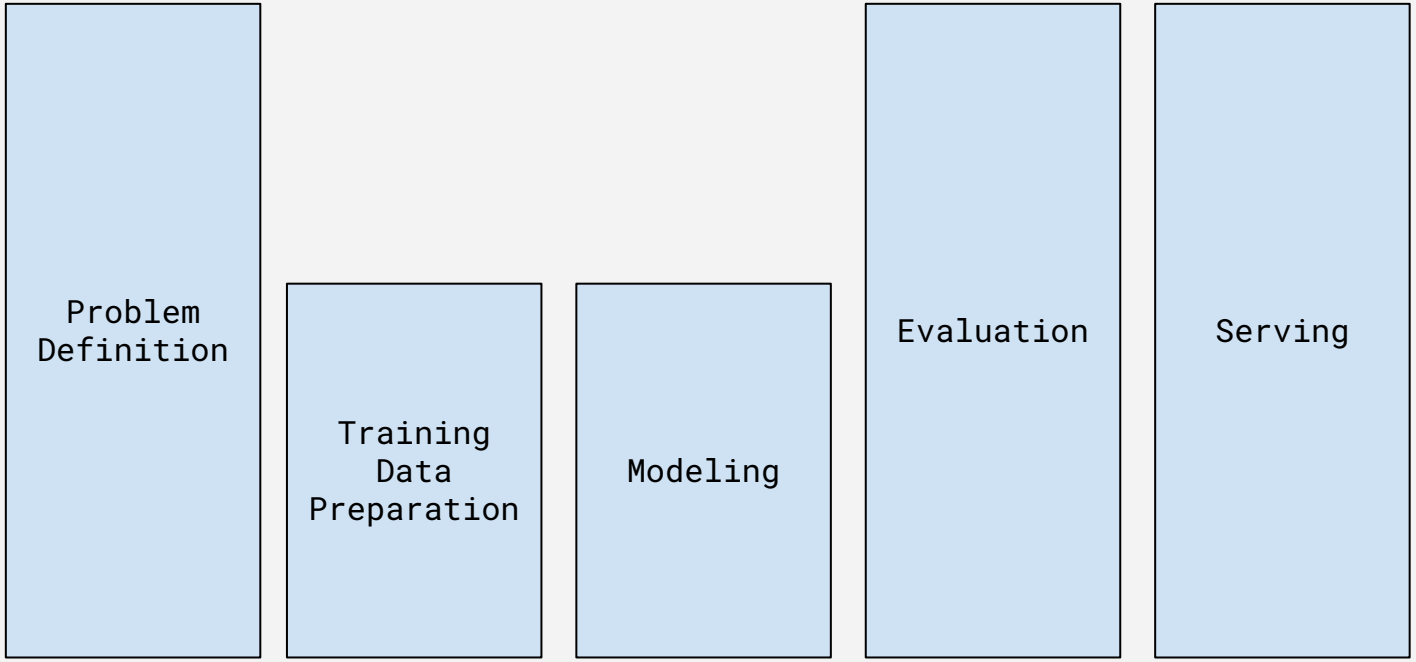
What has mostly changed?

- **More interaction with the business unit** 🤝
  - “Business-oriented” Prompt Engineering
- **More time spent on defining pipelines rather than modeling** 🔗
  - Making sure the LLM works nicely in the ecosystem
- **More focus on evaluation rather than training** 📊
  - LLM evaluation is still an open problem
- **More focus on topics that earlier didn’t bother me at all!** 📋
  - Cost estimation, security, etc.

What has mostly changed? [VISUALIZED]



What has mostly changed? [VISUALIZED]



Are NLP engineers doomed to be ethologists?

