

CATGRAD: A Categorical Compiler for Deep Learning

Paul Wilson

paul@statusfailed.com

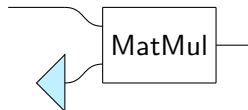
CATGRAD is an open-source compiler for deep learning.¹ Models are represented as *string diagrams*: morphisms constructed by tensor and composition from a set of generating operations. This is a completely syntactic representation which is then *compiled* to a target backend. Instead of using autograd for differentiation, models are mapped functorially into a category of optics to produce a morphism representing a single optimization step of the model. Practically, this means that catgrad can compile a model into static training code which runs without requiring a deep learning framework or autograd implementation.


As a demonstration of the compiler, we provide CATGPT: a string-diagrammatic implementation of a GPT model in CATGRAD.²

CATGRAD is a deep learning *compiler*, as opposed to a framework. This distinction highlights that models are represented purely syntactically as morphisms in a strict monoidal category, as opposed to as arbitrary code in the host language.

Users construct models (i.e. morphisms) inductively by tensor and composition from generating operations. Internally, morphisms are represented as ‘open hypergraphs’ [3, 4]—specifically using the data-parallel datastructure described in [13] (c.f. [12]) based on structured cospans of acsets [9, 1].³

As a simple example, a linear model in CATGRAD is represented by the following string diagram.



This morphism is constructed from two generating operations: a matrix multiplication, and a Parameter : $0 \rightarrow 1$ operation depicted as  indicating a value to be learned. The model is constructed in code as follows, using the @ and >> operators for tensor and composition, respectively.

```
def linear(A: NdArrayType, B: NdArrayType, C: NdArrayType):  
    return (identity(obj(A+B)) @ parameter(obj(B+C))) >> op(Compose(A,B,C))
```

Each A, B, C in the above is a generating object of the category: a value of type NdArrayType consisting of a *shape* and *dtype*. For example, NdArrayType((4, 3, 2), Dtype.int32) is the generating object representing 3D arrays with $4 \cdot 3 \cdot 2 = 24$ elements, each a 32-bit integer.

Once a model has been constructed as a morphism, it can be compiled. The linear model depicted above compiles to the following Python code, where variable names p, x, y have been changed for readability.

¹<https://github.com/statusfailed/catgrad>

²<https://github.com/statusfailed/catgpt>.

³The original datastructure library is available at github.com/yarrow-id/diagrams. We have made several improvements and optimizations in our version: github.com/statusfailed/open-hypergraphs.

```

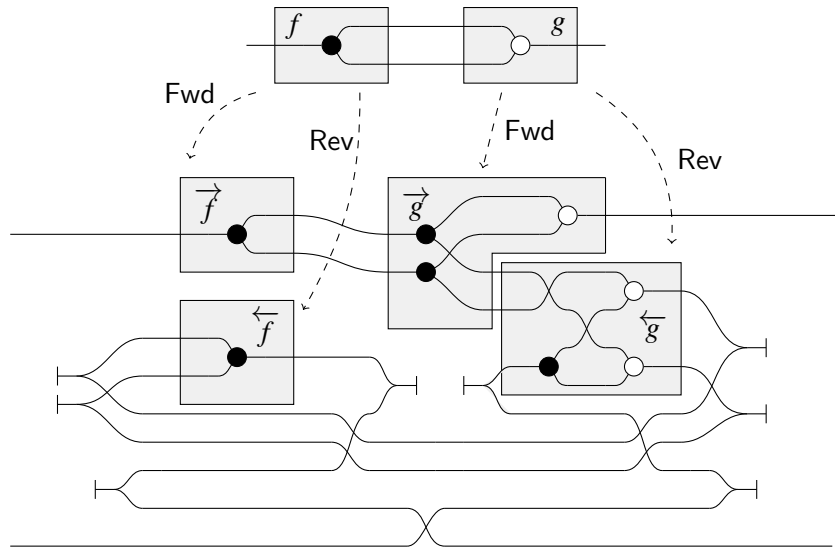
class Dynamic:
    ... # omitted code
    def predict(self, p, x):
        y = x @ p # here @ denotes matrix multiplication
        return [y]

```

Notice that model parameters appear as *inputs* to the compiled `predict` function. Before compiling a model $f : X \rightarrow Y$, it is factored as $f = (p \otimes \text{id}_X) \circ f'$, where p is a tensoring of Parameter operations in f . The morphism $f' : P \times X \rightarrow Y$ is then passed to the backend for compilation.

