TopoCoend Triage

Learning a latent clinical cover before Kan aggregation

Simon Frost

Table of contents

Setup

```
using FunctorFlow
```

`topocoend_block` is the categorical pattern where a system first infers a cover over tokens and then applies a Kan-style aggregation over that inferred relation. This vignette uses a small clinical-triage toy problem: local signals such as oxygen saturation, lactate, and presenting symptoms induce latent contexts like respiratory or metabolic concern.

Build the Example

```
example = build_topocoend_triage_example()
diagram = example[:diagram]
println(summary(diagram))
```

```
FunctorFlow.Diagram
```

The example wires two explicit morphisms:

1. `infer_context_cover`: infer a latent cover from signal tags.
2. `lift_signal_priority`: turn structured signals into scalar priorities.

The left-Kan stage then aggregates those lifted priorities into context-level risk summaries.

Execute the Learned Cover

```
result = execute_topocoend_triage_example(example)
println("Learned cover: ", result.values[diagram.ports[:learned_relation].ref])
println("Context risk: ", result.values[diagram.ports[:output].ref])
```

```
Learned cover: Dict{Any, Vector{Any}}(:respiratory_focus => [:oxygen_saturation, :chief_compla
Context risk: Dict(:respiratory_focus => 0.8166666666666668, :metabolic_focus => 0.77499999999
```

Here the same local signal can belong to multiple inferred contexts. For instance, oxygen saturation participates in both the respiratory and hemodynamic covers.

Summarize the TopoCoend Output

```
summary = summarize_topocoend_triage_example(example)
println("Highest-priority context: ", summary["highest_priority_context"])
println("Context risks: ", summary["context_risks"])
println("Learned relation: ", summary["learned_relation"])
```

```
Highest-priority context: respiratory_focus
Context risks: Dict("respiratory_focus" => 0.8166666666666668, "metabolic_focus" => 0.77499999
Learned relation: Dict("respiratory_focus" => ["chief_complaint", "history", "oxygen_saturatio
```

The point is not the toy numbers; it is the geometry. The global context is not aggregated over a fixed adjacency. Instead, the cover itself is inferred from typed local evidence and only then used as the indexing relation for the Kan aggregation.

Reusing the Helper Directly

The reusable helpers are also exported directly:

```
cover = infer_topocoend_cover(example[:token_payloads];
    context_prototypes=example[:context_prototypes],
    min_overlap=1)
scores = lift_topocoend_scores(example[:token_payloads])

println("Cover via helper: ", cover)
println("Lifted scores: ", scores)
```

```
Cover via helper: Dict{Any, Vector{Any}}(:respiratory_focus => [:oxygen_saturation, :chief_com
Lifted scores: Dict(:oxygen_saturation => 0.9, :chief_complaint => 0.95, :history => 0.6, :lac
```

This makes `TopoCoend` practical outside the block builder itself: the same cover-inference and lifting helpers can be plugged into larger world-model or agent pipelines.

For the broader CATAGI symbolic overview, see 18 Neurosymbolic Pipelines. For the trainable version of this pattern, see `build_topocoend_lux_model` and `RelationInferenceLayer` in 07 Lux Neural Backend.