Tensor Species

Andrew Dudzik[1], Petar Veličković[1], Tamara von Glehn[1], Razvan Pascanu[1], Bruno Gavranović, Paul Lessard[2], João Araújo[1]... and more to come!

[1]Google DeepMind

[2]Symbolica AI

February 2025

What I'm Doing Here

I'm here representing a small but enthusiastic "theory of code" group at Google DeepMind.

We believe that abstract mathematics can inform the design of new neural network architectures.

...even though history suggests otherwise.

Some Problems With Theory

- Theory, in general, often aims at the wrong target.
- Classical programs and neural programs differ substantially, so classical theory can be misleading.
- Neural networks are not sufficiently advanced to take "internal" advantage of high-level languages. Architectures are close to being machine code.
- Highly distributed computing is rarely hardware-agnostic.

Categorical Deep Learning

Last year, we wrote a paper with Bruno and Paul—a category theory paper disguised as a machine learning position paper.

The main thrust was to show that "monadic programming" could be lifted to "2-monadic programming" and applied to the setting where the computation has learnable parameters.

In this way, we can augment standard constructions in functional programming so they're compatible with various notions of reparameterization and weight sharing.

It's a clean picture... or is it?

Position: Categorical Deep Learning is an Algebraic Theory of All Architectures

Bruno Gavranović [*12] **Paul Lessard** [*1] **Andrew Dudzik** [*3]
Tamara von Glehn [3] **João G.M. Araújo** [3] **Petar Veličković** [34]

Abstract

We present our position on the elusive quest for a general-purpose framework for specifying and studying deep learning architectures. Our opinion is that the key attempts made so far lack a coherent bridge between specifying *constraints* which models must satisfy and specifying their *implementations*. Focusing on building a such a bridge, we propose to apply category theory—precisely, the universal *algebra of monads* valued in a 2-category of *parametric maps*—as a single theory elegantly subsuming both of these flavours of neural network de-

networks can be specified in a *top-down* manner, wherein models are described by the *constraints* they should satisfy (e.g. in order to respect the structure of the data they process). Alternatively, a *bottom-up* approach describes models by their *implementation*, i.e. the sequence of tensor operations required to perform their forward/backward pass.

1.1. Our Opinion

It is our **opinion** that ample effort has already been given to both the top-down and bottom-up approaches *in isolation*, and that there hasn't been sufficiently expressive theory to address them both *simultaneously*. *If we want a* ***general*** *guiding*

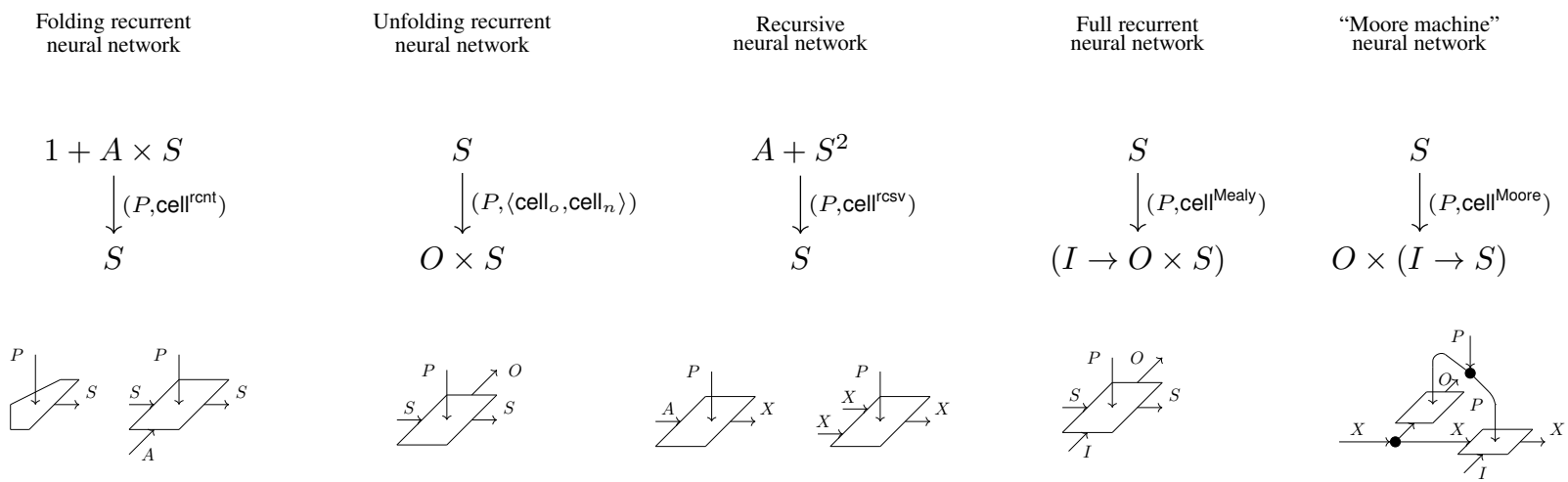
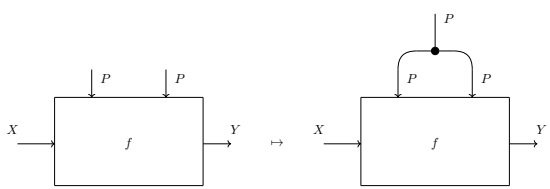


Figure 1. Parametric (co)algebras provide a high-level framework for describing structured computation in neural networks.



the *lax algebras* are sufficient to derive *recursive*, *recurrent*, and similar neural networks from first principles. Notably, morphisms of lax algebras are also expressive enough to capture *1-cocycles*, used to formalise asynchronous neural networks in (Dudzik et al., 2024)—see Appendix H.1.

Why This Is Problematic

Suppose I have a datatype Y that is stored in memory using b bits. Then Y^2 needs $2b$ bits, Y^3 needs $3b$ bits, etc. So if Y is affordable, in all likelihood most powers of Y are affordable.

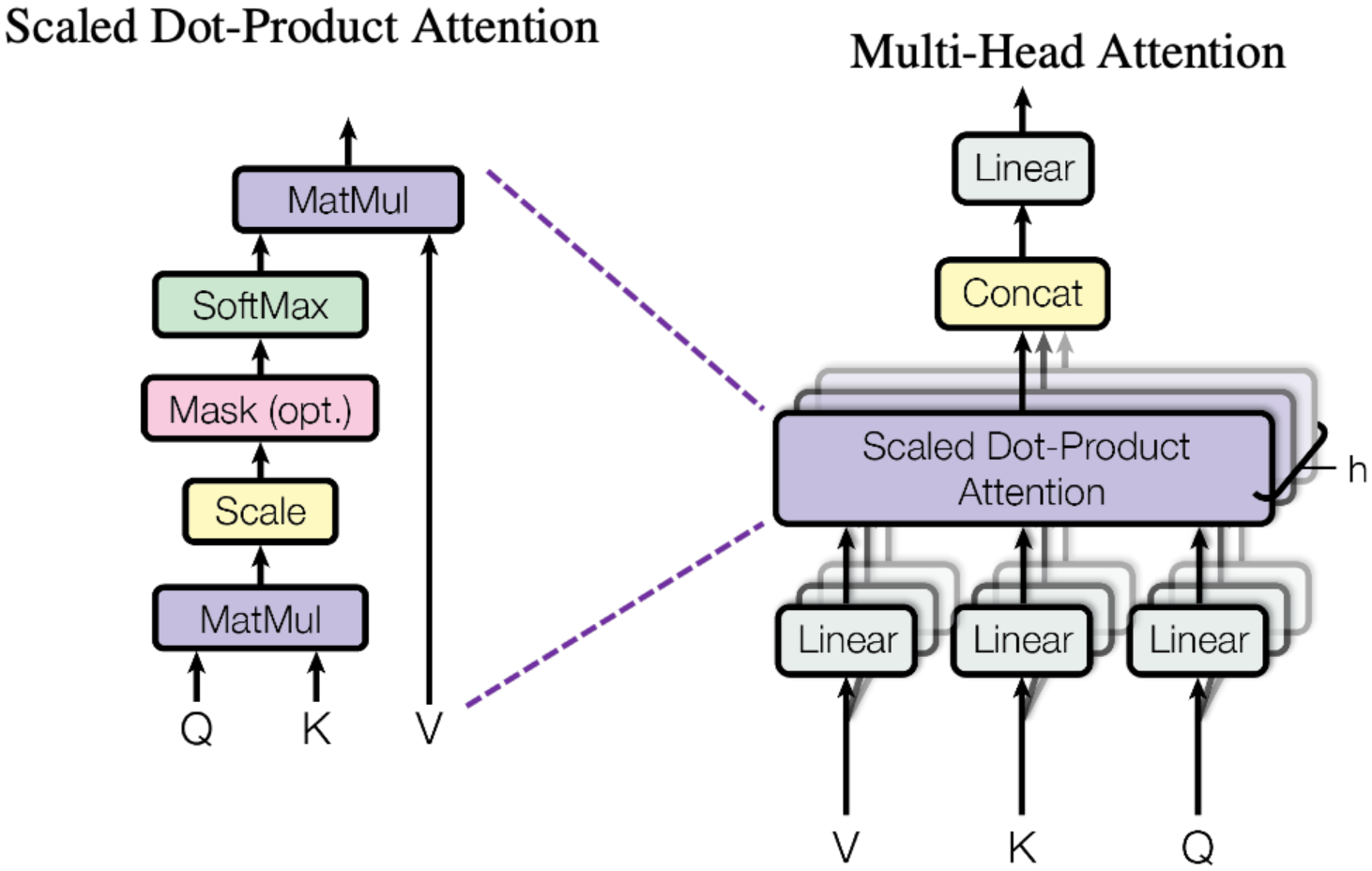
But in neural networks, everything is vectorized, and the situation is quite different. Let c be the number of bits used in our implementation of the reals.

If a vector datatype V requires $c \cdot b$ bits, then $V^{\otimes 2}$ needs $c \cdot b^2$ bits, $V^{\otimes 3}$ needs $c \cdot b^3$ bits, etc. Higher powers are no longer affordable!

So classical computing can be **dangerous** in neural networks. Simply porting existing ideas won't do.

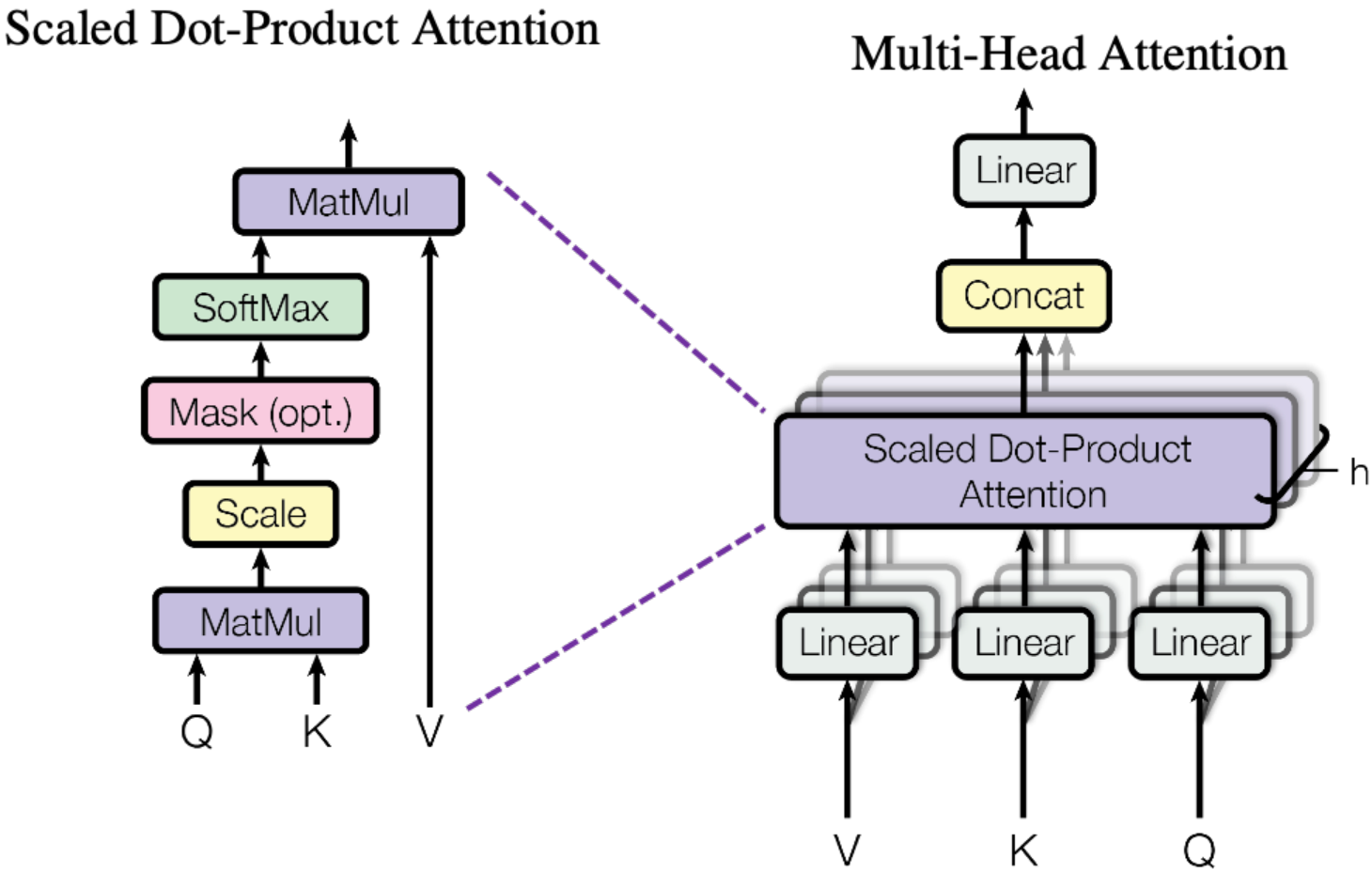
The Big Problem: Attention

The attention mechanism fundamentally changed how we design neural networks. It was popularized in Attention Is All You Need (Vaswani et al., 2017) but appeared earlier in Neural Turing Machines (Graves, Wayne, Danihelka, 2014).