Prompt

↓

Observe

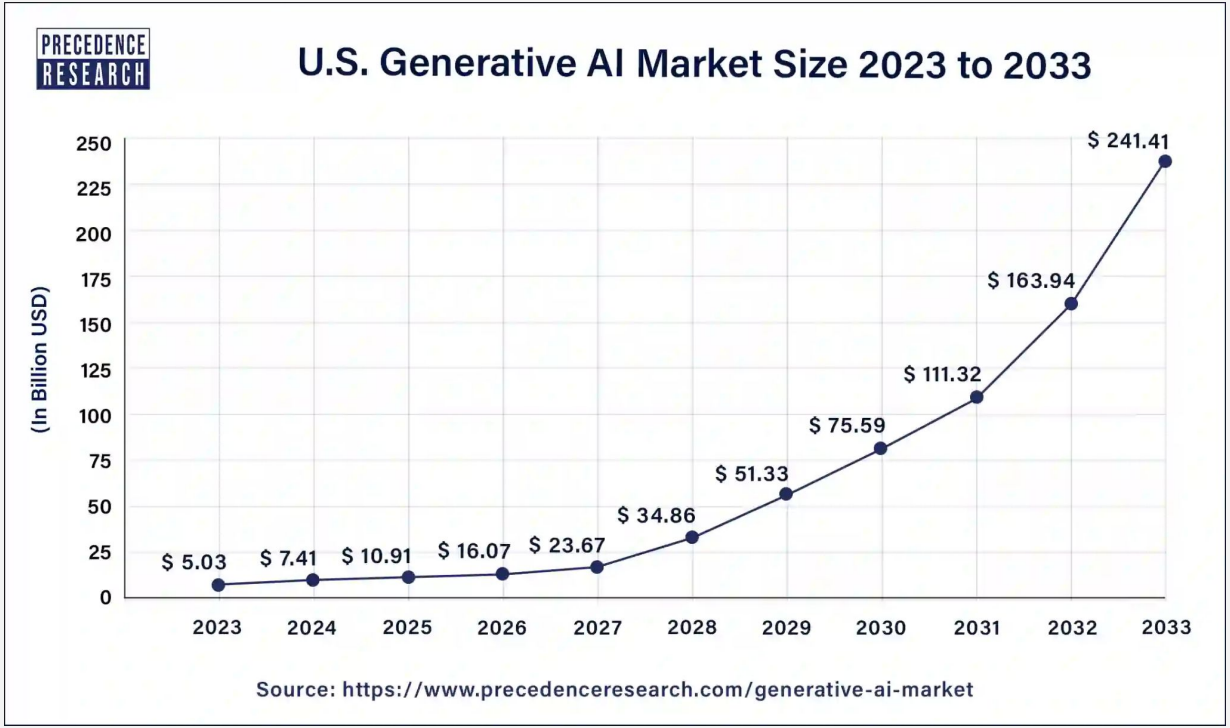
↓

Repeat

Physics of LLMs - ICML 24 tutorial

Come on... it’s not just “Prompt Engineering”

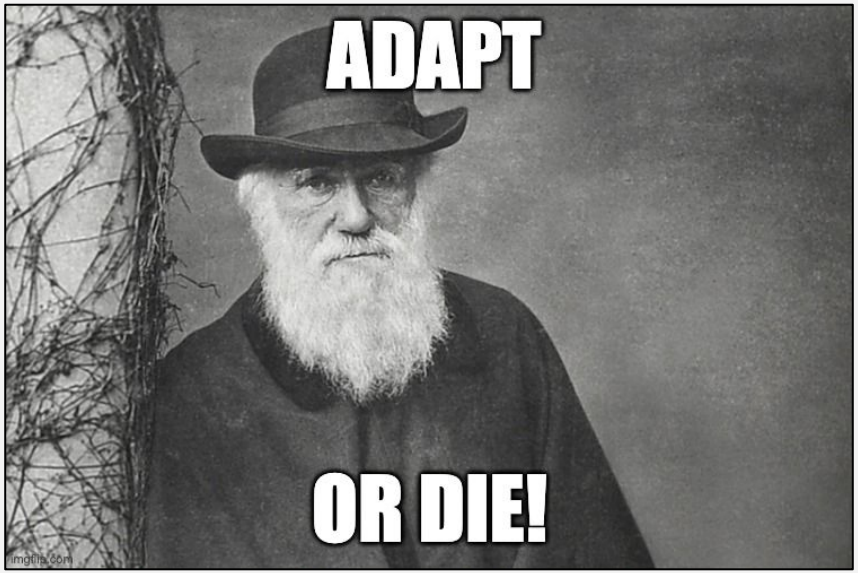
LLMs are here to stay



“The Times They Are A-Changin'...”

