Generalizing Backpropagation for Gradient-Based Interpretability

Kevin Du[ð] **Lucas Torroba Hennigen**[ə] **Niklas Stoehr**[ð]
Alexander Warstadt[ð] **Ryan Cotterell**[ð]
[ð]ETH Zürich [ə]MIT
kevin.du@inf.ethz.ch lucastor@mit.edu niklas.stoehr@inf.ethz.ch
alexanderscott.warstadt@inf.ethz.ch ryan.cotterell@inf.ethz.ch

Abstract

Many popular feature-attribution methods for interpreting deep neural networks rely on computing the gradients of a model's output with respect to its inputs. While these methods can indicate which input features may be important for the model's prediction, they reveal little about the inner workings of the model itself. In this paper, we observe that the gradient computation of a model is a special case of a more general formulation using semirings. This observation allows us to generalize the backpropagation algorithm to efficiently compute other interpretable statistics about the gradient graph of a neural network, such as the highest-weighted path and entropy. We implement this generalized algorithm, evaluate it on synthetic datasets to better understand the statistics it computes, and apply it to study BERT's behavior on the subject–verb number agreement task (SVA). With this method, we (a) validate that the amount of gradient flow through a component of a model reflects its importance to a prediction and (b) for SVA, identify which pathways of the self-attention mechanism are most important.

1 Introduction[1]

One of the key contributors to the success of deep learning in NLP has been backpropagation (Linnainmaa, 1976), a dynamic programming algorithm that efficiently computes the gradients of a scalar function with respect to its inputs (Goodfellow et al., 2016). Backpropagation works by constructing a directed acyclic computation graph[2] that describes a function as a composition of various primitive operations, e.g., $+$, $\times$, and $\exp(\cdot)$, whose gradients are known, and subsequently traversing this graph in topological order to incrementally compute the gradients. Since the runtime of backpropagation is linear in the number of edges of the computation graph, it is possible to quickly perform vast numbers of gradient descent steps in even the most gargantuan of neural networks.

While gradients are arguably most important for training, they can also be used to analyze and interpret neural network behavior. For example, feature attribution methods such as saliency maps (Simonyan et al., 2013) and integrated gradients (Sundararajan et al., 2017) exploit gradients to identify which features of an input contribute most towards the model's prediction. However, most of these methods provide little insight into how the gradient propagates through the computation graph, and those that do are computationally inefficient, e.g., Lu et al. (2021) give an algorithm for computing the highest-weighted gradient path that runs in exponential time.

In this paper, we explore whether examining various quantities computed from the gradient graph of a network, i.e., the weighted graph whose edge weights correspond to the local gradient between two nodes, can lead to more insightful and granular analyses of network behavior than the gradient itself. To do so, we note that backpropagation is an instance of a shortest-path problem (Mohri, 2002) over the $(+, \times)$ semiring. This insight allows us to generalize backpropagation to other semirings, allowing us to compute statistics about the gradient graph beyond just the gradient, all while retaining backpropagation's linear time complexity.[3]

In our experiments, the first semiring we consider is the max-product semiring, which allows us to identify paths in the computation graph which carry most of the gradient, akin to Lu et al.'s (2021) influence paths. The second is the entropy semiring (Eisner, 2002),[4] which summarizes how dispersed the gradient graph is, i.e., whether the gradient

[1]Code and data available at https://github.com/kdu4108/semiring-backprop-exps.

[2]With due care, a computation graph can be extended to the cyclic case.

[3]This is analogous to how, in the context of probabilistic context-free grammars, the inside algorithm can be modified to obtain the CKY algorithm (Collins, 2013), and, in the context of graphical models, how the sum-product algorithm for partition functions can be generalized to the max-product algorithm for MAP inference (Wainwright and Jordan, 2008).

[4]Eisner (2002) refers to this as the expectation semiring.

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 11979–11995

flows in a relatively focalized manner through a small proportion of possible paths or in a widely distributed manner across most paths in the network. With experiments on synthetic data, we validate that the max-product semiring results in higher values for model components we expect to be more critical to the model's predictions, based on the design of the Transformer (Vaswani et al., 2017) architecture. We further apply our framework to analyze the behavior of BERT (Devlin et al., 2019) in a subject–verb agreement task (SVA; Linzen et al., 2016). In these experiments, we find that the keys matrix for subject tokens carries most of the gradient through the last layer of the self-attention mechanism. Our results suggest that semiring-lifted gradient graphs can be a versatile tool in the interpretability researcher's toolbox.

2 Gradient-based interpretability

Neural networks are often viewed as black boxes because their inner workings are too complicated for a user to understand *why* the model produced a particular prediction for a given input. This shortcoming has spawned an active field of research in developing methods to better understand and explain how neural networks work. For example, feature attribution methods aim to measure the sensitivity of a model's predictions to the values of individual input features. Many of these methods quantify feature attribution as the gradient of the model's output with respect to an input feature (Simonyan et al., 2013; Smilkov et al., 2017; Sundararajan et al., 2017). We note that while the general reliability and faithfulness of gradient-based methods has been a contentious area of research (Adebayo et al., 2018; Yona and Greenfeld, 2021; Amorim et al., 2023), gradient-based methods have nonetheless continued to be widely used (Han et al., 2020; Supekar et al., 2022; Novakovsky et al., 2022).

Other works have applied feature attribution methods to not only highlight sensitive input features but also uncover important internal neurons. Leino et al. (2018) define influence as the gradient of a quantity of interest with respect to a neuron, averaged across a collection of inputs of interest. Lu et al. (2020) further define and analyze the notion of influence paths, i.e., paths in the computation graph between the neuron of interest and the output that on average carry most of the gradient. By applying this method to analyze the behavior of Gulordava et al.'s (2018) LSTM language model on the SVA task, they draw conclusions about which internal components of the LSTM are most sensitive to the concept of number agreement based on the paths with the greatest amount of influence.