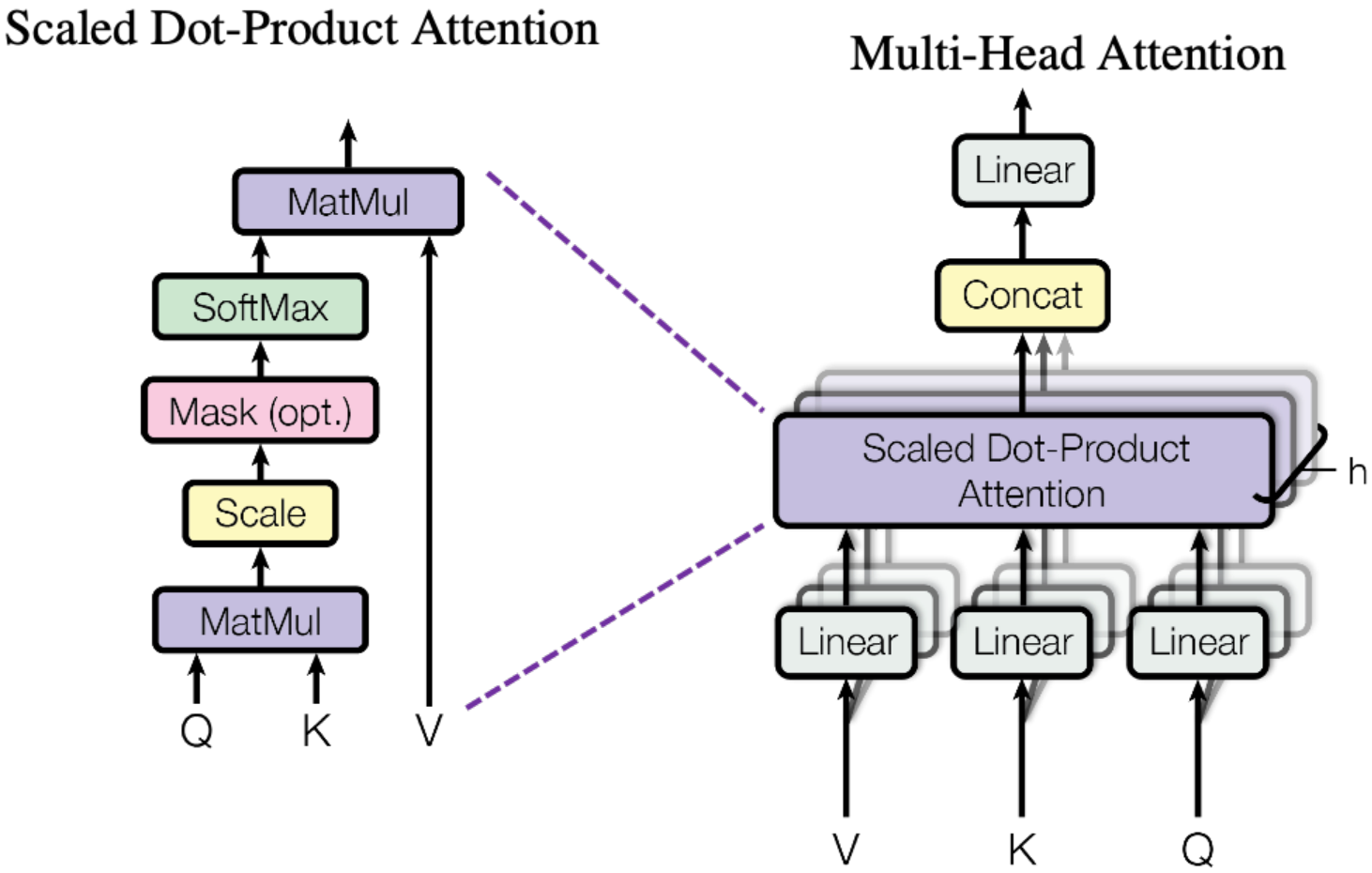
The Big Problem: Attention

Prior to ChatGPT, attention was severely under-appreciated, particularly outside of the professional ML community. Academic work focused on RNNs, ConvNets, and other textbook examples.



The Big Problem: Attention

Strangely, nobody seems to know a good "purely algebraic" theory of attention... even though superficially, the math isn't complicated.



Talking-Heads Attention

Noam Shazeer*,
Google
noam@google.com

Zhenzhong Lan*
Google
lanzhzh@google.com

Youlong Cheng*
Google
ylc@google.com

Nan Ding*
Google
dingnan@google.com

Le Hou*
Google
lehou@google.com

March 6, 2020

Abstract

We introduce "talking-heads attention" - a variation on multi-head attention which includes linear projections across the attention-heads dimension, immediately before and after the softmax operation. While inserting only a small number of additional parameters and a moderate amount of additional computation, talking-heads attention leads to better perplexities on masked language modeling tasks, as well as better quality when transfer-learning to language comprehension and question answering tasks.

1 Introduction

Neural Attention was introduced by [Bahdanau et al., 2014] as a way of extracting information from variable-length representations. The Transformer model [Vaswani et al., 2017] uses "multi-head" attention, consisting of multiple attention layers ("heads") in parallel, each with different projections on its inputs and outputs. By using a dimensionality reduction in the input projections, the computational cost is kept similar to that of basic attention. Quality is improved, presumably due to the ability to attend to multiple positions simultaneously based on multiple different types of relationships.

As noted in [Vaswani et al., 2017][1], taking this process to the extreme (more attention heads projected to lower dimensionality) becomes counterproductive. We believe that this is due to the fact that the query-vectors and key-vectors become so low-dimensional that their dot product can no longer constitute an informative matching function.

In this paper, we introduce a new variant, "talking-heads attention", that addresses this problem by inserting a learned linear projection across the attention-heads dimension of the attention-logits tensor. This allows each attention function to depend on all of the keys and queries. We also insert a second such projection immediately following the softmax.

We show experimentally that inserting these "talking-heads" projections leads to better perplexities on masked language modeling tasks, as well as better quality when transfer-learning to language comprehension and question answering tasks.

2 Notation

In our pseudocode, we use capital letters to represent tensors and lower-case letters to represent their dimensions. Each tensor is followed by a dimension list in brackets. For example, a 4-dimensional image-

*Noam Shazeer devised the talking-heads architecture, ran the T5 experiments and wrote most of the paper. Zhenzhong Lan had the initial idea of talking-heads attention, designed and coordinated part of the experiments. Youlong Cheng reproduced BERT in MeshTensorFlow and run all the talking heads experiments for MeshTensorFlow BERT. Nan Ding ran the ALBERT experiments. Le Hou visualized and analyzed the learned weights of talking-heads.

[1] Section (A) of table 3 in [Vaswani et al., 2017]. Also the first sections of tables 1 and 5 of this paper.

Simple Attention: The Code

3 Review of Attention Algorithms

3.1 Dot-Product Attention

Simple dot-product attention can be described by the pseudocode below. The logits L are computed as the dot-products of the query-vectors and the memory-vectors. For each query, the logits are passed through a softmax function to produce weights, and the different memory-vectors are averaged together, weighted by those weights. In this code, we show the case where there are n different queries all attending to the same m memory-vectors. If there is only one query, the code is identical except that the "n" dimension is removed from all tensors.

```
def DotProductAttention(
    X[n, d],   # n query-vectors with dimensionality d
    M[m, d]):  # m memory-vectors with dimensionality d
  L[n, m] = einsum(X[n, d], M[m, d])           # Attention logits
  W[n, m] = softmax(L[n, m], reduced_dim=m)    # Attention weights
  Y[n, d] = einsum(W[n, m], M[m, d])
  return Y[n, d]
```

We can see that there are only two operations in play: `einsum` and `softmax`. This is true even in more complicated variations of attention:

Multi-head Attention

```
def MultiHeadAttentionConcise(X, M, P_q, P_k, P_v, P_o):
  L[n, m, h] = einsum(X[n, d_X],
                      M[m, d_M],
                      P_q[d_X, d_k, h],
                      P_k[d_M, d_k, h])
  W[n, m, h] = softmax(L[n, m, h], reduced_dim=m)
  Y[n, d] = einsum(W[n, m, h],
                   M[m, d_M],
                   P_v[d_M, d_v, h],
                   P_o[d_Y, d_v, h])
  return Y[n, d_Y]
```

Roadmap

- **Algorithmic Alignment**
- Einsums as Polynomials
- Symmetry and Species
- Attention and Softmax

NNs are bad computers

As a rule, neural networks are bad at everything computers have traditionally been good at: consistency, mathematical correctness, rule-based reasoning...

In particular, attention-based networks rarely exhibit good "length generalization". No matter how good your dataset of problems is, it will always have *bounded length*.

I can train a transformer—or almost any network architecture—to correctly add, say, 6-digit numbers. But will it be able to add 7-digit numbers? In most cases, no.

In fact, LLMs can't even count.

Transformers need glasses! 👓 Information over-squashing in language tasks

Federico Barbero*
University of Oxford
federico.barbero@cs.ox.ac.uk

Andrea Banino
Google DeepMind
abanino@google.com

Steven Kapturowski
Google DeepMind
skapturowski@google.com

Dharshan Kumaran
Google DeepMind
dkumaran@google.com

João G.M. Araújo
Google DeepMind
joaogui@google.com

Alex Vitvitskyi
Google DeepMind
avlife@google.com

Razvan Pascanu
Google DeepMind
razp@google.com

Petar Veličković
Google DeepMind
petarv@google.com

Abstract

We study how information propagates in decoder-only Transformers, which are the architectural backbone of most existing frontier large language models (LLMs). We rely on a theoretical signal propagation analysis—specifically, we analyse the representations of the last token in the final layer of the Transformer, as this is the representation used for next-token prediction. Our analysis reveals a *representational collapse* phenomenon: we prove that certain distinct sequences of inputs to the Transformer can yield arbitrarily close representations in the final token. This effect is exacerbated by the low-precision floating-point formats frequently used in modern LLMs. As a result, the model is provably unable to respond to these sequences in different ways—leading to errors in, e.g., tasks involving counting or copying. Further, we show that decoder-only Transformer language models can lose sensitivity to specific tokens in the input, which relates to the well-known phenomenon of *over-squashing* in graph neural networks. We provide empirical evidence supporting our claims on contemporary LLMs. Our theory also points to simple solutions towards ameliorating these issues.

1 Introduction

In recent years the field of Natural Language Processing (NLP) has been revolutionised through the introduction of Transformer-based architectures [30]. Large Transformers trained on some version of next-token prediction, known as *Large* Language Models (LLMs), have demonstrated impressive performance across different tasks, including conversational agents [10, 19], understanding multi-modal inputs [1], and code completion [16]. Most contemporary LLMs specifically focus on the decoder part of the original Transformer architecture, and are commonly referred to as *decoder-only* Transformers. Consequently, we focus primarily on such models in this paper.

However, despite the impressive performance of Transformers, recent works have uncovered surprising failures that may point to fundamental issues in their architecture. For instance,

*Work performed while at Google DeepMind.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

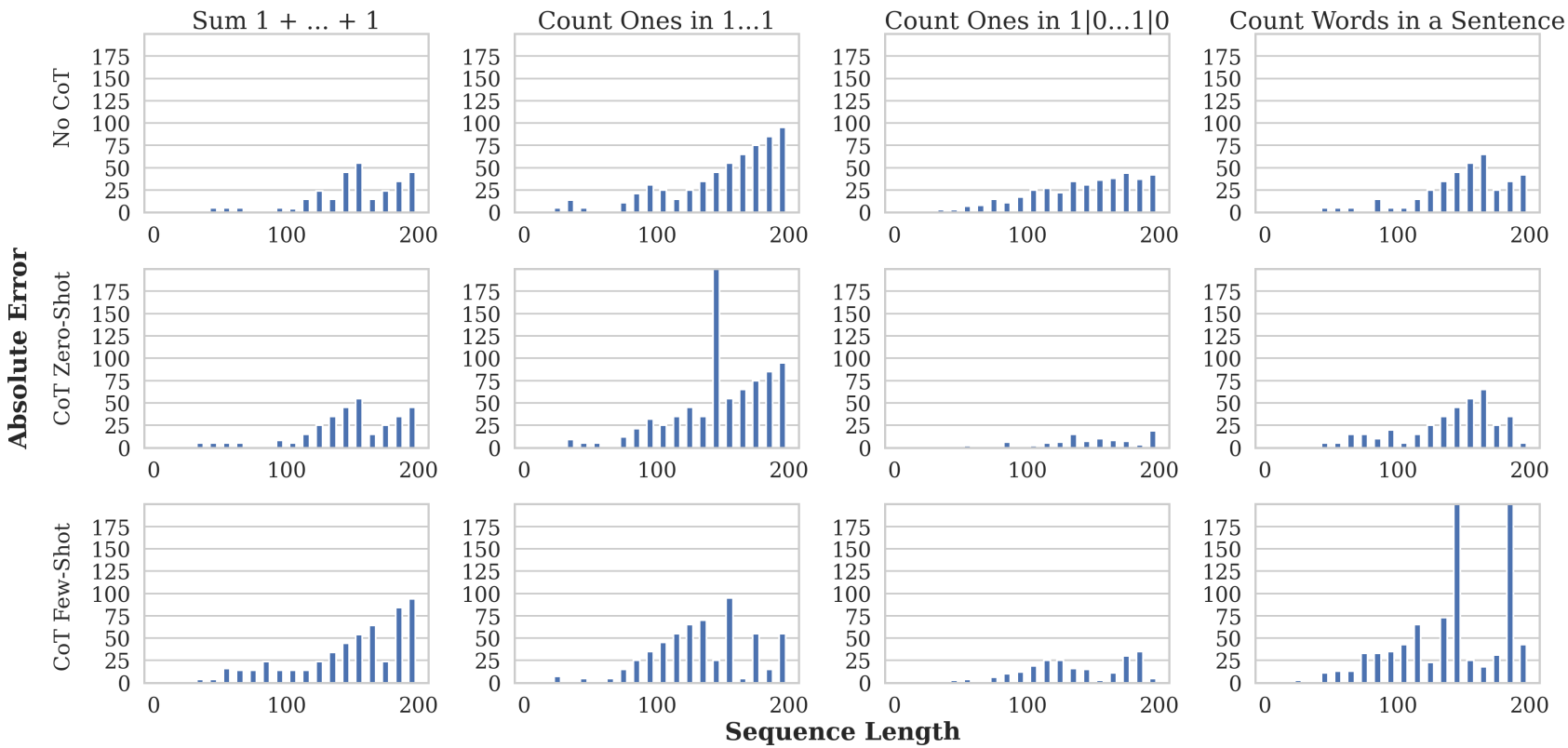


Figure 3: Gemini 1.5 being prompted to sum $1 + \cdots + 1$ (Column 1), Count the number of ones in a sequence of 1s (Column 2), Count the number of ones in a sequence of ones and zeroes (the sequence is a Bernoulli sequence with probability of sampling a one being 0.7) (Column 3), and to counter the number of times a word appears in a sentence (Column 4).

Language Models Use Trigonometry to Do Addition

Subhash Kantamneni [1] **Max Tegmark** [1]

Abstract

Mathematical reasoning is an increasingly important indicator of large language model (LLM) capabilities, yet we lack understanding of how LLMs process even simple mathematical tasks. To address this, we reverse engineer how three mid-sized LLMs compute addition. We first discover that numbers are represented in these LLMs as a generalized helix, which is strongly causally implicated for the tasks of addition and subtraction, and is also causally relevant for integer division, multiplication, and modular arithmetic. We then propose that LLMs compute addition by manipulating this generalized helix using the "Clock" algorithm: to solve $a + b$, the helices for a and b are manipulated to produce the $a + b$ answer helix which is then read out to model logits. We model influential MLP outputs, attention head outputs, and even individual neuron preactivations with these helices and verify our understanding with causal interventions. By demonstrating that LLMs represent numbers on a helix and manipulate this helix to perform addition, we present the first representation-level explanation of an LLM's mathematical capability.