“The Times They Are A-Changin'...”

… but a good ML engineer embraces the change!



Like old Charles Darwin once said:

Follow the scientific method! 🔬

Like old Charles Darwin once said:

Follow the scientific method! 🔬

The ML engineer (scientific) method:

Like old Charles Darwin once said:

Follow the scientific method! 🔬

The ML engineer (scientific) method:

- Set up an experimental framework;

Like old Charles Darwin once said:

Follow the scientific method! 🔬

The ML engineer (scientific) method:

- Set up an experimental framework;
- Break down the problem into steps;

Like old Charles Darwin once said:

Follow the scientific method! 🔬

The ML engineer (scientific) method:

- Set up an experimental framework;
- Break down the problem into steps;
- Conduct isolated experiments to validate hypothesis;

Like old Charles Darwin once said:

Follow the scientific method! 🔬

The ML engineer (scientific) method:

- Set up an experimental framework;
- Break down the problem into steps;
- Conduct isolated experiments to validate hypothesis;
- Follow an evaluation protocol to compare different solutions.

Introducing...

DSPy

DSPy

- DSPy stands for **Declarative Self-improving Python**

DSPy

- DSPy stands for **Declarative Self-improving Python**
- Instead of brittle prompts, you write **compositional Python code**

DSPy

- DSPy stands for **Declarative Self-improving Python**
- Instead of brittle prompts, you write **compositional Python code**
- DSPy's tools teach your LLM to deliver high-quality outputs

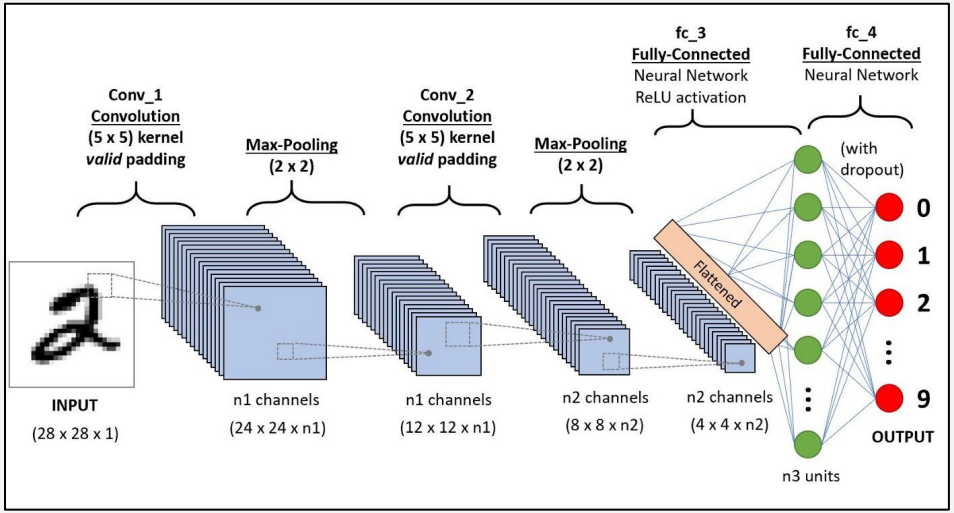
DSPy

- DSPy stands for **Declarative Self-improving Python**
- Instead of brittle prompts, you write **compositional Python code**
- DSPy's tools teach your LLM to deliver high-quality outputs
  - They separate the flow of your **program** from the **parameters (LM prompts and weights)** of each step;