However, Lu et al.'s (2020) method exhaustively enumerates all paths in the computation graph and ranks them by the amount of influence along each one. As the number of paths in a computation graph is usually exponential in the depth of a neural network, this quickly becomes intractable for larger networks (Lu et al., 2021). Therefore, this method is limited to computing influence paths for networks with very small numbers of paths. Indeed, while Lu et al. (2020) computed the influence along $40\,000$ paths for a 2-layer LSTM, follow-up work that attempted to apply this method to BERT had to use an approximation which might not find the correct paths (Lu et al., 2021). The method we propose does not exhibit this issue and scales to any network one can train using backpropagation.

3 Generalizing backpropagation

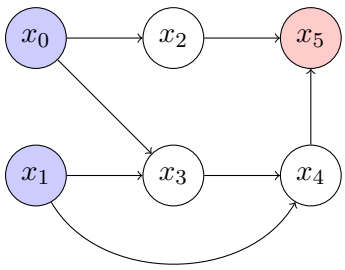
In this section, we build toward our generalization of backpropagation as a semiring-weighted dynamic program. At a high level, we observe that if we replace the addition and multiplication operations in the typical backpropagation algorithm with similar operations that satisfy the necessary properties, then the resulting algorithm will compute other useful statistics about the network's gradient graph in the same runtime as backpropagation. In the remainder of this section, we make this notion of swapping operations precise by formulating backpropagation as a semiring algorithm, and later in §4 we describe how different semirings yield different, useful, views of the gradient graph.

3.1 Computation graphs

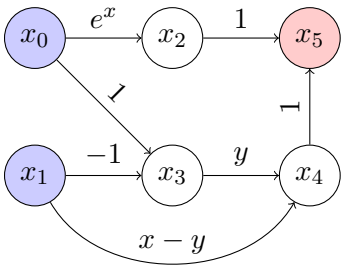
Many classes of functions, e.g., machine learning models, can be expressed as compositions of differentiable functions. Such functions can described by a computation graph (Goodfellow et al., 2016). A **computation graph** is an ordered[5] directed acyclic graph (DAG) where every node is associated with the application of a primitive operation, e.g., $+$, $\times$, and $\exp(\cdot)$, to the parents of that node. These primitives all share the property that their gradients have

[5]We require that nodes be ordered since primitives might not be invariant to permutations of their arguments, e.g., $\frac{x}{y} \neq \frac{y}{x}$ in general.

$x_0 = x$
$x_1 = y$
$x_2 = \exp(x_0)$
$x_3 = x_0 - x_1$
$x_4 = x_1 \times x_3$
$x_5 = x_2 + x_4$



(a) Computation graph of $f(x, y)$



(b) Gradient graph of $f(x, y)$

Figure 1: Comparison of the computation graph (Fig. 1a) and gradient graph (Fig. 1b) of $f(x, y) = e^x + (x - y)y$. The expressions and primitives associated with each node in the graphs are shown on the left. Notice that the gradient graph is a labeled variant of the computation graph. Input nodes are shown in blue, and output nodes in red.

a closed form and are assumed to be computable in constant time for the sake of analysis. Source nodes in the graph are called **input nodes**, and every computation graph has a designated **output node** that encapsulates the result of the function.[6] An example computation graph is shown in Fig. 1a.

If all input nodes are assigned a value, then one can perform a **forward pass**, which calculates the value of the function at those inputs by traversing the graph in a topological order,[7] evaluating the values of each node until we reach the output node. This procedure is shown in Algorithm 1.

Algorithm 1 Forward-propagation

def Forwardpropagation(G, D, τ):
  ▷ *G is a computation graph with topologically-sorted nodes $V = [v_1, \ldots, v_N]$.*
  ▷ *D is an ordered dictionary mapping from nodes to their values, with $D[v_i]$ initialized to the input value associated with $v_i \forall i \in [1, \ldots, m]$.*
  ▷ *$\tau : (V, V) \to \mathbb{N}$ is a function that maps a parent node to the index of the argument list of a function corresponding to a node. That is, given a node v and parent node u, τ maps to an index in $\{1, \ldots, |\pi(v)|\}$ for all $v \in V, u \in \pi(v)$.*
  for $k = m + 1, \ldots, N$:
    $(\boldsymbol{a}_k)_{\tau(v_k,u)} \leftarrow (D[u])_{u \in \pi(v_k)}$ ▷ *Retrieve the value for each input u and store in the ordered argument tuple $\boldsymbol{a}_k$*
    $D[v_k] \leftarrow f_k(\boldsymbol{a}_k)$
  return D

3.2 Backpropagation

Encoding a function as a computation graph is useful because it enables the efficient computation of its gradients via automatic differentiation (Griewank and Walther, 2008). Let G be a computation graph with topologically sorted nodes $v_1, \ldots, v_N$, where v_N is its output node. The goal of **automatic differentiation** is to compute $\frac{\mathrm{d}v_N}{\mathrm{d}v_i}$ for some node v_i in G. Bauer (1974) shows that $\frac{\mathrm{d}v_N}{\mathrm{d}v_i}$ can be expressed as:

$$\frac{\mathrm{d}v_N}{\mathrm{d}v_i} = \sum_{p \in \mathcal{P}(i,N)} \prod_{(j,k) \in p} \frac{\mathrm{d}v_k}{\mathrm{d}v_j} \tag{1}$$

where $\mathcal{P}(i, N)$ denotes the set of **Bauer paths**—directed paths in the computation graph G from node v_i to node v_N.[8] That is, the gradient of the output v_N with respect to a node v_i equals the sum of the gradient computed along every path between v_i and v_N, where the gradient along a path is the product of the gradient assigned to each edge along that path. The gradient of each edge is easy to compute, as it corresponds to the gradient of a primitive. To distinguish the original, unweighted computation graph from its gradient-weighted counterpart, we call the latter the **gradient graph** $\mathcal{G}(\cdot)$ of a function; an example is shown in Fig. 1b. Note that this is a function of the input nodes, since the edge gradients are dependent on the input nodes.