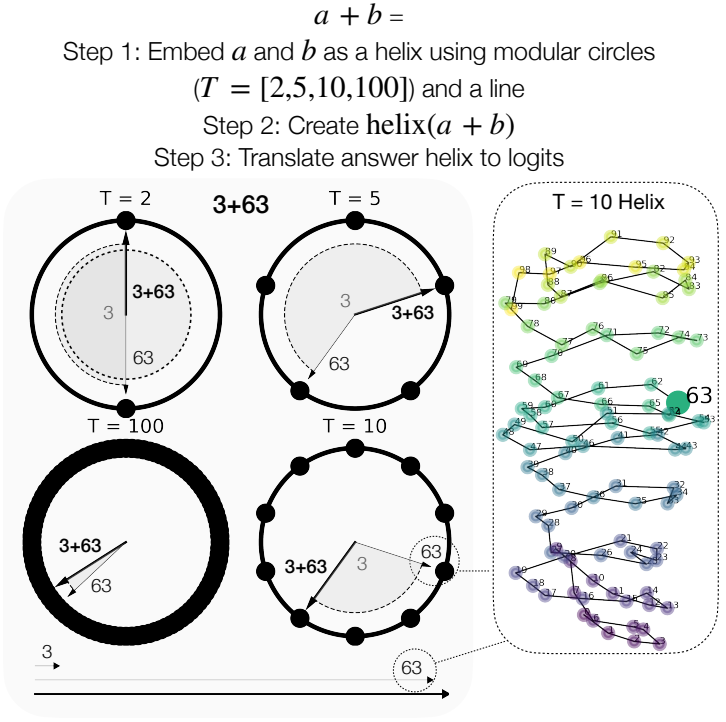


Figure 1. **Illustrating the Clock algorithm.** We find that LLMs represent numbers on a helix. When computing the addition problem $a + b$, LLMs rotate the a and b helices, as if on a clock, to create the $a + b$ helix and read out the final answer.

The CLRS Benchmark

How I got involved: Learning dynamic programming algorithms.

(Veličković et al. 2022) introduced an open-source benchmark for algorithmic tasks, e.g. sorting, pathfinding, knapsack.

They used graphs as a common language to phrase these algorithms in terms of datatypes that could be encoded into, or decoded out of, neural networks.

Remember, the goal isn't just to model a dataset, but to extrapolate correctly.

CLRS

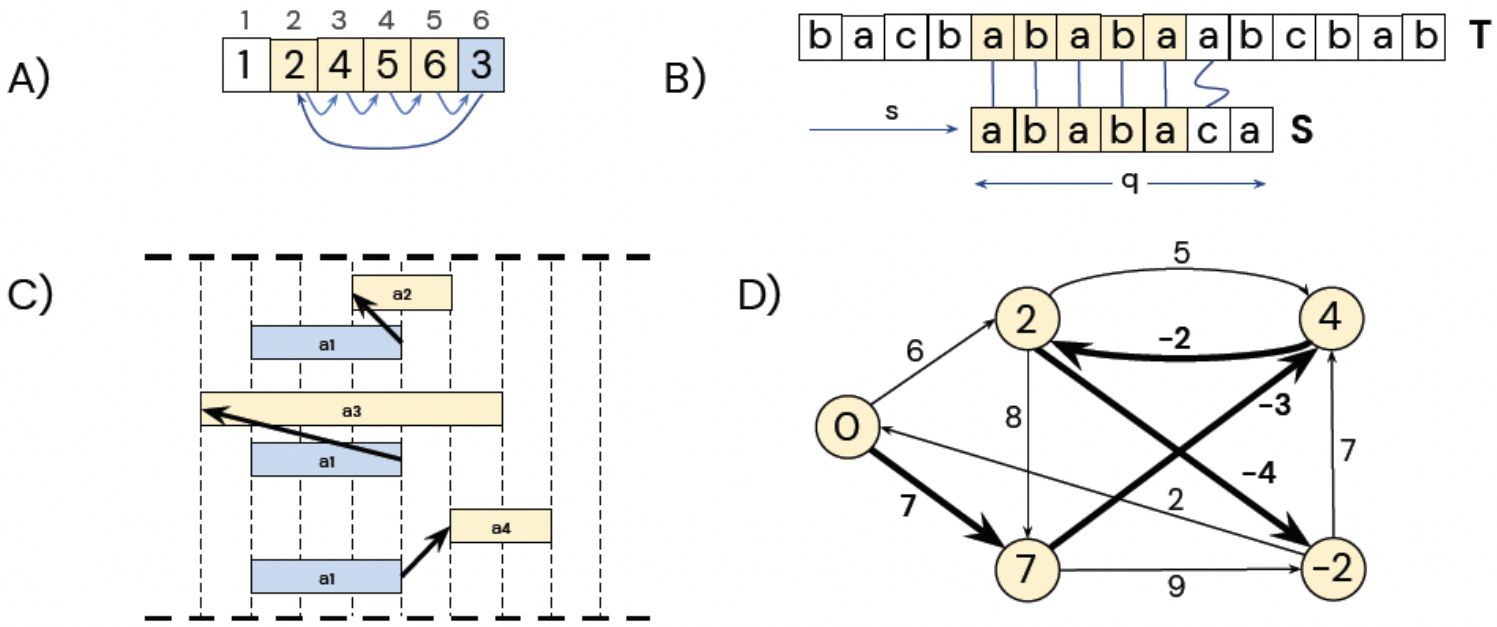


Figure 1. Example of four algorithms within CLRS-30. A) insertion sort; B) string matching; C) greedy task scheduling; D) shortest paths.

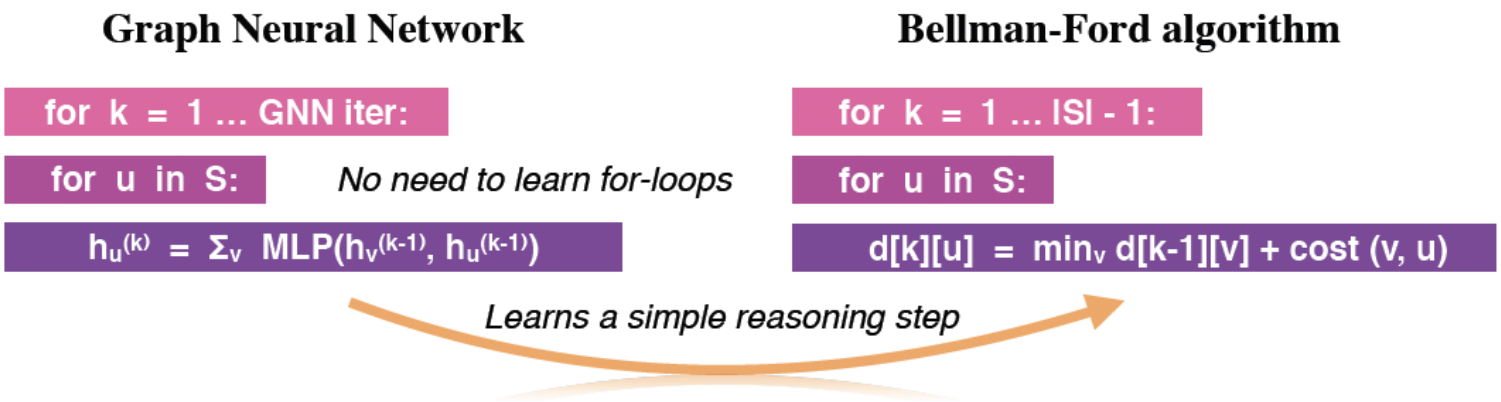
Algorithmic Alignment

Which networks generalize properly?

Previously, (Xu et al. 2019) introduced the idea of "alignment", the obvious-but-not-obvious fact that networks do better at length generalization if they resemble the code they're trying to learn.

This paper was the first clue that there was mathematical structure in algorithms that could inform neural network design.

Algorithmic Alignment



Graph Neural Networks are Dynamic Programmers

Andrew Dudzik[*]
DeepMind
adudzik@deepmind.com

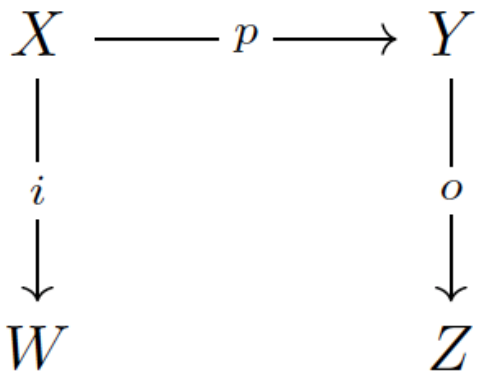
Petar Veličković[*]
DeepMind
petarv@deepmind.com

Abstract

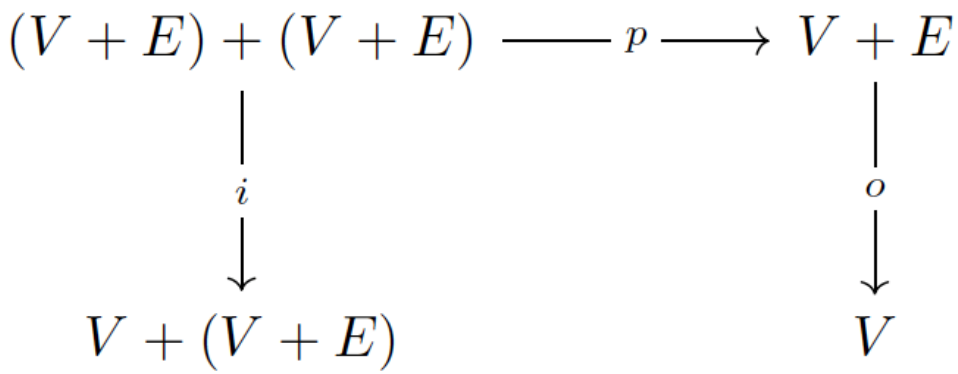
Recent advances in neural algorithmic reasoning with graph neural networks (GNNs) are propped up by the notion of algorithmic alignment. Broadly, a neural network will be better at learning to execute a reasoning task (in terms of sample complexity) if its individual components align well with the target algorithm. Specifically, GNNs are claimed to align with dynamic programming (DP), a general problem-solving strategy which expresses many polynomial-time algorithms. However, has this alignment truly been demonstrated and theoretically quantified? Here we show, using methods from category theory and abstract algebra, that there exists an intricate connection between GNNs and DP, going well beyond the initial observations over individual algorithms such as Bellman-Ford. Exposing this connection, we easily verify several prior findings in the literature, produce better-grounded GNN architectures for edge-centric tasks, and demonstrate empirical results on the CLRS algorithmic reasoning benchmark. We hope our exposition will serve as a foundation for building stronger algorithmically aligned GNNs.

Polynomial Execution Structure

General Execution:

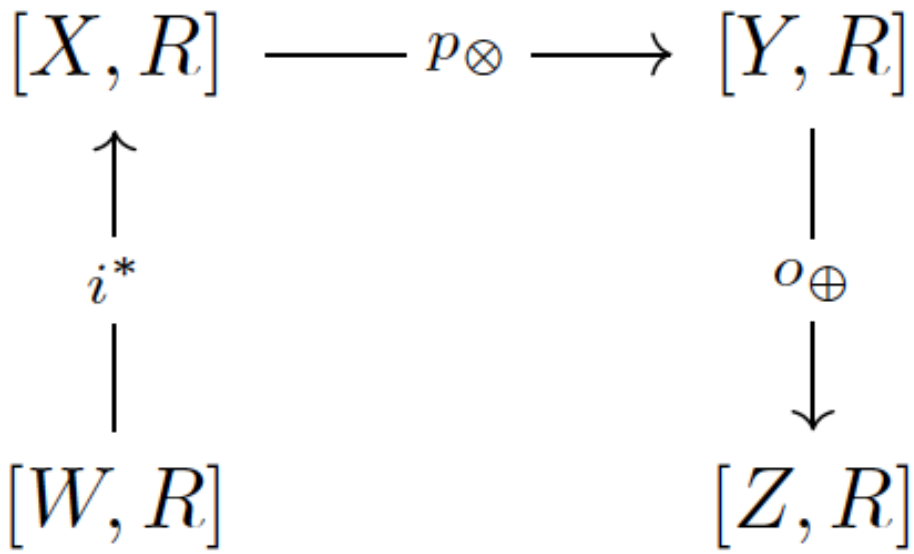


Bellman-Ford:



The Integral Transform

If R is a commutative semiring:



Commutative Semirings

Proposition

The category of finite polynomials is the Lawvere theory for commutative semirings. i.e. every commutative semiring in a monoidal category $(\mathtt{C}, \otimes)$ *is uniquely described by a monoidal functor* $(\mathtt{FinPoly}, +) \to (\mathtt{C}, \otimes)$.

This is the fancy way to say: I can plug numbers into polynomials.

Algebraic Alignment is Algorithmic Alignment

We showed that picking the *correct* semiring was important for generalization.

Classical algorithms typically use some variant of the tropical semiring $(\mathbb{R} \cup \{-\infty\}, +, \max)$, while neural networks, by convention, operate in $(\mathbb{R}, \times, +)$.

Our best algorithmic networks still rely on tropical operations, or smooth approximations of them.

Learnable Commutative Monoids for Graph Neural Networks

Euan Ong
University of Cambridge
elyro2@cam.ac.uk

Petar Veličković
DeepMind / University of Cambridge
petarv@deepmind.com

Abstract

Graph neural networks (GNNs) have been shown to be highly sensitive to the choice of aggregation function. While summing over a node's neighbours can approximate any permutation-invariant function over discrete inputs, Cohen-Karlik et al. [2020] proved there are set-aggregation problems for which summing cannot generalise to unbounded inputs, proposing recurrent neural networks regularised towards permutation-invariance as a more expressive aggregator. We show that these results carry over to the graph domain: GNNs equipped with recurrent aggregators are competitive with state-of-the-art permutation-invariant aggregators, on both synthetic benchmarks and real-world problems. However, despite the benefits of recurrent aggregators, their $O(V)$ depth makes them both difficult to parallelise and harder to train on large graphs. Inspired by the observation that a well-behaved aggregator for a GNN is a commutative monoid over its latent space, we propose a framework for constructing learnable, commutative, associative binary operators. And with this, we construct an aggregator of $O(\log V)$ depth, yielding exponential improvements for both parallelism and dependency length while achieving performance competitive with recurrent aggregators. Based on our empirical observations, our proposed *learnable commutative monoid* (LCM) aggregator represents a favourable tradeoff between efficient and expressive aggregators.

Roadmap

- Algorithmic Alignment
- **Einsums as Polynomials**
- Symmetry and Species
- Attention and Softmax

Example Einsums

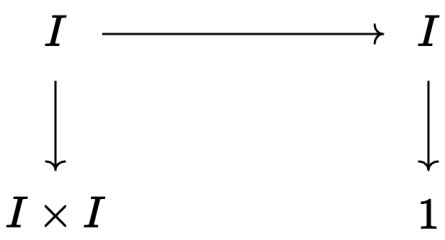
Einsum String	Operation	Equation
`'ii->'`	trace	$\sum_i a_{ii}$
`'ii->i'`	diagonal	a_{ii}
`'i,j->ij'`	outer product	$a_i b_j$
`'ij->j'`	sum over first axis	$\sum_i a_{ij}$
`'ij->ji'`	matrix transpose	a_{ji}
`'ij,jk->ik'`	matrix multiplication	$\sum_j a_{ij} b_{jk}$

Einsums

Einsums represent the bulk of *shape-changing* operations in neural networks.