DSPy

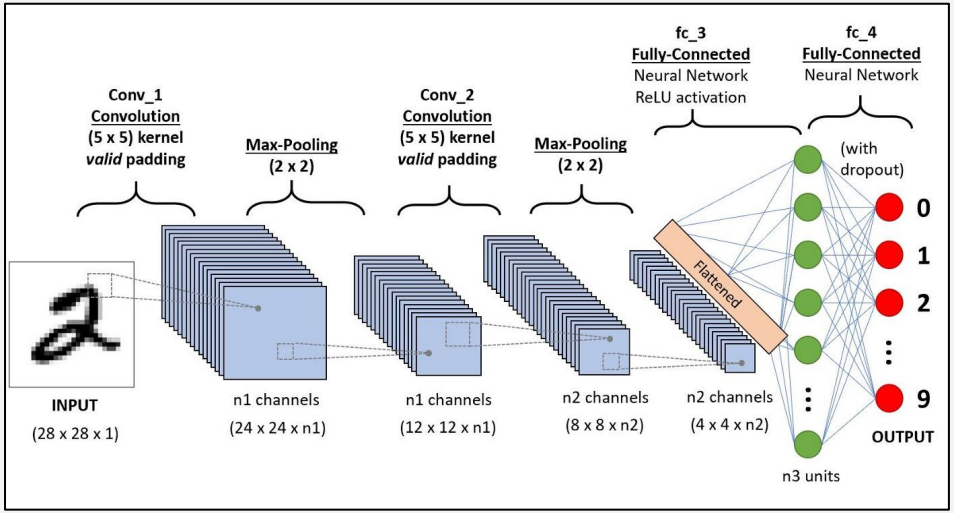
- DSPy stands for **Declarative Self-improving Python**
- Instead of brittle prompts, you write **compositional Python code**
- DSPy's tools teach your LLM to deliver high-quality outputs
  - They separate the flow of your **program** from the **parameters (LM prompts and weights)** of each step;
  - It introduces **optimizers, metrics and evaluation loops** for tuning the parameters of your program as a standard ML experimental framework.

A Deep Learning Program...



A Deep Learning Program...

… written in PyTorch

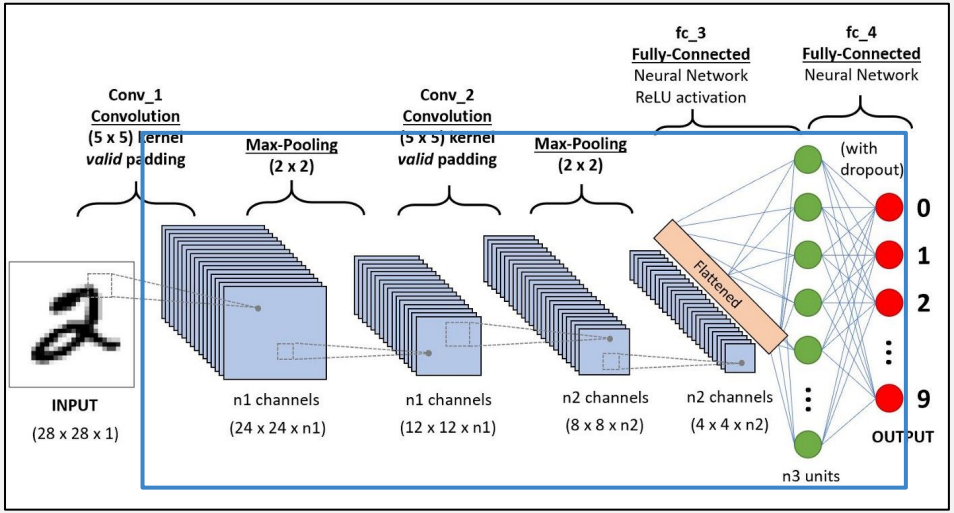



```python
class ConvNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3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



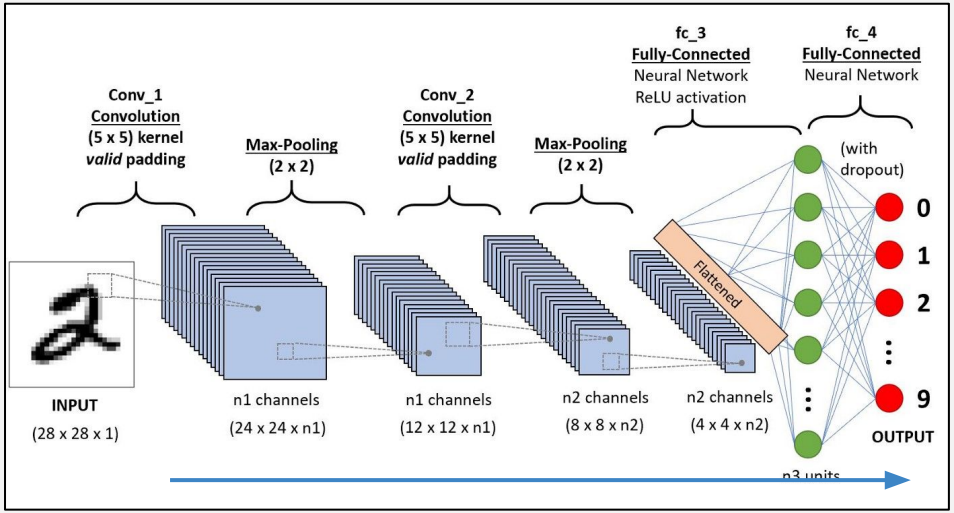