In general, naïvely computing Eq. (1) term by term is intractable since $\mathcal{P}(i, N)$ can be exponential in the number of nodes in the computation graph. By leveraging the distributivity of multiplication over addition, **backpropagation**[9] uses dynamic programming and the caching of intermediate values from the forward pass to compute Eq. (1) in $O(|E|)$ time, where $|E|$ is the number of edges

[6] For simplicity, we only consider scalar-valued functions, but extensions to vector-valued functions are possible and indeed commonplace in the literature.

[7] A topological ordering of a DAG is an ordering of its nodes such that node i precedes node j iff i is not a child of j.

[8] A directed path is an ordered set of node pairs, i.e., $\langle (i_1, i_2), (i_2, i_3), \ldots, (i_{p-1}, i_p) \rangle$ where the second element of each pair matches the first element of the subsequent pair.

[9] Also known as reverse-mode automatic differentiation.

in G (Goodfellow et al., 2016, p. 206). Backpropagation can be seen as traversing the computation graph in reverse topological order and computing the gradient of the output node with respect to each intermediate node until v_i is reached.[10]

3.3 Semiring backpropagation

The crucial observation at the core of this paper is that backpropagation need not limit itself to addition and multiplication: If, instead, we replace those operations with other binary operators that also exhibit distributivity, say $\oplus$ and $\otimes$, then this new algorithm would compute:

$$\frac{\top_{(\oplus,\otimes)} v_N}{\top_{(\oplus,\otimes)} v_i} \triangleq \bigoplus_{p \in \mathcal{P}(i,N)} \bigotimes_{(j,k) \in p} \frac{\mathrm{d}v_k}{\mathrm{d}v_j} \tag{2}$$

Clearly, the interpretation of this resulting quantity depends on how $\oplus$ and $\otimes$ are defined. We discuss different options in §4, and in the remainder of this section we focus on how $\oplus$ and $\otimes$ have to behave to make them suitable candidates for replacement.

To make this notion more rigorous, we first need to introduce the notion of a semiring.

Definition 3.1. A **semiring** (over a set $\mathbb{K}$) is an algebraic structure $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ such that:

1. $\oplus$: $\mathbb{K} \times \mathbb{K} \to \mathbb{K}$ is a commutative and associative operation with identity element $\bar{0}$;
2. $\otimes$: $\mathbb{K} \times \mathbb{K} \to \mathbb{K}$ is an associative operation with identity element $\bar{1}$;
3. $\otimes$ distributes over $\oplus$;
4. $\bar{0}$ is an annihilator, i.e., for any $k \in \mathbb{K}$, $k \otimes \bar{0} = \bar{0} = \bar{0} \otimes k$.

If we replace the operations and identity elements in backpropagation according to the semiring identities and operations, we obtain **semiring backpropagation**, shown in Algorithm 2. Regular backprogation amounts to a special case of the algorithm when run on the **sum-product semiring** $(\mathbb{R}, +, \times, 0, 1)$.

Aggregated derivative. Eq. (2) defines $\frac{\top_{(\oplus,\otimes)} v_N}{\top_{(\oplus,\otimes)} v_i}$ for a single node v_i. However, often it is useful to aggregate this quantity across a set of nodes. For example, when a token is embedded into a d-dimensional vector, each of its dimensions corresponds to a node in the computation graph, say $\mathcal{V} = \{v_1, \ldots, v_d\}$. Then, $\frac{\top_{(\oplus,\otimes)} v_N}{\top_{(\oplus,\otimes)} v_j}$ for the j^{th} component of the representation does not capture the semiring-derivative with respect to the *entire* representation of the token. Hence, we define the **aggregated derivative** with respect to a set of nodes $\mathcal{V}$ as:[11]

$$\frac{\top_{(\oplus,\otimes)} v_N}{\top_{(\oplus,\otimes)} \mathcal{V}} \triangleq \bigoplus_{v \in \mathcal{V}} \frac{\top_{(\oplus,\otimes)} v_N}{\top_{(\oplus,\otimes)} v} \tag{3}$$

Algorithm 2 Semiring backpropagation
This algorithm is executed after the forward pass of a computation graph.

def Backpropagation(G, D):
  ▷ *G is a computation graph with topologically-sorted nodes $V = [v_1, ..., v_N]$.*
  ▷ *D is an ordered dictionary mapping from node v_i to its value, $\forall i \in [1, ..., m]$, computed by the forward pass*
  for $v \in V$:
    $B[v] \leftarrow \bar{0}$ ▷ *B is a dictionary mapping from v_i to $\frac{\top_{(\oplus,\otimes)} v_N}{\top_{(\oplus,\otimes)} v_i}$*
  $B[v_N] \leftarrow \bar{1}$
  for $i = N, ..., 1$:
    for u in $\pi(v_i)$:
      $B[u] \leftarrow B[u] \oplus \left(\frac{\mathrm{d}v_i}{\mathrm{d}u}\Big|_{D[u]} \otimes B[v_i] \right)$
  return B

For standard backpropagation, let $\oplus$ be the addition ($+$) operator and $\otimes$ be the times ($\times$) operator.

4 Interpreting semiring gradients

In §3, we showed how to generalize backpropagation to the semiring case. For any semiring of our choosing, this modified algorithm will compute a different statistic associated with a function's gradient. We begin by motivating the standard $(+, \times)$ semiring which is common in the interpretability literature, before discussing the implementation and interpretation of the max-product and entropy semirings we focus on in this work.

4.1 What is a $(+, \times)$ gradient?

We start by reviewing the gradient interpretation in the $(+, \times)$ semiring, which corresponds to the

[10] Another efficient algorithm for computing Eq. (1) is forward-mode automatic differentiation, which is most useful when one has more output nodes than input nodes in the network (Griewank and Walther, 2008). Since our formulation assumes a single output node, we focus solely on backpropagation.