Einsums as Polynomials

They can be described using polynomials:

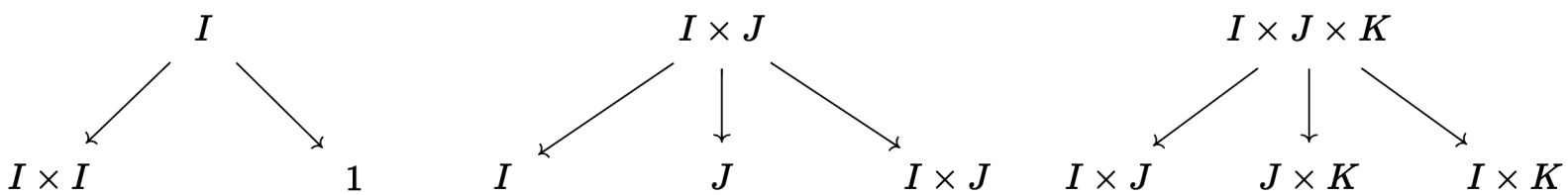


trace

outer product

Einsums as Polynomials

They can also be described more concisely using "parametric spans" (Bergomi, Vertechi, 2022)):



Einsums: Closed Under Gradients

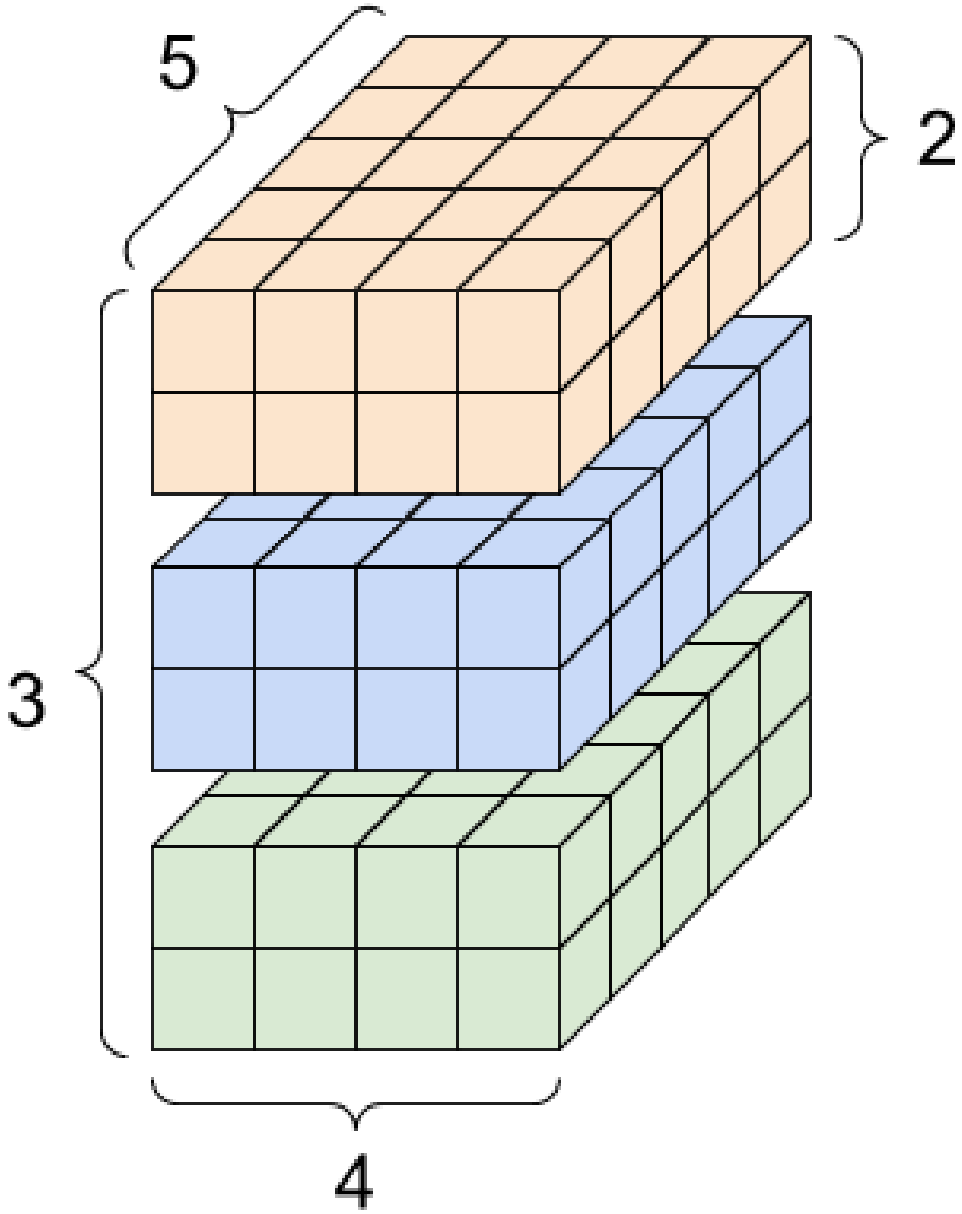
Exercise

The gradient flow through an einsum is an einsum. Hint: Permute the feet!

Roadmap

- Algorithmic Alignment
- Einsums as Polynomials
- **Symmetry and Species**
- Attention and Softmax

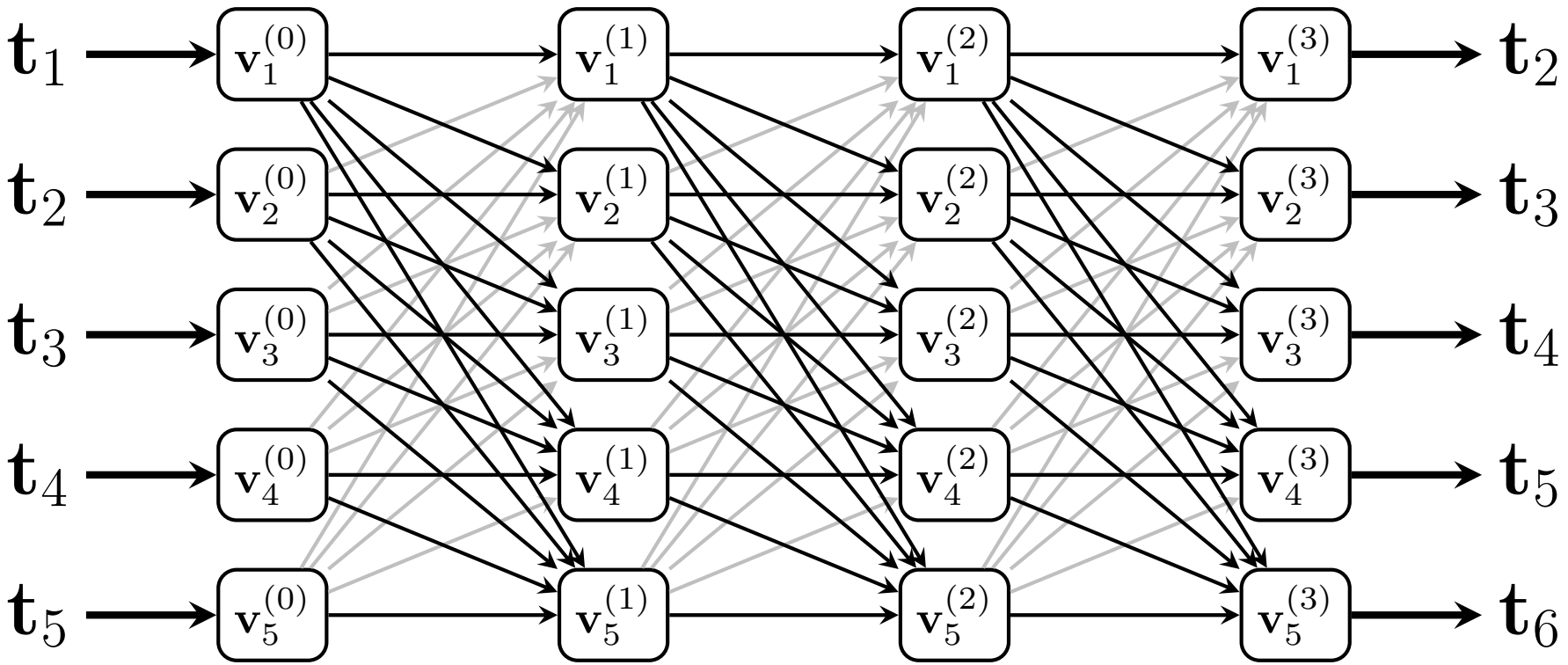
Tensor Sizes are Hyperparameters



The Symmetric Group

The nature of distributing simple computations across hundreds or thousands or cores means that we are always conscious of the symmetric group.

Permutation equivariance is fundamental to performant architectures. Transformers, in particular, rely on symmetry... even when there is none.


$\mathbf{t}_1$
$\mathbf{t}_2$
$\mathbf{t}_3$
$\mathbf{t}_4$
$\mathbf{t}_5$
$\mathbf{v}_1^{(0)}$ $\mathbf{v}_2^{(0)}$ $\mathbf{v}_3^{(0)}$ $\mathbf{v}_4^{(0)}$ $\mathbf{v}_5^{(0)}$
$\mathbf{v}_1^{(1)}$ $\mathbf{v}_2^{(1)}$ $\mathbf{v}_3^{(1)}$ $\mathbf{v}_4^{(1)}$ $\mathbf{v}_5^{(1)}$
$\mathbf{v}_1^{(2)}$ $\mathbf{v}_2^{(2)}$ $\mathbf{v}_3^{(2)}$ $\mathbf{v}_4^{(2)}$ $\mathbf{v}_5^{(2)}$
$\mathbf{v}_1^{(3)}$ $\mathbf{v}_2^{(3)}$ $\mathbf{v}_3^{(3)}$ $\mathbf{v}_4^{(3)}$ $\mathbf{v}_5^{(3)}$
$\mathbf{t}_2$
$\mathbf{t}_3$
$\mathbf{t}_4$
$\mathbf{t}_5$
$\mathbf{t}_6$

LLMs Are Very Symmetric

Only symmetry breaking is the causal mask:

$$+\begin{pmatrix} 0 & 0 & \cdots & 0 \\ -\infty & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ -\infty & -\infty & \cdots & 0 \end{pmatrix}$$

We have to inject positional encodings to reconstruct the ordering, it's not respected in the computation! Yet this is state of the art.

Even though tokens come in a list, it's better to treat them as a bag.

Some Other Work

Deep Sets (Zaheer et al., 2017) emphasized the importance, in Graph Neural Networks, of invariance with respect to the symmetric group.

Natural Graph Networks (de Haan, Cohen, Welling, 2020) emphasized the importance of equivariance with respect to graph isomorphisms, by looking at functors out of the groupoid of graphs.

Set Species

Definition

A (finite) **set species** is a functor $A : \mathtt{core}(\mathtt{FinSet}) \to \mathtt{FinSet}$.

Equivalently, it is a sequence of finite sets $A_0, A_1, \ldots$, together with, for each n, an action of the symmetric group S_n on A_n.

Example

Let A_n be the set of cyclic orderings of $\{1, \ldots, n\}$. We have an action of S_n on A_n given by
$\pi \cdot (a_1\ a_2 \ldots a_n) = (\pi(a_1)\ \pi(a_2) \ldots \pi(a_n))$, making $A_0, A_1, \ldots$ into a set species.

Vector Species

Definition

A (finite, real) **vector species** is a functor $V : \mathtt{core}(\mathtt{FinSet}) \to \mathtt{FinVect}$.

Equivalently, it is a sequence of finite-dimensional real vector spaces $V_0, V_1, \ldots$, together with, for each n, a linear action of the symmetric group S_n on V_n.

Multivariate species

We also want species in d variables, i.e.

$$\mathtt{core}(\mathtt{FinSet})^d \to \mathtt{FinSet}$$

$$\mathtt{core}(\mathtt{FinSet})^d \to \mathtt{FinVect}$$

We can use these to better formalize einsums. Consider the 3-variable species $I(a, b, c) = a$, $J(a, b, c) = b$, and $K(a, b, c) = c$.

Matmul: A Polynomial in 3-variable Species

```
einsum('ij,jk->ik')
```

$$\boldsymbol{I} \times \boldsymbol{J} \times \boldsymbol{K} + \boldsymbol{I} \times \boldsymbol{J} \times \boldsymbol{K} \longrightarrow \boldsymbol{I} \times \boldsymbol{J} \times \boldsymbol{K}$$

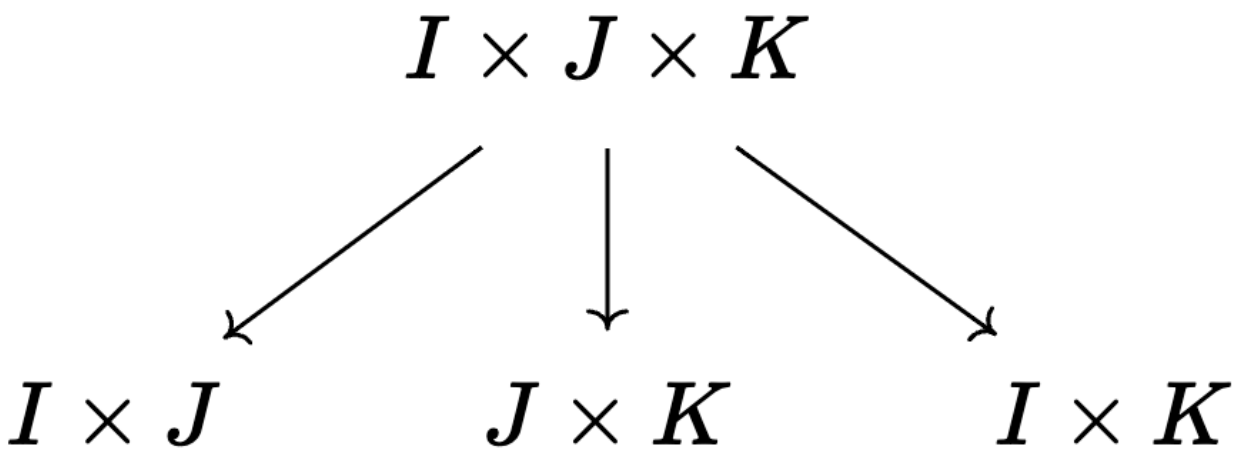
$$\downarrow \qquad\qquad \downarrow$$

$$\boldsymbol{I} \times \boldsymbol{J} + \boldsymbol{J} \times \boldsymbol{K} \qquad\qquad \boldsymbol{I} \times \boldsymbol{K}$$

$$(\mathbb{R}^I \otimes \mathbb{R}^J) \oplus (\mathbb{R}^J \otimes \mathbb{R}^K) \rightsquigarrow \mathbb{R}^I \otimes \mathbb{R}^K$$

Matmul: A Trivalent Span in 3-variable Species

```
einsum('ij,jk->ik')
```



$$(\mathbb{R}^I \otimes \mathbb{R}^J) \oplus (\mathbb{R}^J \otimes \mathbb{R}^K) \rightsquigarrow \mathbb{R}^I \otimes \mathbb{R}^K$$

Decategorifying Species

Definition

If $A_0, A_1, \ldots$ is a set species, and each A_n is finite, we define the **generating function associated to** A as follows:

$$(\#A)(x) := \sum_{n \geq 0} \frac{\#A_n}{n!} x^n$$

Generating functions are nice to have around as a "sanity check". Interesting properties of GFs usually mean interesting properties of species.