```python
class ConvNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3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

BUILDING BLOCKS

A Deep Learning Program...

… written in PyTorch



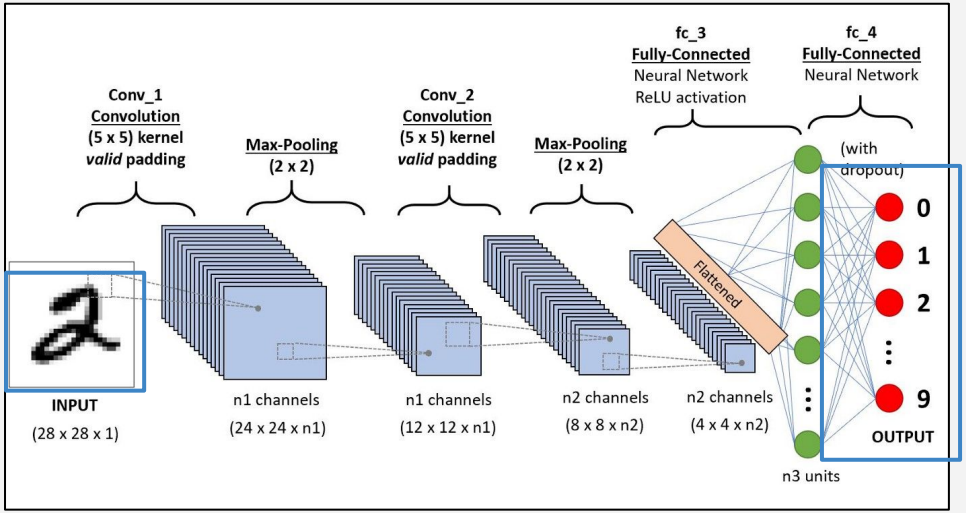

```python
class ConvNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3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

PROGRAM FLOW

A Deep Learning Program...

… written in PyTorch




```python
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
```

INPUT/OUTPUT DEFINITION

(we don't care too much about what's happening in the middle)

DSPy: LLM = PyTorch: Deep Learning

LLM task: multi-hop QA system

```python
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: LLM = PyTorch: Deep Learning

LLM task: multi-hop QA system

BUILDING BLOCKS

```python
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: LLM = PyTorch: Deep Learning

LLM task: multi-hop QA system