[11] This is equivalent to adding a dummy source node v_0 with outgoing edges of weight $\bar{1}$ to each node $v \in \mathcal{V}$ to the gradient graph and computing $\frac{\top_{(\oplus,\otimes)} v_N}{\top_{(\oplus,\otimes)} v_0}$.

standard definition of the gradient. We explain why and how the gradient can be useful for interpretability. Let $f : \mathbb{R}^D \to \mathbb{R}$ be a function differentiable at $\mathbf{y} \in \mathbb{R}^D$ (e.g., a neural network model). The derivative of f at $\mathbf{y}$, $\nabla f(\mathbf{y})$, can be interpreted as the best linear approximation of the function at $\mathbf{y}$ (Rudin, 1976), viz., for any unit vector $\mathbf{v} \in \mathbb{R}^D$ and scalar $\epsilon > 0$, we have:

$$f(\mathbf{y} + \epsilon\mathbf{v}) = f(\mathbf{y}) + \nabla f(\mathbf{y})^\top (\epsilon\mathbf{v}) + o(\epsilon) \quad (4)$$

As such, one can view gradients as answering *counterfactual* questions: If we moved our input $\mathbf{y}$ in the direction $\mathbf{v}$ for some small distance ϵ, what is our best guess (relying only on a local, linear approximation of the function) about how the output of the model would change?[12]

Gradient-based methods (as discussed in §2) are useful to interpretability precisely because of this counterfactual interpretation. In using gradients for interpretability, researchers typically implicitly consider $\mathbf{v} = \mathbf{e}_i$, i.e., the i^{th} natural basis vector, which approximates the output if we increment the model's i^{th} input feature by one. We can then interpret the coordinates of the gradient as follows: If its i^{th} coordinate is close to zero, then we can be reasonably confident that small changes to that specific coordinate of the input should have little influence on the value of f. However, if the gradient's i^{th} coordinate is large in magnitude (whether positive or negative), then we may conclude that small changes in the i^{th} coordinate of the input *should* have a large influence on the value of f.

The subsequent two sections address a shortcoming in exclusively inspecting the gradient, which is fundamentally an aggregate quantity that sums over all individual Bauer paths. This means, however, that any information about the structure of that path is left out, e.g., whether a few paths' contributions dominate the others. The semiring gradients that we introduce in the sequel offer different angles of interpretation of such counterfactual statements.

4.2 What is a $(\max, \times)$ gradient?

While the $(+, \times)$ gradient has a natural interpretation given by calculus and has been used in many prior works (Simonyan et al., 2013; Bach et al., 2015; Sundararajan et al., 2017) to identify input features that are most sensitive to a model's output, it cannot tell us *how* the gradient flows through a gradient graph, as discussed in §4.1. One way to compute a different quantity is to change the semiring. The **max-product semiring** $(\mathbb{R} \cup \{-\infty, \infty\}, \max, \times, -\infty, 1)$ is an enticing candidate: In contrast to the $(+, \times)$ semiring, computing the gradient with respect to the $(\max, \times)$ semiring can help illuminate *which* components of the network are most sensitive or critical to the model's input. The $(\max, \times)$ gradient specifically computes the gradient along the Bauer path that has the highest value. We term this path the **top gradient path** in the sequel. Formally, the $(\max, \times)$ gradient between v_i and v_N is:

$$\frac{\nabla_{(\max,\times)} v_N}{\nabla_{(\max,\times)} v_i} \triangleq \max_{p \in \mathcal{P}(i,N)} \prod_{(j,k)\in p} \frac{\mathrm{d}v_k}{\mathrm{d}v_j} \quad (5)$$

Note that variants of this definition are possible, e.g., we could have considered the *absolute* values of the gradients $\left|\frac{\mathrm{d}v_k}{\mathrm{d}v_j}\right|$ if we did not care about the overall impact as opposed to the most *positive* impact on the output v_N.

The top gradient path can be used to examine branching points in a model's computation graph. For example, in Transformer (Vaswani et al., 2017) models, the input to an attention layer branches when it passes through both the self-attention mechanism and a skip connection. The input further branches within the self-attention mechanism between the keys, values, and queries (see Fig. 3 for an illustration). By examining the top gradient path at this branching point, we can identify not only whether the skip connection or self-attention mechanism is more critical to determining input sensitivity, but also which component within the self-attention mechanism itself (keys, queries, or values) carries the most importance.

Implementation. By using the max-product semiring in the backpropagation algorithm, we can compute the top gradient path in $O(|E|)$ time, where $|E|$ is the number of edges in the computation graph (Goodfellow et al., 2016, p. 206). See App. A for more details.

4.3 What is an entropy gradient?

In addition to identifying the single top gradient path, it is also helpful to have a more holistic view of the gradient paths in a graph. In particular, we may be interested in the path entropy of the gradient graph, i.e., the dispersion of the magnitudes of the path weights. Formally, for an input $\mathbf{y}$ and

[12] Indeed, this locality is a common source of criticism for gradient-based interpretability metrics as discussed in §2.

its corresponding gradient graph $\mathcal{G}(\mathbf{y})$ with nodes $v_1, \ldots, v_N$, the **entropy** of all paths between v_i and v_N is defined as:

$$\frac{\neg_{\text{Ent}} v_N}{\neg_{\text{Ent}} v_i} \triangleq -\sum_{p \in \mathcal{P}(i,N)} \left|\frac{g(p)}{Z}\right| \log \left|\frac{g(p)}{Z}\right| \quad (6)$$

where $g(p) \triangleq \prod_{(j,k)\in p} \frac{\mathrm{d}v_k}{\mathrm{d}v_j}$ is the gradient of path p and $Z = \sum_{p\in\mathcal{P}(i,N)} |g(p)|$ is a normalizing factor.

Intuitively, under this view, the gradient graph $\mathcal{G}(\cdot)$ encodes an (unnormalized) probability distribution over paths between v_i and v_N where the probability of a given path is proportional to the absolute value of the product of the gradients along each edge. The entropy then describes the dispersion of the gradient's flow through all the possible paths in the graph from v_i to v_N. For a given graph, the entropy is greatest when the gradient flows uniformly through all possible paths, and least when it flows through a single path.