The binomial theorem:

$$e^{x+y} = e^x e^y$$

Species and polynomials

The generating function is related to the groupoid cardinality of the analytic functor $\mathtt{Set} \to \mathtt{Set}$:

$$\hat{A}(X) := \sum_{n \geq 0} \frac{A_n \times X^n}{S_n}$$

N.B. this correspondence gives an identification of polynomial functors with species where each group action is free.

Some Basic Datatypes

Counting Problem	Species	Generating Function
No Data	$\mathbb{E}$	e^x
Is Empty	1	1
Is Nonempty	$\mathbb{E} - 1$	$e^x - 1$
Is Singleton	$\mathbb{X}$	x
Pick One	$\mathbb{X} \cdot \mathbb{E}$	xe^x
Pick Two	$(1 + \mathbb{X}) \cdot \mathbb{X} \cdot \mathbb{E}$	$(1 + x)xe^x$
Pick Two (unique)	$\mathbb{X}^2 \cdot \mathbb{E}$	$x^2 e^x$
Pick a Total Order	$\mathbb{L}$	$\frac{1}{1-x}$
Pick a Partition	$\mathbb{E} \circ (\mathbb{E} - 1)$	$e^{e^x - 1}$
Pick a Graph	$\mathbb{G}$	nonconvergent

Some Basic Datatypes

Datatype	Memory (Discrete)	Memory (Vectorized)
Pick One	$\log n$	$c \cdot n$
Pick Two	$2 \log n$	$c \cdot n^2$
Pick a Total Order	$n \log n$	$c \cdot n!$
Pick a Graph	$\log 2 \cdot \binom{n}{2}$	$c \cdot 2^{\binom{n}{2}}$

The Cauchy Product

Cauchy product: $f(x), g(x) \mapsto f(x)g(x)$
This corresponds to dividing a set in two pieces, and giving the first structure to the first piece, and the second structure to the second piece.

Example

How many ways can I divide a set into two pieces, such that the first piece is nonempty, and the second piece has exactly one element?

Answer

"This set is nonempty" is described by $e^x - 1$, *"this set has one element" is described by* x, *so the answer is* $(e^x - 1) \cdot x$.

The Cauchy Product

$$(F \cdot G)[n] := \sum_{k \coprod l = n} F[k] \times G[l]$$

$$= \sum_{k+l=n} \binom{n}{k} F[k] \times G[l]$$

Cauchy Bimonoids

Notably, it is often the case that "traversable" datatypes are Cauchy bimonoids. For example, the species of graphs can be given a simple monoid structure sending two graphs to their disjoint union:

$$\mathbb{G}[S] \times \mathbb{G}[T] \to \mathbb{G}[S + T]$$

... and a simple comonoid structure sending a graph to its restrictions along two complementary subsets:

$$\mathbb{G}[S + T] \to \mathbb{G}[S] \times \mathbb{G}[T]$$

Proposition

A Cauchy bimonoid in vector species with $V_0 = \mathbb{R}$ is Hopf, i.e. has an antipode.

The Substitution Product

Substitution product: $f(x), g(x) \mapsto f(g(x))$

Example

How many ways can I form a nonempty partition of a set into nonempty sets?

Answer

"This set is nonempty" is described by $e^x - 1$*, so we compose this with itself to get* $e^{e^x - 1} - 1$*.*

The Substitution Product

$$(F \circ G)[n] := \sum_{\lambda \vdash n} F[|\lambda|] \times \prod_{i \in \lambda} G[i]$$

Operads

A monoid with respect to the substitution product is called a (symmetric) *operad*.

Example

Let $\Delta(n)$ be the set of probability distributions on $\{1, \ldots, n\}$. Δ is an operad with respect to the composition law given by forming joint distributions:

$$\Delta(k) \times \Delta(n_1) \times \cdots \times \Delta(n_k) \to \Delta(n_1 + \cdots + n_k)$$

$$(p_i)_i, (q_{1j})_j, \ldots, (q_{kj})_j \mapsto (p_i q_{ij})_{ij}$$

Δ is called the *simplicial operad*, and it will come up a bit later.

Much simpler example: monoids

If M is a monoid in sets, we can upgrade M to an operad by identifying it with the species $M[1] = M$, $M[k] = 0$ for $k \neq 1$.

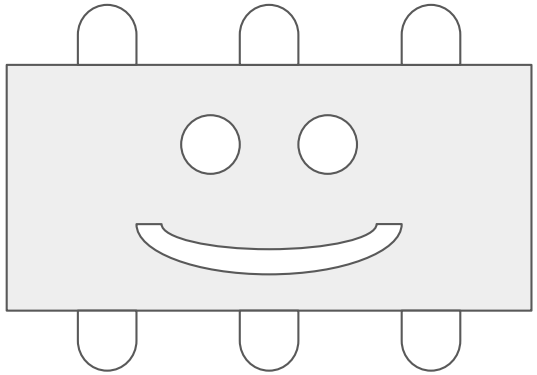
i.e. monoids (in particular, groups) are operads where every operation has exactly one output.

Roadmap

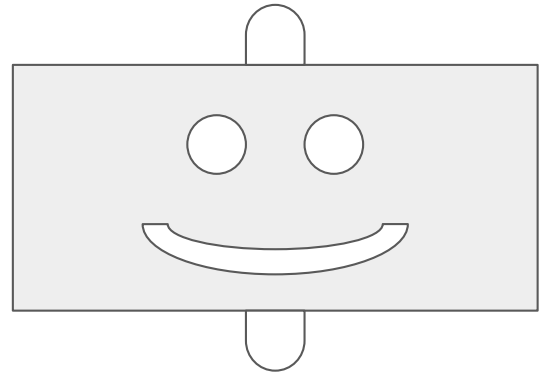
- Algorithmic Alignment
- Einsums as Polynomials
- Symmetry and Species
- **Attention and Softmax**

Nancy Pays Attention to Martha

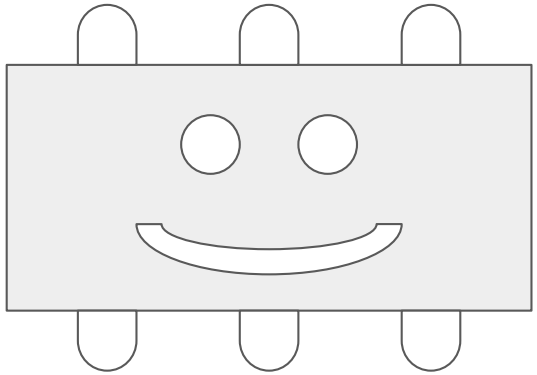
a story of neural cooperation



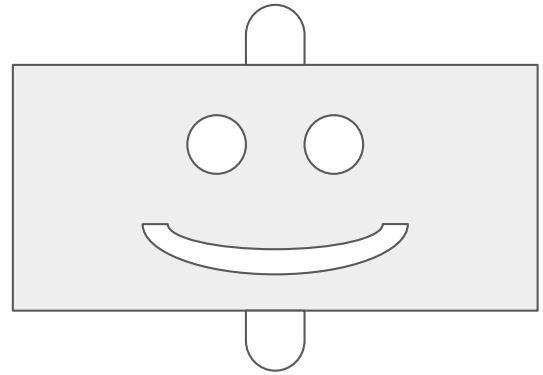
Martha (m=3)



Nancy (n=1)

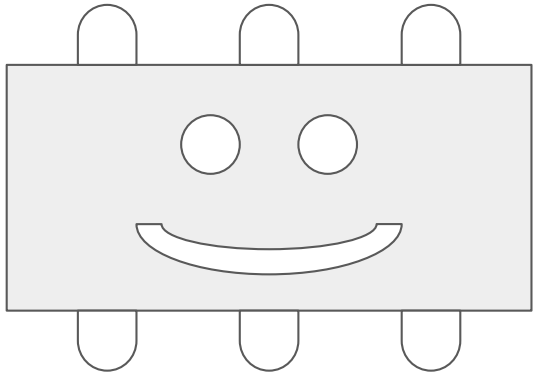


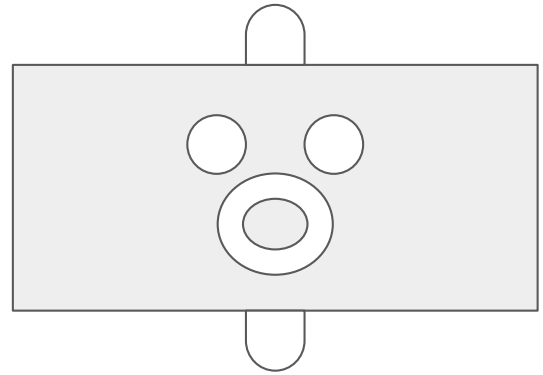
Martha (m=3)



Nancy (n=1)

1

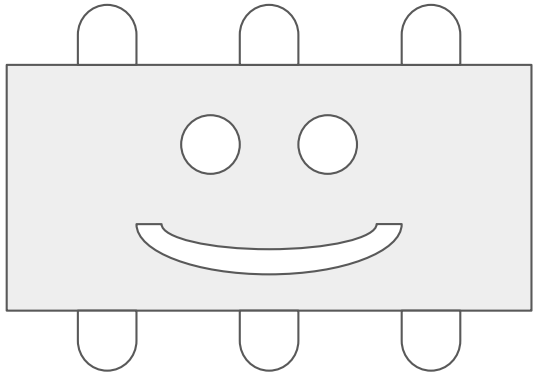


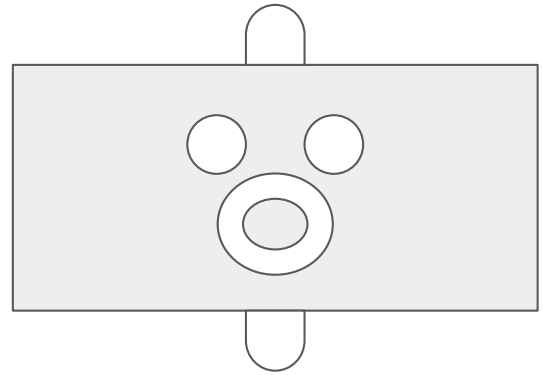


Martha (m=3)

Nancy (n=1)

1
0

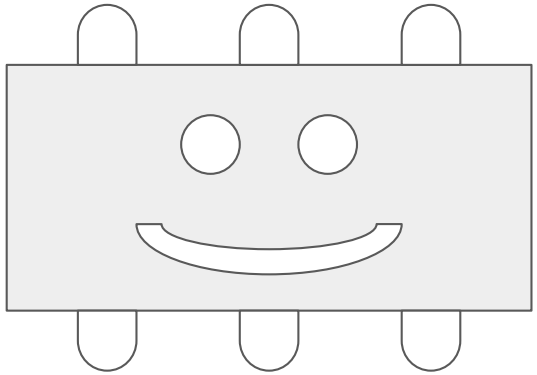


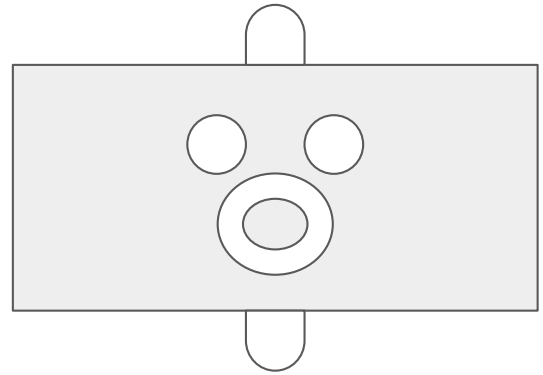


Martha (m=3)

Nancy (n=1)

1
0
1

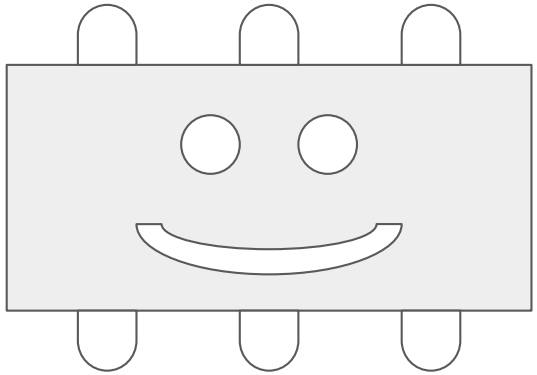


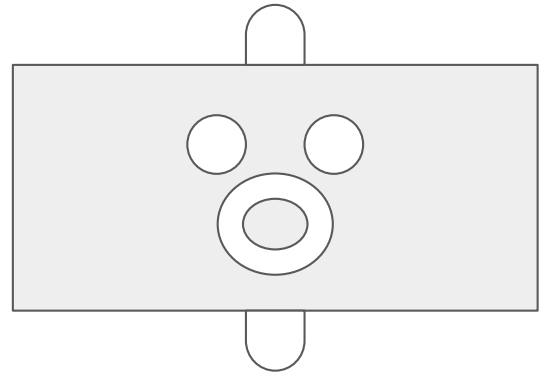


Martha (m=3)

Nancy (n=1)

1
0
1
1

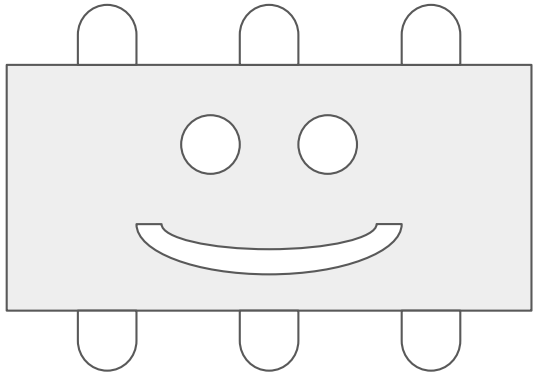


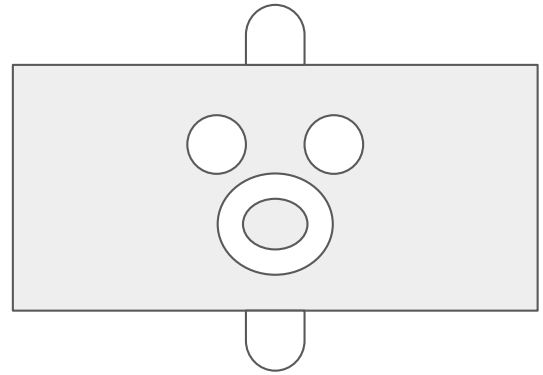


Martha (m=3)

Nancy (n=1)