Training, differentiation, and optimization

In order to *train* a model, it is transformed into an optic using the algorithm defined in [13, Section 10], itself inspired by observations from [2, 11, 7]. The example below depicts a string diagram for the function $x \mapsto x^2$ and its corresponding ‘optic diagram’ after this transformation.



This ‘bidirectional mapping’ maps each generating operation f into a *forward* morphism \vec{f} , and *reverse* morphism \overleftarrow{f} (a reverse derivative [5]), and a *residual* object. The residual dictates what information is passed from the forward to reverse map: in the case of linear morphisms like \bullet , the residual is the unit object: no information is passed. Some generators like \circlearrowleft : $A \times A \rightarrow A$ are *lenses*: the residual is the source object, and inputs are copied and passed to the reverse map.

Note that this map represents the computation of both the function $x \mapsto x^2$ and its reverse derivative $x, \delta_y \mapsto 2x \cdot \delta_y$ simultaneously. Further, while the optic transformation requires use of Frobenius structure to ‘bend wires around’, the resulting map is monogamous acyclic [3, 4]. It therefore represents a morphism of a symmetric monoidal category, meaning it is possible to compile it to a *function*.

Finally, by pre- and post-composing the model with ‘update’ and ‘displacement’ optics as described in [6], the bidirectional transformation results in a morphism representing a single optimization step of the model; theoretical details can be found in [6] and [14].

CATGPT

As a demonstration, we have implemented a GPT model in CATGRAD whose architecture is a simplified version of nanoGPT [8], itself based on GPT-2 [10]. Code is available at <https://github.com/statusfailed/catgpt>.

References

- [1] John C. Baez & Kenny Courser (2019): *Structured Cospans*. doi:10.48550/ARXIV.1911.04630. Available at <https://arxiv.org/abs/1911.04630>.
- [2] Guillaume Boisseau (2020): *String Diagrams for Optics*. arXiv:2002.11480.
- [3] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobociński & Fabio Zanasi (2016): *Rewriting modulo symmetric monoidal structure*. In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, ACM, doi:10.1145/2933575.2935316. Available at <https://doi.org/10.1145/2933575.2935316>.
- [4] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobocinski & Fabio Zanasi (2022): *String Diagram Rewrite Theory I: Rewriting with Frobenius Structure*. arXiv:2012.01847.
- [5] Robin Cockett, Geoffrey Cruttwell, Jonathan Gallagher, Jean-Simon Pacaud Lemay, Benjamin MacAdam, Gordon Plotkin & Dorette Pronk (2019): *Reverse derivative categories*, doi:10.4230/LIPIcs.CSL.2020.18.
- [6] G. S. H. Cruttwell, Bruno Gavranović, Neil Ghani, Paul Wilson & Fabio Zanasi (2021): *Categorical Foundations of Gradient-Based Learning*, doi:10.48550/ARXIV.2103.01931. Available at <https://arxiv.org/abs/2103.01931>.
- [7] Bruno Gavranović (2022): *Space-time tradeoffs of lenses and optics via higher category theory*. arXiv:2209.09351.
- [8] Andrej Karpathy: *nanoGPT*. Available at <https://github.com/karpathy/nanoGPT>.
- [9] Evan Patterson, Owen Lynch & James Fairbanks (2022): *Categorical Data Structures for Technical Computing*. *Compositionality* 4, p. 5, doi:10.32408/compositionality-4-5. Available at <https://doi.org/10.32408/compositionality-4-5>.
- [10] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei & Ilya Sutskever (2019): *Language Models are Unsupervised Multitask Learners*.
- [11] Mitchell Riley (2018): *Categories of Optics*. arXiv:1809.00738.
- [12] Paul Wilson & Fabio Zanasi (2022): *The Cost of Compositionality: A High-Performance Implementation of String Diagram Composition*. *Electronic Proceedings in Theoretical Computer Science* 372, p. 262–275, doi:10.4204/eptcs.372.19. Available at <http://dx.doi.org/10.4204/EPTCS.372.19>.
- [13] Paul Wilson & Fabio Zanasi (2023): *Data-Parallel Algorithms for String Diagrams*. arXiv:2305.01041.
- [14] Paul William Wilson (2023): *Category-theoretic data structures and algorithms for learning polynomial circuits*. Ph.D. thesis, University of Southampton. Available at <https://eprints.soton.ac.uk/483757/>.