```python
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: LLM = PyTorch: Deep Learning

LLM task: multi-hop QA system

```python
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

The DSPy Universe

- **Prompt encapsulation and versioning** through:
  - 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`, `KNNFewShot`, `COPRO`[1], `MIPROv2`[2]

- And much more:
  - DSPy Assertions;
  - DSPy Ensembles;
  - ...

1. Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q. V., Zhou, D., & Chen, X. (2023). Large Language Models as Optimizers. arXiv
2. Opsahl-Ong, K., Ryan, M. J., Purtell, J., Broman, D., Potts, C., Zaharia, M., & Khattab, O. (2023). Optimizing Instructions and Demonstrations for Multi-Stage Language Model Programs. arXiv.

DEMO

DSPy

All code available: https://github.com/riccardo-alle/dspy-demo

DSPy: personal takeaways

- **DSPy is meant for modeling, not deployment;**
  - For a nice overview of “what is DSPy meant for?” consult these FAQs
- DSPy enforces **modularization** of your program, improving readability and customization of the solution;
- **DSPy Signatures** are a nice concept to encapsulate and version prompts;
- **Prompt tuning optimizers can’t make miracles**
  - But it’s interesting step towards “AutoML” in the reign of LLMs
- **A lot of the magic is too hidden**
  - At the current state of LLMs, we still want full control on every char of the prompt;
- **Fast-growing research-based library**
  - It means also instability and not very well documented “researchy” code

LLM-as-a-judge metrics

```python
class ProductReviewsSummarizationQualityAssesment(dspy.Signature):
    """Assess the quality of a summary based on the following criteria:
    * product focus: is the summary focused on product descriptions and not single users experience;
    * consistency: is the summary consistent and does not present contradictory information;
    * language: is the summary written with proper syntax, gramar and non-offensive language.
    """

    summary = dspy.InputField(desc="Machine generated product reviews summary")
    product_focus = dspy.OutputField(desc="Yes if the summary is product-focused, No otherwise")
    consistency = dspy.OutputField(desc="Yes if the summary is consistent, No otherwise")
    language = dspy.OutputField(desc="Yes if the summary uses proper language, No otherwise")

# Use a more powerful model as a judge
llm_as_a_judge = dspy.AzureOpenAI(deployment_id="gpt4-o1")

# Later use this function as `dspy.Evaluate(metric=validate_summarization_quality, ...)`
def validate_summarization_quality(pred: SummarizationPrediction) -> bool:

    pred_summary = pred.summary

    with dspy.context(lm=llm_as_a_judge):
        assessment = dspy.ChainOfThought(ProductReviewsSummarizationQualityAssesment)(
            summary=pred_summary
        )

        is_product_focused = assessment.product_focus.lower() == "yes"
        is_consistent = assessment.consistency.lower() == "yes"
        uses_proper_language = assessment.proper_language.lower() == "yes"

    return is_product_focused and is_consistent and uses_proper_language
```

LLM Ensembles

```python
import dspy
from dspy.teleprompt import Ensemble

programs = [Program_1(), Program_2()]
teleprompter = Ensemble(reduce_fn=dspy.majority, size=2)
ensembled_program = teleprompter.compile(programs)

# Use an ensemble of programs to get predictions
ensembled_program(llm_input)
```

Either uses the full set or randomly samples a subset into a single program.

LLM Ensembles (of optimized programs)

```python
from dsp.utils import flatten, deduplicate

agents = [x[-1] for x in optimized_program.candidate_programs[:5]]

class AgentsEnsemble(dspy.Module):
    def __init__(self):
        self.aggregate = dspy.ChainOfThought('context, question -> answer')
        self.temperature = temperature

    def forward(self, question):
        with dspy.context(lm=llm_copy.copy():
            preds = [agent(question=question) for agent in agents]
            context = deduplicate(flatten([flatten(p.observations) for p in preds]))

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