Implementation. Eisner (2002) proposed to efficiently compute the entropy of a graph by lifting the graph's edge weights into the **expectation semiring** $(\mathbb{R}\times\mathbb{R}, \oplus, \otimes, \bar{0}, \bar{1})$ where $\bar{0} = \langle 0, 0\rangle$, $\bar{1} = \langle 1, 0\rangle$ and:

- $\oplus$: $\langle a, b\rangle \oplus \langle c, d\rangle = \langle a + c, b + d\rangle$
- $\otimes$: $\langle a, b\rangle \otimes \langle c, d\rangle = \langle ac, ad + bc\rangle$

To leverage the expectation semiring, we first lift the weight of each edge in the gradient graph from w to $\langle |w|, |w| \log |w|\rangle$ (where w is the local derivative between two connected nodes in the gradient graph). Then, by computing:

$$\langle Z, -\sum_{p\in\mathcal{P}(i,N)} |g(p)| \log |g(p)|\rangle \quad (7)$$
$$= \bigoplus_{p\in\mathcal{P}(i,N)} \bigotimes_{(j,k)\in p} \left\langle \left|\frac{\mathrm{d}v_k}{\mathrm{d}v_j}\right|, -\left|\frac{\mathrm{d}v_k}{\mathrm{d}v_j}\right| \log \left|\frac{\mathrm{d}v_k}{\mathrm{d}v_j}\right| \right\rangle$$

in linear time using Algorithm 2, we obtain $\langle Z, \sum_{p\in\mathcal{P}(i,N)} |g(p)| \log |g(p)|\rangle$, which are the normalizing factor and the unnormalized entropy of the graph, respectively. As shown by Li and Eisner (2009), we can then compute $\frac{\neg_{\text{Ent}} v_N}{\neg_{\text{Ent}} v_i} = \log Z - \frac{1}{Z}\sum_{p\in\mathcal{P}(i,N)} |g(p)| \log |g(p)|$.

5 Experiments

To demonstrate the utility of semiring backpropogation, we empirically analyze their behavior on two simple transformer models (1-2 layers) on well-controlled, synthetic tasks. We also explore semiring backpropogation on a larger model, BERT (Devlin et al., 2019), on the popular analysis task of subject–verb agreement to understand how our method can be useful for interpreting language models in more typical settings.

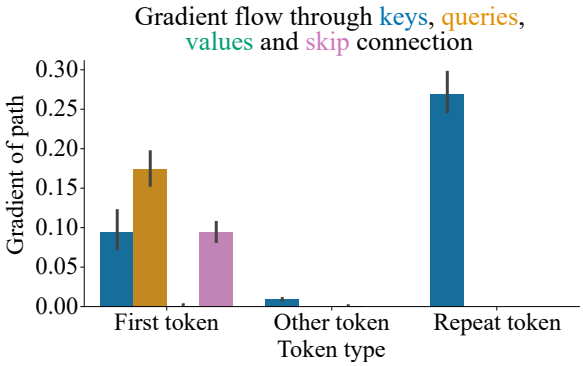


Figure 2: High gradient flow through the queries for the first token and keys of the repeat token match the expected important components of self-attention for each token type, respectively.

To implement semiring backpropagation, we developed our own Python-based reverse-mode automatic differentiation library, building off of the pedagogical library Brunoflow (Ritchie, 2020) and translating it into JAX (Bradbury et al., 2018).[13]

5.1 Validation on a synthetic task

Setup. In this experiment, we test the hypothesis that most of the gradient should flow through the components that we judge *a priori* to be most critical to the model's predictions. We are particularly interested in whether the gradient flow through a Transformer matches our expectation of the self-attention mechanism's components. So, while we compute the top gradient path from the output to the input representations, we only inspect the top path at a Transformer's main branching point, which is when the hidden state is passed into the skip connection and the keys, values, and queries of the self-attention mechanism (Fig. 3). If we observe higher levels of gradients flowing through one branch, a natural interpretation is that this component is more critical for the model's prediction. To test whether this interpretation is justified, we construct a task where we can clearly reason about how a well-trained Transformer model ought to behave and identify how well the top gradient flow aligns with our expectations of a model's critical component.

[13] Library available at https://github.com/kdu4108/brunoflow.

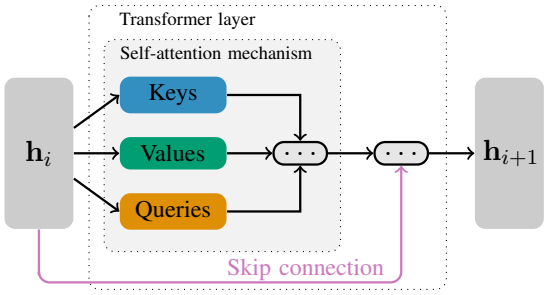


Figure 3: Simplified computation graph for the i^{th} layer of a Transformer (Vaswani et al., 2017) encoder at the key branching point where a hidden state is passed into the skip connection and the keys, values, and queries of the self-attention mechanism. We use $\mathbf{h}_i$ to denote the hidden representations returned by layer $i-1$, and use "..." to denote others parts of the computation graph.

Model. We use a 1-layer Transformer model with hidden layer size of 16 and 2 attention heads to minimize branching points and increase interpretability. We train this model to achieve 100% validation accuracy on the task described below.

Task. We design the `FirstTokenRepeatedOnce` task to target the utility of this method for interpreting the self-attention mechanism. In this task, an input consists of a sequence of numbers, which is labeled according to whether the first token appears again at any point in the sequence, e.g., `[1, 4, 6, 1]` → `True`, whereas `[3, 4, 6, 2]` → `False`. Furthermore, the inputs are constrained such that the first token will be repeated at most once, to isolate the decision-making of the model to the presence (or lack thereof) of a single token. We randomly generate a dataset of 10 000 points with sequence length 10 and vocab size 20. The correct decision-making process for this task entails comparing the first token to all others in the sequence and returning `True` if there is a match. This is, in fact, analogous to how queries and keys function within the self-attention mechanism: A query q_t is compared to the key $k_{t'}$ of each token t' in the sequence and the greater the match, the greater attention paid to token t' by query token t. We would therefore expect that the self-attention mechanism relies heavily on the query representation of the first token and key representations of the remaining tokens and, in particular, the key representation of the repeated token, if present. In turn, we hypothesize the max-product gradient value will primarily originate from the queries branch for the first token and keys for the remaining tokens, and be especially high for the repeat token.