1
0
1
1
0



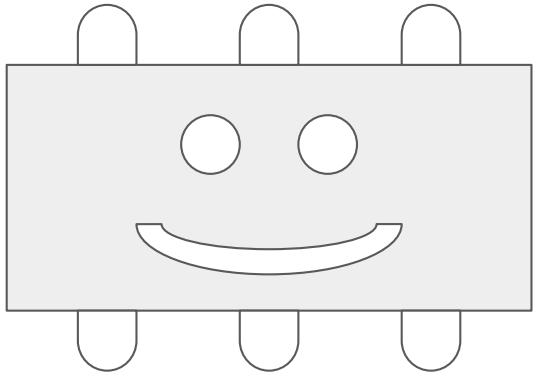


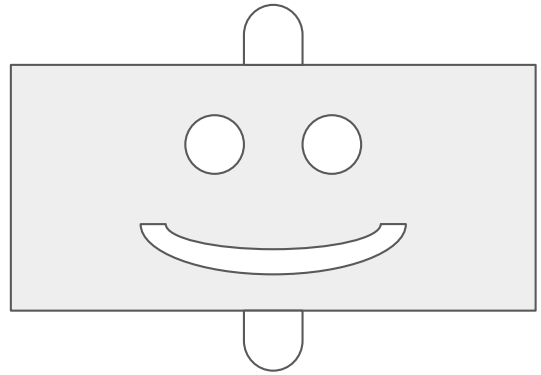
Martha (m=3)

Nancy (n=1)

Q

1
0
1
1
0

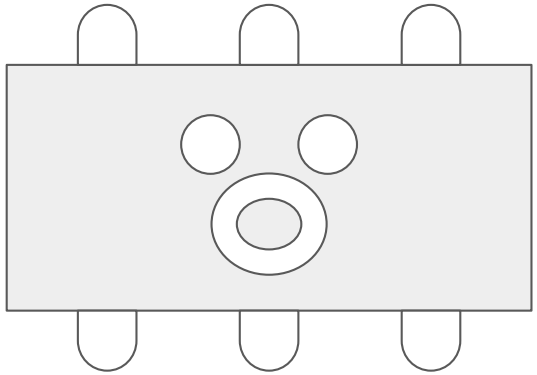




Martha (m=3)

Nancy (n=1)

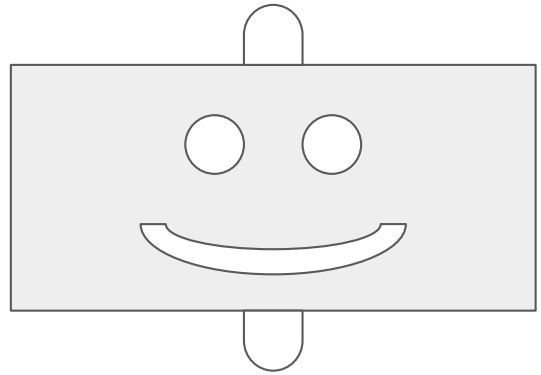
1 1 0



Martha (m=3)

Q

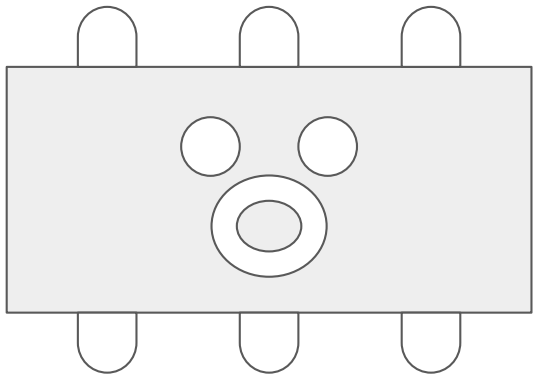
1
0
1
1
0



Nancy (n=1)

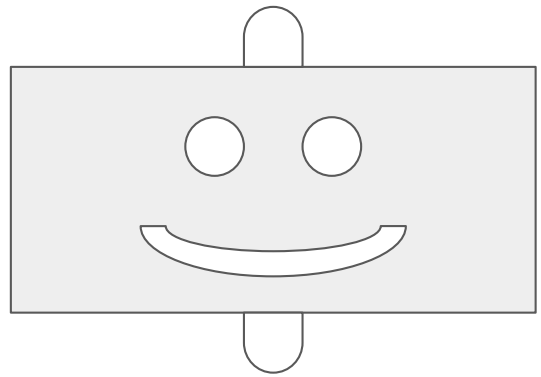
Q

1 1 0
1 0 1



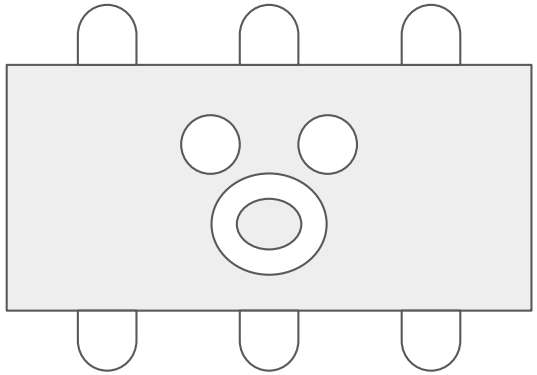
Martha (m=3)

1
0
1
1
0



Nancy (n=1)

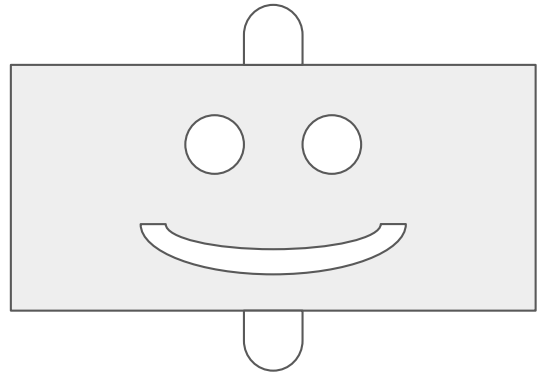
1 1 0
1 0 1
0 1 1



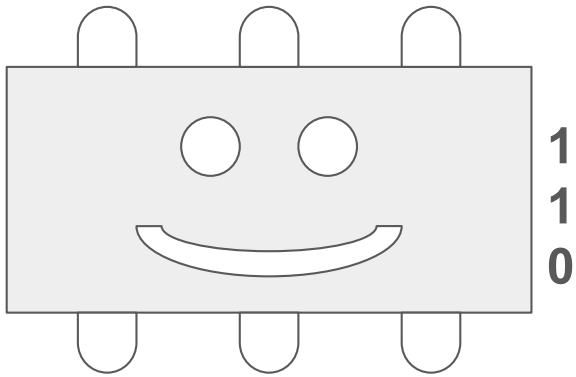
Martha (m=3)

Q

1
0
1
1
0



Nancy (n=1)