Results. The results, summarized in Fig. 2, provide strong evidence for our hypothesis that the behavior of the $(\max, \times)$ gradient reflects the importance of the different model components. We observe all expected gradient behaviors described in the previous paragraph, and especially that the highest gradient flow (for any token) is through the keys of the repeat token.

5.2 Top gradient path of BERT for subject–verb agreement

Setup. We now apply this method to understand the self-attention mechanism of a larger model (BERT) for the more complex NLP task of SVA. We subsample 1000 examples from the dataset from Linzen et al. (2016) and use spaCy (Matthew et al., 2020) to identify the subject and attractors within each sentence. We then filter down to 670 sentences after removing sentences where BERT tokenizes the subject or attractors as multiple tokens. Using the max-product semiring, we then compute the top gradient path through the different branches (skip connection, keys, values, and queries) for (a) the subject of a sentence, (b) the attractors of a sentence, and (c) all tokens of a sentence.

Model. BERT (Devlin et al., 2019) is a popular encoder-only Transformer model for many NLP tasks. BERT's architecture consists of multiple Transformer encoder layers stacked atop each other, along with a task-specific head. We use the `google/bert_uncased_L-6_H-512_A-8` pretrained model from Huggingface (Wolf et al., 2020), which has 6 attention layers, hidden size of 512, and 8 attention heads.

Task. We consider the subject–verb number agreement task in our experiments. Variants of this task in English have become popular case studies in neural network probing. Notably, this phenomenon has been used to evaluate the ability for models to learn hierarchical syntactic phenomena (Linzen et al., 2016; Gulordava et al., 2018). It has also served as a testing ground for interpretability studies which have found evidence of individual hidden units that track number and nested dependencies (Lakretz et al., 2019), and that removing individual hidden units or subspaces from the models' representation space have a targeted impact on model predictions (Finlayson et al., 2021; Lasri et al., 2022). Our formulation of the task uses BERT's native masked language modeling capability by recasting it as a cloze task: We mask a verb in the

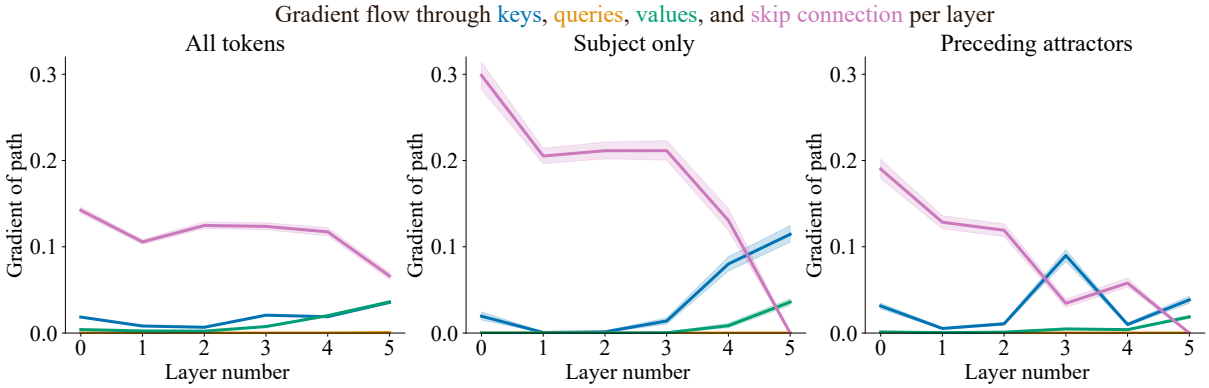


Figure 4: Each plot depicts the top gradient path behavior of BERT for different tokens in a sentence, averaged across the 670 sentences. Notably, for subjects (middle plot), the *skip connection* contains the top gradient path for all layers except the final layer, which is consistent with findings from (Lu et al., 2021). In the final layer, most of the gradient for both the subject and attractors flows through the *keys* of the self-attention mechanism. This differs from the gradient flow averaged across all tokens, indicating this behavior is specific to the nouns of the sentence.

sentence and compare the probabilities with which BERT predicts the verb forms with correct and incorrect number marking. For example, given the input "all the other albums produced by this band `[MASK]` their own article," we compare the probabilities of "have" (correct) and "has" (incorrect). We compute the gradient with respect to the difference between the log probability of the two inflections.

The data for this experiment is from Linzen et al. (2016). All the examples in their dataset also include one or more **attractors**. These are nouns such as "band" in the example above, which (a) are not the subject, (b) precede the verb, and (c) disagree with the subject in number. Furthermore, all masked verbs are third person and present tense, to ensure that number agreement is non-trivial.

Results. From Fig. 4, we highlight key differences between the $(\max, \times)$ gradient behavior for subject tokens and all tokens in general. Most saliently, for subject tokens only, the max-product gradient flows entirely through the self-attention mechanism in the last layer and mostly through the skip connection in earlier layers, which is consistent with findings from Lu et al. (2021). Moreover, within the self-attention mechanism, most (76%) of the gradient in the last layer for the subject flows through the keys matrix. In contrast, across all tokens, the top gradient paths mostly through the skip connection for all layers, and otherwise is more evenly distributed between keys and values.