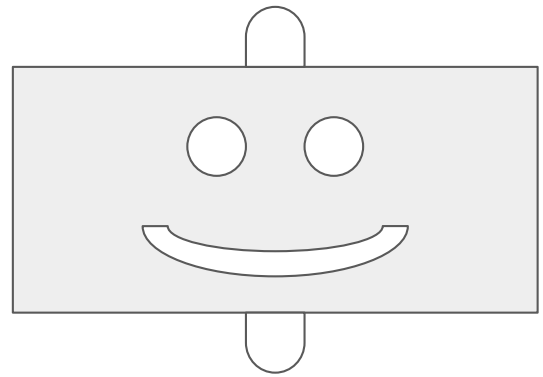
1
1
0

V

1
1
0
1

Q

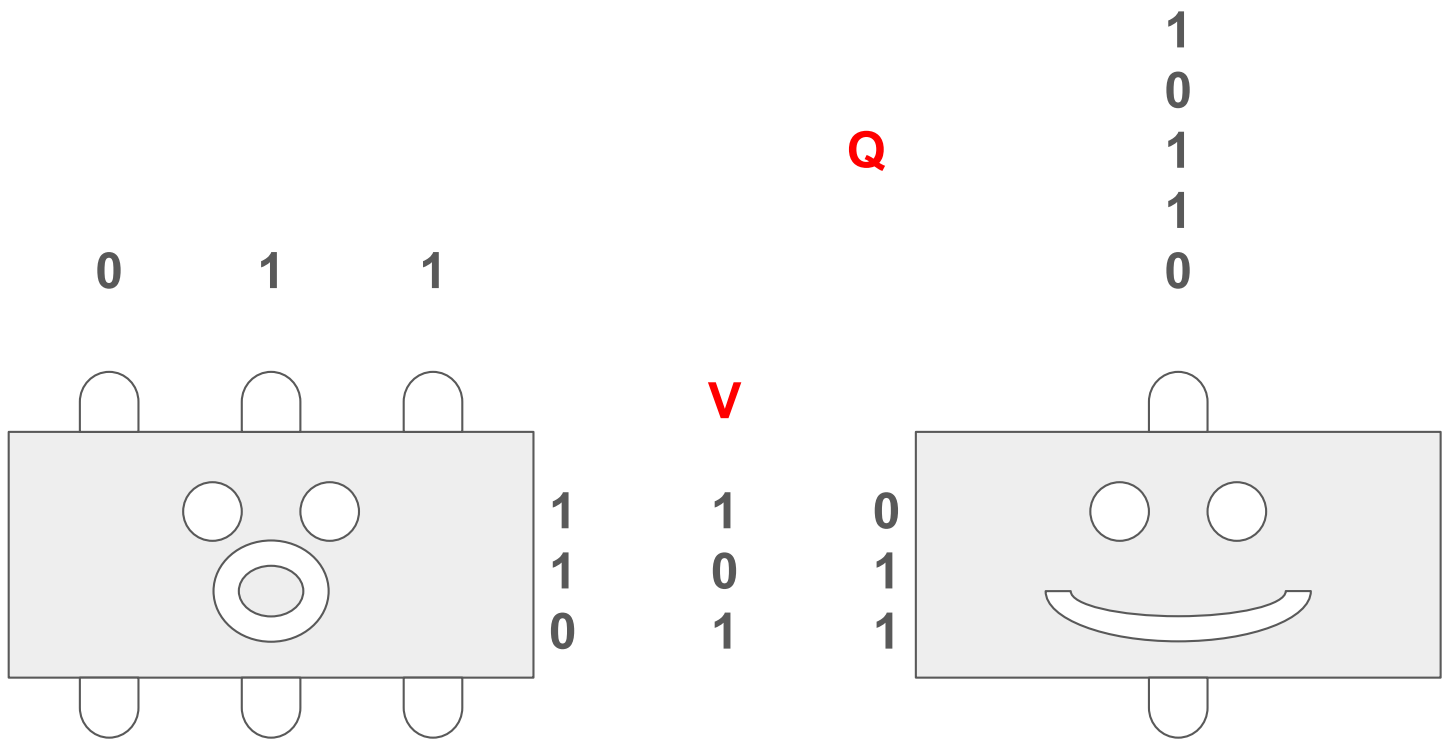
1
0
1
1
0

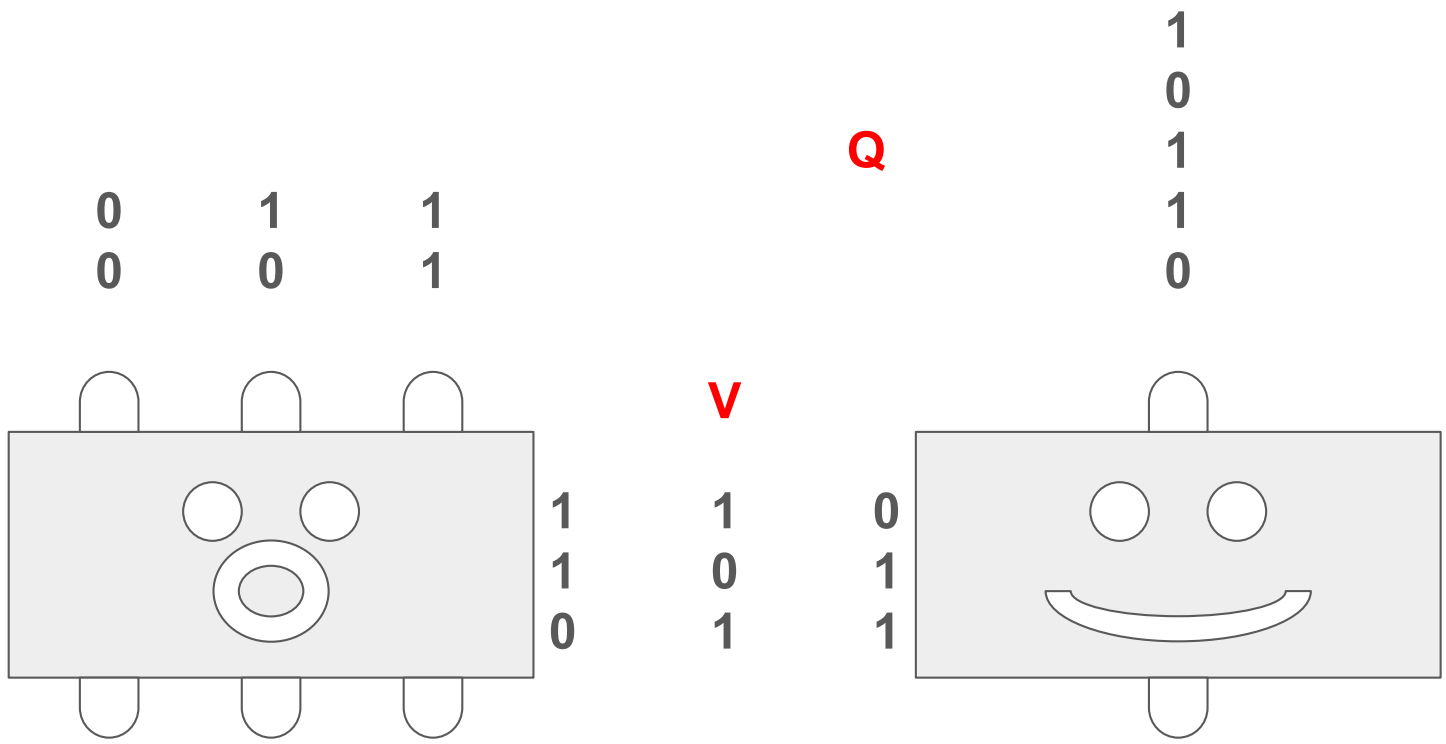


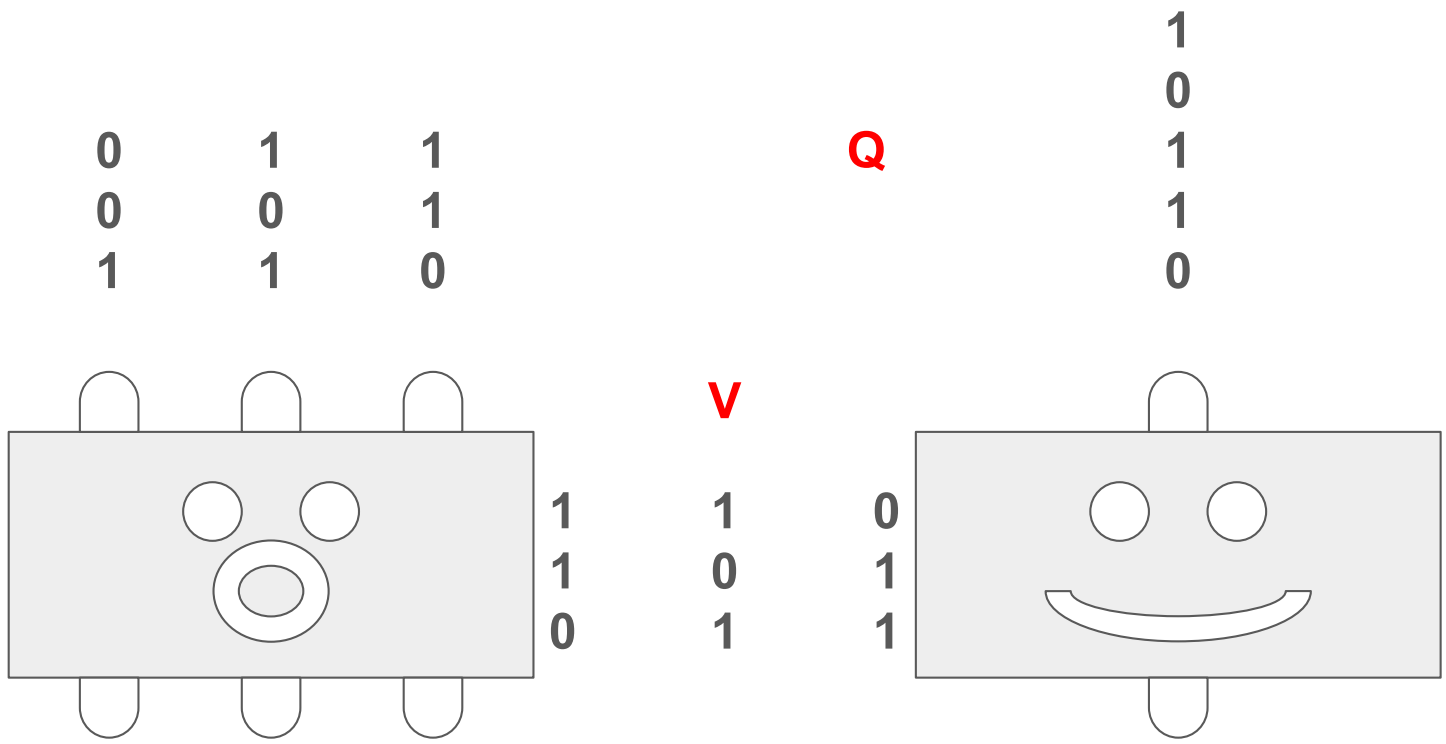
0
1
1

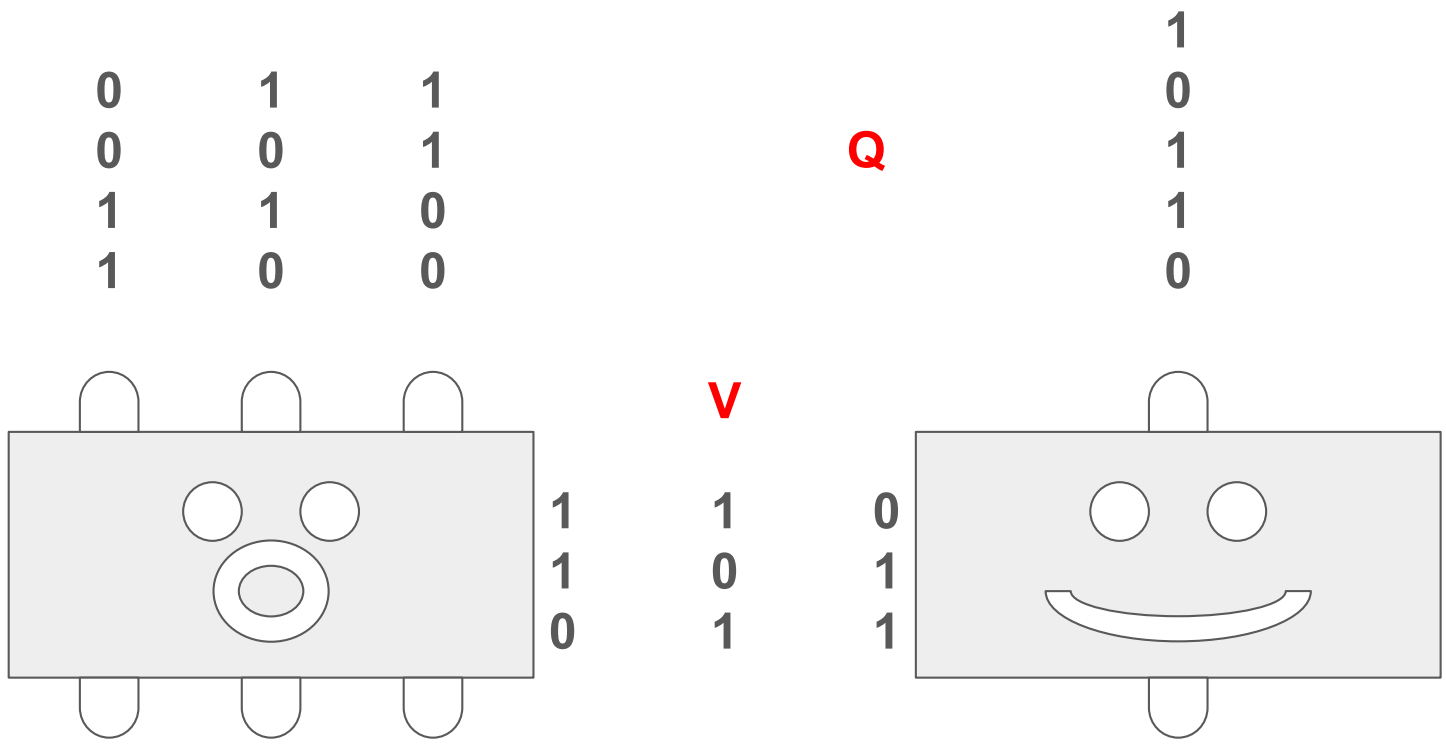
Martha (m=3)

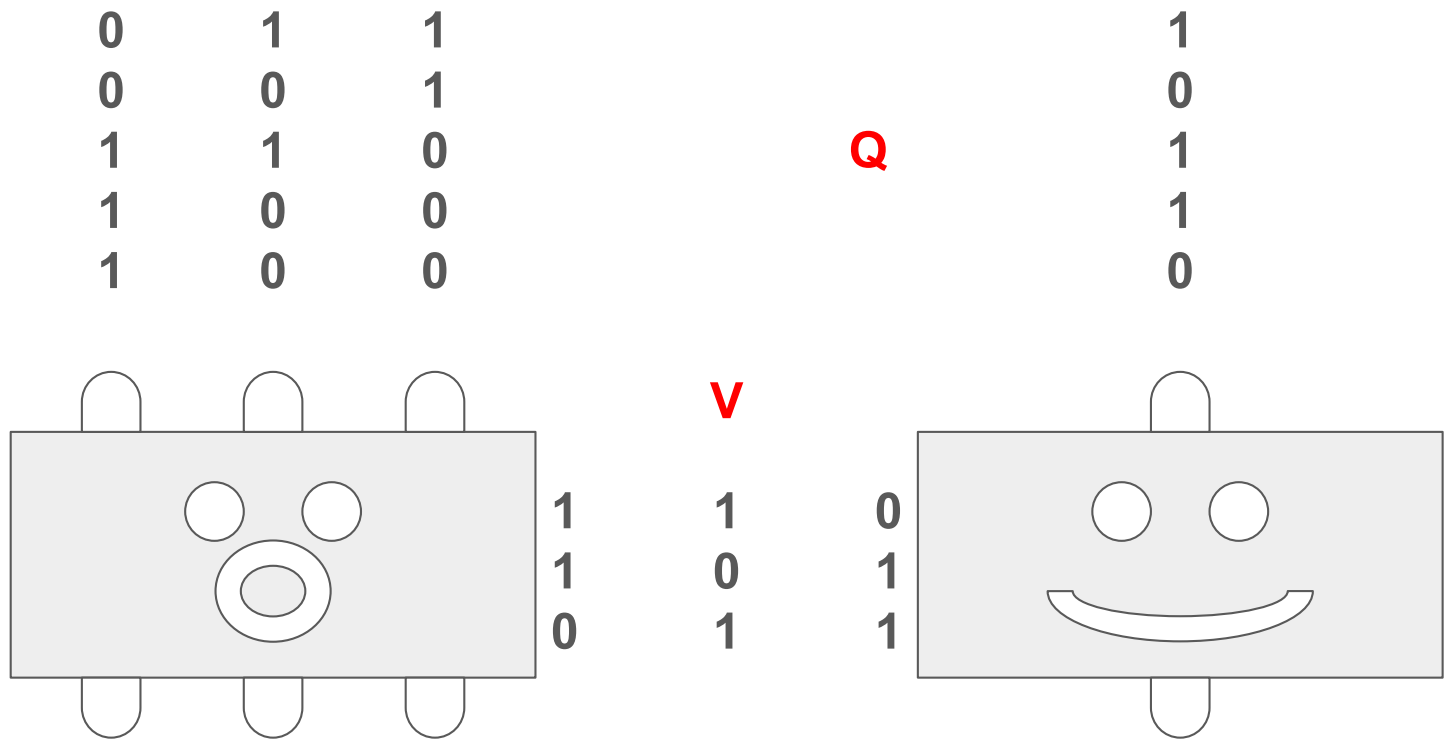
Nancy (n=1)

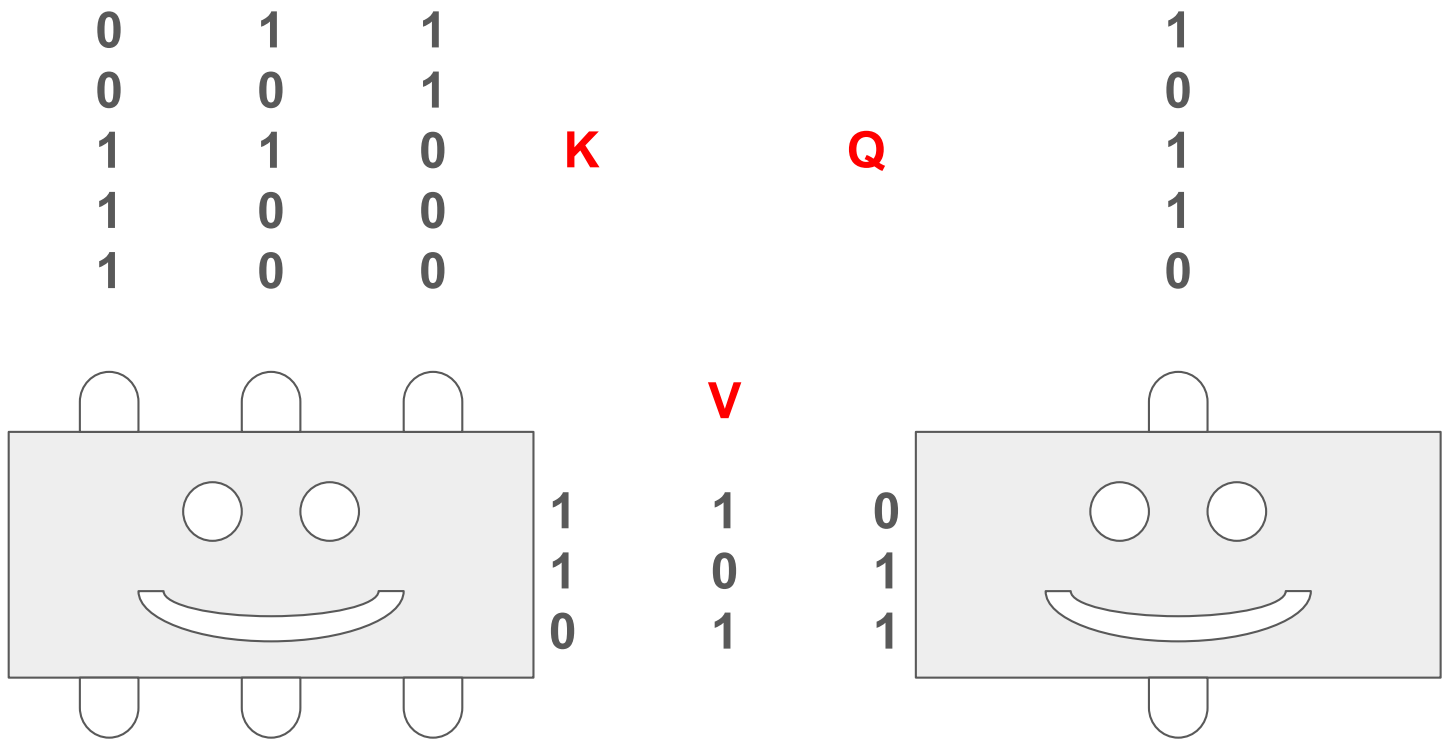

Q
1
0
1
1
0
0 1 1
V
1
1
0
1
1
0
1
1
0
Martha (m=3)
Nancy (n=1)

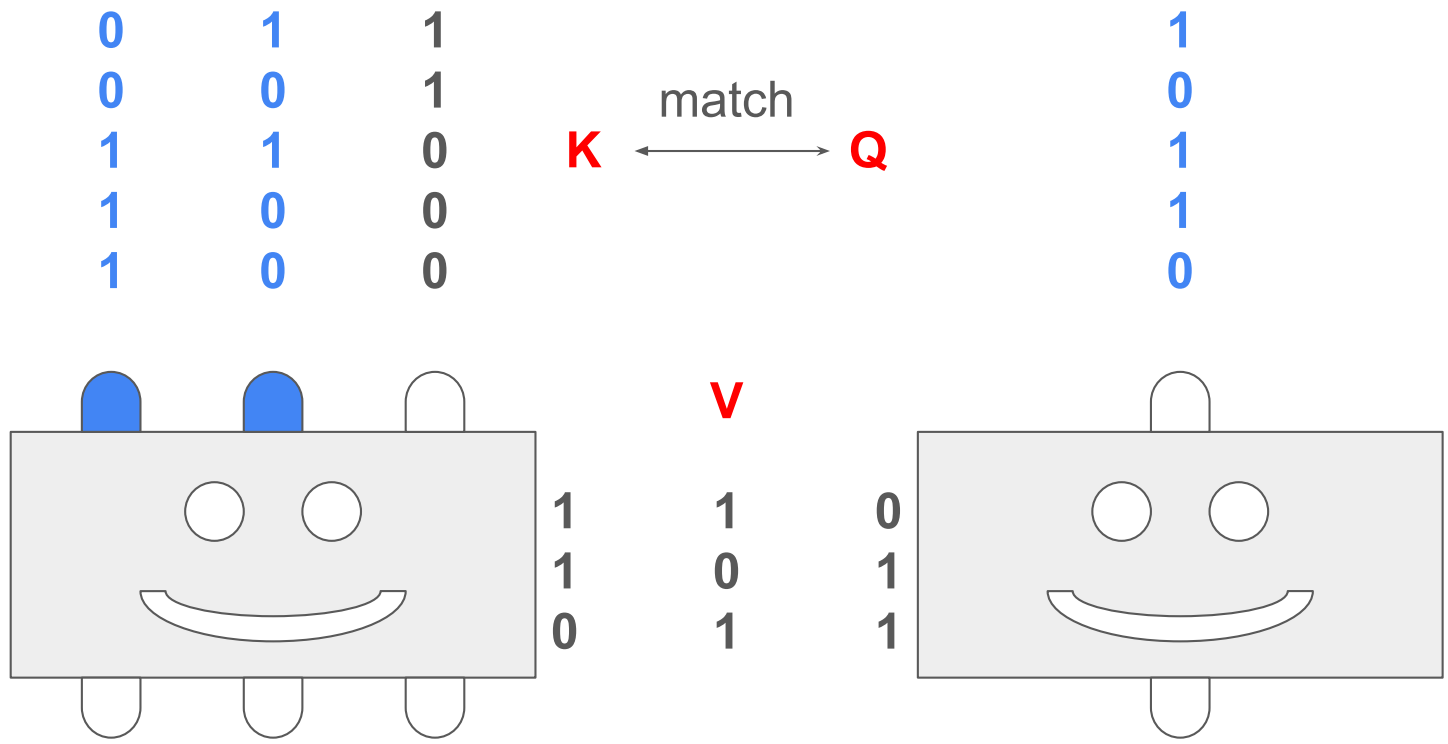

Q
1
0
1
1
0
0 1 1
0 0 1
V
1
1
0
1
1 1 0
0 1 1
Martha (m=3)
Nancy (n=1)