We also note similarities and differences between the gradient flows of the subject and preceding attractors. Both exhibit a similar trend in which the gradient flows primarily through the keys

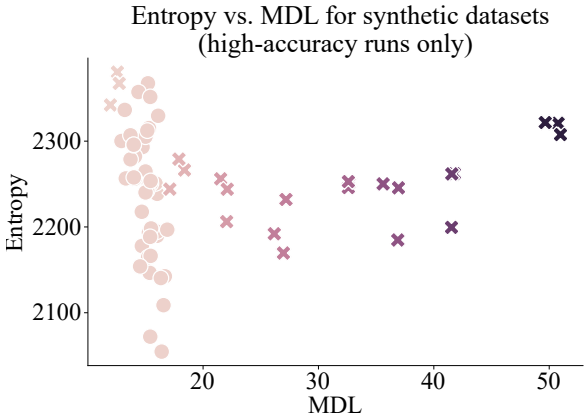


Figure 5: Each point represents the entropy and MDL of a model trained on a given dataset (3 seeds per dataset). We denote the `BinCountOnes` datasets with the × marker and other tasks (`ContainsTokenSet` and tasks from Lovering et al., 2021) with the • marker. The hue corresponds to the number of classes of the task; the lightest hue indicates a binary problem while the darker hues indicate more classes (max of 36).

(and entirely through the self-attention mechanism) in the last layer. However, the top gradient has a greater magnitude for the subject than the attractors (especially in the keys). Since self-attention uses a token's keys to compute the relative importance of that token to the `[MASK]` token, we speculate that the max-product gradient concentrating primarily on the keys (and more so for the subject than attractors) reflects that a successful attention mechanism relies on properly weighting the importances of the subject and attractors.

5.3 Gradient graph entropy vs. task difficulty

Setup. This experiment tests the hypothesis that the entropy of a model's gradient graph is positively

correlated with the difficulty of the task that the model was trained to solve. We construct a variety of synthetic tasks and compare the average gradient entropy of a 2-layer transformer on examples in each of these tasks. We measure the difficulty of a task with the minimum description length (MDL; Rissanen, 1978).[14] Following the approach used by Lovering et al. (2021) and Voita and Titov (2020), we measure MDL by repeatedly training the model on the task with increasing quantities of data and summing the loss from each segment. The higher the MDL, the more difficulty the model had in extracting the labels from the dataset, and therefore the more challenging the task. We hypothesize that a model will have higher entropy for more difficult tasks because it will require using more paths in its computation graph. During our analysis, we drop runs where the model was unable to achieve a validation accuracy of $> 90\%$, to avoid confounding results with models unable to learn the task.

Model. For all tasks, we use the same 2-layer transformer architecture with a hidden layer size of 64, 4 attention heads, and always predicts a distribution over 36 classes (with some possibly unused); this ensures our results are comparable across tasks with different numbers of classes. We train the models for 50 epochs on each of the synthetic datasets.

Task. We design a variety of synthetic tasks in order to control for difficulty more directly. In the `ContainsTokenSet` family of tasks, an input is a sequence of S numbers and labeled `True` or `False` based on whether the input contains *all* tokens in a pre-specified token set. Different tasks within `ContainsTokenSet` are defined by the pre-specified token set. The `BinCountOnes` family of tasks is parameterized by a number of classes C. In this task, an input x is a sequence of S numbers. The label y is determined by the number of 1s in the sequence according to the following function: $y(x) = \left\lceil \frac{\texttt{Count1}(x)}{S/C} \right\rceil - 1$, i.e., in the 2-class instance of `BinCountOnes`, an input is labeled 0 if it contains $\leq S/2$ 1s and 1 if it contains $> S/2$ 1s. Finally, we also evaluate on the synthetic datasets `Contains1`, `AdjacentDuplicate`, `FirstTokenRepeatedImmediately`, and `FirstTokenRepeatedLast` from (Lovering et al., 2021). For more details, see App. C.

Results. The results show clear evidence against our initial hypothesis that gradient entropy increases as a function of task difficulty, as measured by MDL. While there appears to be some patterns evident between entropy and MDL in Fig. 5, their interpretation is unclear. From observing the lightest-hued points there appears to be a negative linear relationship between entropy and MDL for the binary tasks. However, confusingly, the × points seem to suggest a quadratic-like relationship between entropy and MDL for the `BinCountOnes` tasks. We speculate that this could be explained by a phase-change phenomena in the model's learning dynamics. That is, for sufficiently easy tasks, the model need not focalize much in order to solve the task. Incrementally more difficult tasks may require the model to focalize more, thus resulting in the decreasing entropy for tasks below a certain MDL threshold. Then, once a task is sufficiently difficult, the model is required to use more of the network to solve the task. Therefore, we see this increase in entropy as the MDL increases past a certain threshold for the `BinCountOnes` task. The presence of these clear (although somewhat mystifying) patterns indicates that there exists *some* relationship between entropy and MDL. More experimentation is needed to understand the relationship between entropy and MDL for task difficulty.

6 Conclusion

We presented a semiring generalization of the backpropagation algorithm, which allows us to obtain an alternative view into the inner workings of a neural network. We then introduced two semirings, the max-product and entropy semirings, which provide information about the branching points of a neural network and the dispersion of the gradient graph. We find that gradient flow reflects model component importance, gradients flowing through the self-attention mechanism for the subject token pass primarily through the keys matrix, and the entropy has some relationship with the difficulty of learning a task. Future work will consider semirings outside the scope of this work, e.g., the **top-k semiring** (Goodman, 1999) to track the top-k gradient paths, as well as computing semirings online for control during training.

[14] The MDL of a dataset under a model measures the number of bits required to communicate the labels of the dataset, assuming the sender and receiver share both the unlabeled data and a model, which can be used to reduce the information the sender must transmit. Alternatively, MDL can be thought of as the area under the loss curve as a function of dataset size.