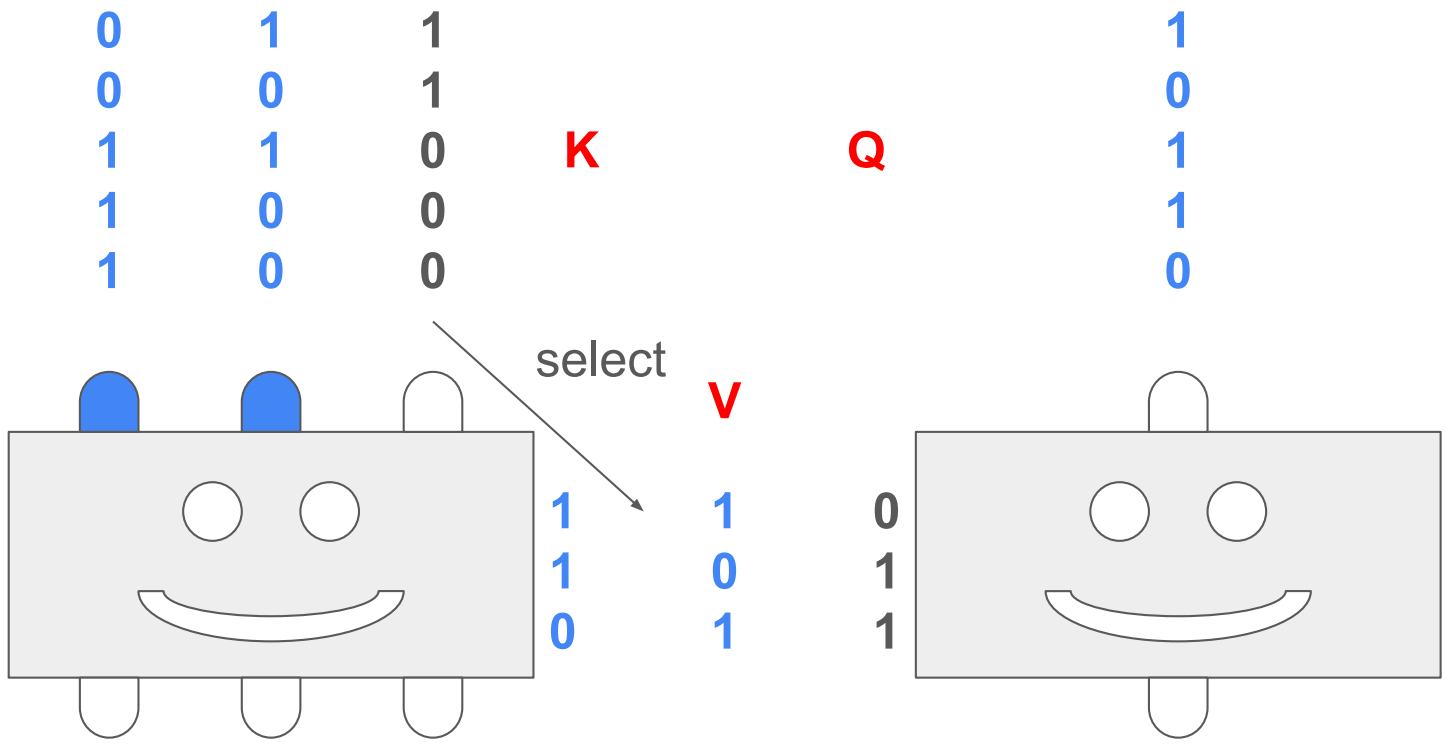

0 1 1
0 0 1
1 1 0
1
1
0
V
1 1
0 1
1 1
Q
1
0
1
1
0
Martha (m=3)
Nancy (n=1)

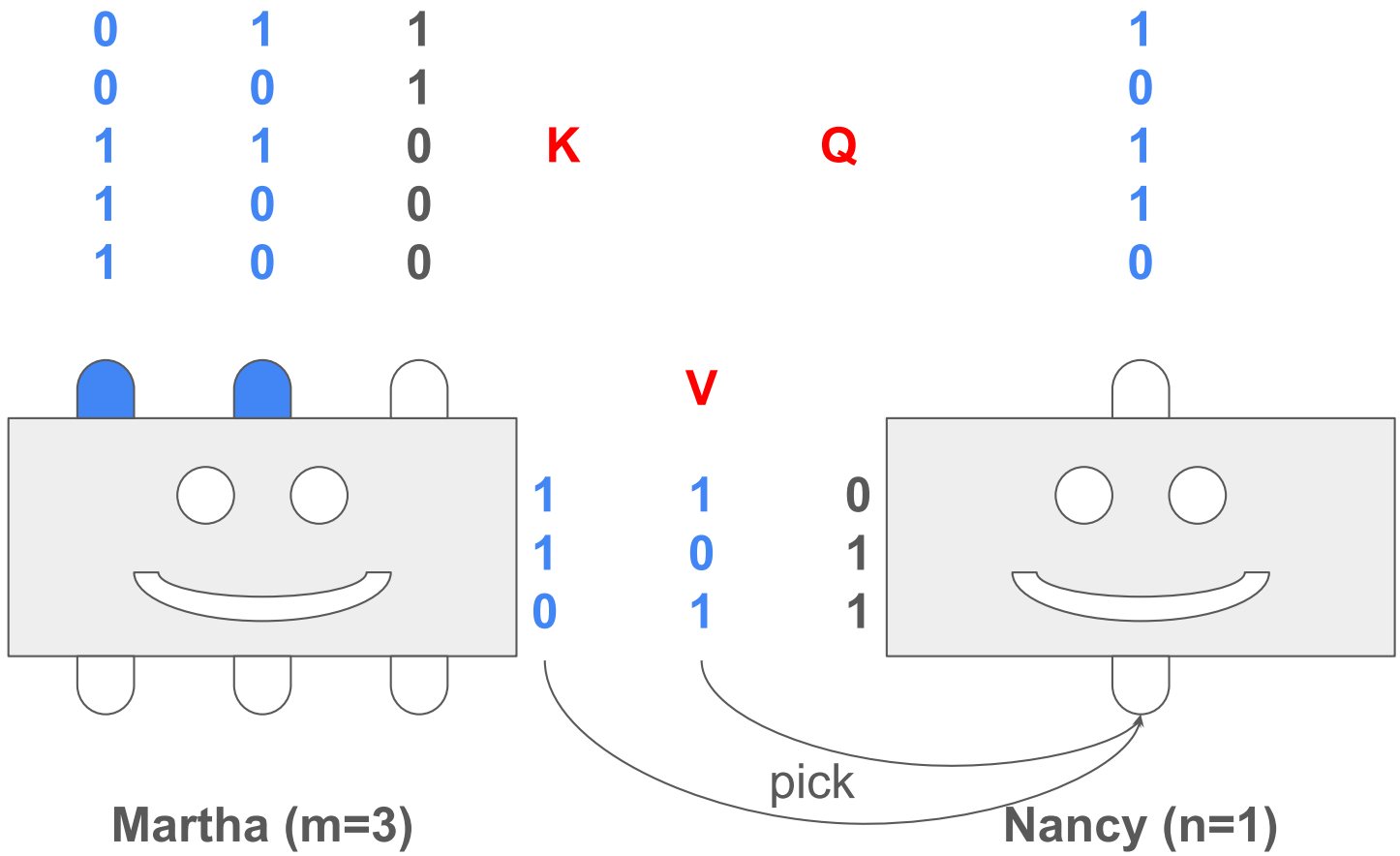

0 1 1
0 0 1
1 1 0
1 0 0
Q
1
0
1
1
0
V
1 1
1 0
0 1
1 1
0 1
Martha (m=3)
Nancy (n=1)


0 1 1
0 0 1
1 1 0
1 0 0
1 0 0
Q
1
0
1
1
0
V
1 1
1 0
0 1
1 1
0 1
1 0
Martha (m=3)
Nancy (n=1)


0 1 1
0 0 1
1 1 0
1 0 0
1 0 0
K
Q
1
0
1
1
0
V
1 1
1 0
0 1
0 1
1 1
Martha (m=3)
Nancy (n=1)


0 1 1
0 0 1
1 1 0
1 0 0
1 0 0
match
K
Q
1
0
1
1
0
V
1 1
1 0
0 1
0 1
1 1
Martha (m=3)
Nancy (n=1)


0 1 1
0 0 1
1 1 0
1 0 0
1 0 0
K
Q
1
0
1
1
0
select
V
1 1
1 0
0 1
0
1
1
Martha (m=3)
Nancy (n=1)


0 1 1
0 0 1
1 1 0
1 0 0
1 0 0
K
Q
1
0
1
1
0
V
1 1
1 0
0 1
0 1
1 1
pick
Martha (m=3)
Nancy (n=1)

Three forms of picking

Since we cannot eliminate the possibility of multiple matches, a "picking" strategy is needed. There are three basic strategies:

- Introduce a tie-breaking mechanism. This could be a predetermined order on Martha's outputs, or somehow inferred from the query and keys. Many dynamic programming algorithms depend on handling tie-breaking consistently.
- Sample from a probability distribution. This is elegant but introduces non-determinism.
- Take the expected value over a probability distribution. This is the preferred method in the vectorized setting, but is usually nonsensical in classical programs.

Attention in shapes

Values: $m \times l$

Keys: $m \times k$

Queries: $n \times k$

Matching: $(n \times k) \times (k \times m) \to n \times m$

Selection: $n \times m \to n \times m$

Sampling: $(n \times m) \times (m \times l) \to n \times l$

Note that we are conjugating the selection function by a shape change.

The Standard Selector: softmax

The textbook way to produce a probability distribution from a vector:

$$\text{softmax}(L_1, \ldots, L_n) := \left(\frac{e^{L_1}}{\sum_j e^{L_j}}, \ldots, \frac{e^{L_n}}{\sum_j e^{L_j}}\right)$$

Hot and cold softmax

We generally scale the softmax inputs by a thermodynamic parameter $\beta > 0$, the "inverse temperature". This is exactly the Boltzmann distribution, if you think of the L_i as negative energy.

Note the behavior for special values of β:

	$\text{softmax}(\beta L_i)$
$\beta \to \infty$	argmax (almost)
$\beta = 0$	uniform
$\beta \to -\infty$	argmin (almost)

Masked softmax

Probability distributions are often masked, by replacing logits with $-\infty$, guaranteeing that the associated probability is zero.

In this setting, we can extend softmax to a surjective function:

$$\text{softmax} : [-\infty, \infty)^n \setminus \{(-\infty, \ldots, -\infty)\} \to \Delta(n)$$

We have $\text{softmax}(L_i) = \text{softmax}(L'_i)$ if and only if there exists $t \in \mathbb{R}$ with $L'_i = L_i + t$ for all i. This looks familiar...

Curiosity 1: Softmax and tropical projective space

Softmax actually exhibits the standard cylinder over tropical projective space:

$$\mathbb{T} = ([-\infty, \infty), +, \max)$$

$$\mathbb{P}^n_{\mathbb{T}} := (\mathbb{T}^n \setminus 0)/(\mathbb{T} \setminus 0)$$

$$\text{softmax} : \mathbb{A}^n_{\mathbb{T}} \setminus 0 \to \mathbb{P}^n_{\mathbb{T}}$$

The tropicals have been spotted several times in LLMs, e.g. (Gaubert, Vlassopoulos, 2024).

Curiosity 2: Softmax maximizes entropy

Proposition

Fix some $L \in \mathbb{R}$.

Given a constraint $\sum_i L_i = L$, there exists $\beta = \beta(L)$ such that $\text{softmax}(\beta L_i)$ *maximizes the Shannon entropy $S = -\sum_i p_i \log p_i$.*

Proof.

$-\beta$ is the Lagrange multiplier $\frac{\partial S}{\partial L}$. See a thermodynamics text or calculus student for details. □

So however much "total energy" is represented in the softmax distribution, that distribution is maximally uncertain among distributions with the same energy.

Entropy is a 1-cocycle for Δ

Proposition

(Shannon) Every 1-cocycle $\Delta \to \mathbb{R}$ is a scalar multiple of the Shannon entropy.

This comes from interpreting the chain rule for conditional entropy as a cocycle condition:

$$H(X, Y) = H(X) + H(Y|X)$$

Asynchronous Algorithmic Alignment with Cocycles

Andrew Dudzik
Google DeepMind
adudzik@google.com

Tamara von Glehn
Google DeepMind
tamaravg@google.com

Razvan Pascanu
Google DeepMind
razp@google.com

Petar Veličković
Google DeepMind
petarv@google.com

Abstract

State-of-the-art neural algorithmic reasoners make use of message passing in graph neural networks (GNNs). But typical GNNs blur the distinction between the definition and invocation of the message function, forcing a node to send messages to its neighbours at every layer, synchronously. When applying GNNs to learn to execute dynamic programming algorithms, however, on most steps only a handful of the nodes would have meaningful updates to send. One, hence, runs the risk of inefficiencies by sending too much irrelevant data across the graph. But more importantly, many intermediate GNN steps have to learn the identity functions, which is a non-trivial learning problem. In this work, we explicitly separate the concepts of node state update and message function invocation. With this separation, we obtain a mathematical formulation that allows us to reason

Cocycles as "carry digits"

Inspired by (Isaksen, 2002), we previously wrote a paper about asynchronous computation, where we identified the 1-cocycle condition as the condition that it was equivalent to aggregate before or after a carry:

$$\delta_{gh}(s) = \delta_g(hs) + \delta_h(s)$$

In curried form:

$$D(gh) = D(g)h + D(h)$$

Curiosity 3: Entropy as Carrying?

When a monoid M acts on a state S with outputs A, a "coherent carry" is just a (right) 1-cocycle $M \to [S, A]$, or equivalently a (left) 1-cocycle $M^{op} \to [S, A]$.

So in principle, we can interpret entropy as a carry for an action of the simplicial *co*-operad Δ^{op}, e.g.:

$$a \mapsto (p_1 a, \ldots, p_n a)$$

My Question to Mathematicians

We saw that our canonical selection function, softmax, is maximizing a 1-cocycle for an operad.

Question

Why are we doing this?

Thanks for listening!

Questions?