7 Limitations

While our approach inherits the linear runtime complexity of the backpropagation algorithm, runtime concerns should not be fully neglected. Firstly, the linear runtime is only an analytical result, not an empirical measure. This means that the actual runtime of the backpropagation and thus our algorithm depend heavily on their implementation. For instance, some deep learning frameworks do a better job at reusing and parallelizing computations than others (Goodfellow et al., 2016). Indeed, our code is optimized for good readability and extensibility at the expense of speed, which hints at another limitation of our approach: Our approach requires deep integration with the framework as it needs access to all model weights and the computation graph. For this reason, our approach cannot be easily packaged and wrapped around any existing model or framework and we instead developed our own JAX-based reverse-mode autodifferentiation library, based on the numpy-based Brunoflow library (Ritchie, 2020). While we release our library to enable other researchers to analyze models through their gradient graphs, it faces some computational and memory constraints. In our experiments, running the three semirings together on a single sentence can take several minutes (depending on sentence length) using `google/bert_uncased_L-6_H-512_A-8`, the 6-layered pretrained BERT from Huggingface (Wolf et al., 2020), totaling our experimentation time on our datasets at about 10 CPU-hours. For improved adoption of this method, we encourage the direct integration of semiring implementations into the most popular deep learning frameworks. Our final point pertains not only to our study but to most interpretability approaches: One has to be careful when drawing conclusions from gradient paths. Cognitive biases, wrong expectations, and omitted confounds may lead to misinterpretation of results.

Ethics statement

We foresee no ethical concerns with this work. Our work aims to make the inner workings of neural network models more interpretable. On this account, we hope to contribute to reducing biases inherent in model architectures, pre-trained model weights, and tasks by increasing overall transparency.

Acknowledgements

Kevin Du acknowledges funding from the Fulbright/Swiss Government Excellence Scholarship. Lucas Torroba Hennigen acknowledges support from the Michael Athans fellowship fund. Niklas Stoehr acknowledges funding through the Swiss Data Science Center (SDSC) Fellowship. Alex Warstadt acknowledges support through the ETH Postdoctoral Fellowship program.

References

Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. 2018. Sanity checks for saliency maps. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

José P. Amorim, Pedro H. Abreu, João Santos, Marc Cortes, and Victor Vila. 2023. Evaluating the faithfulness of saliency maps in explaining deep learning models using realistic perturbations. *Information Processing & Management*, 60(2):103225.

Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. 2015. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS ONE*, 10(7):e0130140.

Friedrich L. Bauer. 1974. Computational graphs and rounding error. *SIAM Journal on Numerical Analysis*, 11(1):87–96.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. JAX: Composable transformations of Python+NumPy programs.

Michael Collins. 2013. Probabilistic context-free grammars (PCFGs). Technical report, Columbia University.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Jason Eisner. 2002. Parameter estimation for probabilistic finite-state transducers. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 1–8, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Matthew Finlayson, Aaron Mueller, Sebastian Gehrmann, Stuart Shieber, Tal Linzen, and Yonatan Belinkov. 2021. Causal analysis of syntactic agreement mechanisms in neural language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1828–1843, Online. Association for Computational Linguistics.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT press.

Joshua Goodman. 1999. Semiring parsing. *Computational Linguistics*, 25(4):573–605.

Andreas Griewank and Andrea Walther. 2008. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, second edition. Society for Industrial and Applied Mathematics, USA.

Kristina Gulordava, Piotr Bojanowski, Edouard Grave, Tal Linzen, and Marco Baroni. 2018. Colorless green recurrent networks dream hierarchically. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1195–1205, New Orleans, Louisiana. Association for Computational Linguistics.

Xiaochuang Han, Byron C. Wallace, and Yulia Tsvetkov. 2020. Explaining black box predictions and unveiling data artifacts through influence functions. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5553–5563, Online. Association for Computational Linguistics.

Yair Lakretz, German Kruszewski, Theo Desbordes, Dieuwke Hupkes, Stanislas Dehaene, and Marco Baroni. 2019. The emergence of number and syntax units in LSTM language models. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 11–20, Minneapolis, Minnesota. Association for Computational Linguistics.

Karim Lasri, Tiago Pimentel, Alessandro Lenci, Thierry Poibeau, and Ryan Cotterell. 2022. Probing for the usage of grammatical number. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8818–8831, Dublin, Ireland. Association for Computational Linguistics.

Klas Leino, Shayak Sen, Anupam Datta, Matt Fredrikson, and Linyi Li. 2018. Influence-directed explanations for deep convolutional networks. In *IEEE International Test Conference*, pages 1–8. IEEE.

Zhifei Li and Jason Eisner. 2009. First- and second-order expectation semirings with applications to minimum-risk training on translation forests. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 40–51, Singapore. Association for Computational Linguistics.

Seppo Linnainmaa. 1976. Taylor expansion of the accumulated rounding error. *BIT*, 16(2):146–160.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535.

Charles Lovering, Rohan Jha, Tal Linzen, and Ellie Pavlick. 2021. Predicting inductive biases of fine-tuned models. In *International Conference on Learning Representations*.

Kaiji Lu, Piotr Mardziel, Klas Leino, Matt Fredrikson, and Anupam Datta. 2020. Influence paths for characterizing subject-verb number agreement in LSTM language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4748–4757, Online. Association for Computational Linguistics.

Kaiji Lu, Zifan Wang, Piotr Mardziel, and Anupam Datta. 2021. Influence patterns for explaining information flow in BERT. In *Advances in Neural Information Processing Systems*, volume 34, pages 4461–4474. Curran Associates, Inc.

Honnibal Matthew, Montani Ines, Van Landeghem Sofie, and Boyd, Adriane. 2020. spaCy: Industrial-strength Natural Language Processing in Python.

Mehryar Mohri. 2002. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages, and Combinatorics*, 7(3):321–350.

Gherman Novakovsky, Nick Dexter, Maxwell W. Libbrecht, Wyeth W. Wasserman, and Sara Mostafavi. 2022. Obtaining genetics insights from deep learning via explainable artificial intelligence. *Nature Reviews Genetics*, pages 1–13.

Jorma Rissanen. 1978. Modeling by shortest data description. *Automatica*, 14(5):465–471.

Daniel Ritchie. 2020. Brunoflow: A pedagogical deep learning framework. Technical report.

Walter Rudin. 1976. *Principles of Mathematical Analysis*. International series in pure and applied mathematics. McGraw-Hill.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2013. Deep inside convolutional networks: Visualising image classification models and saliency maps.

Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. 2017. Smoothgrad: Removing noise by adding noise. In *Proceedings of the ICML Workshop on Visualization for Deep Learning*.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, page 3319–3328. JMLR.org.

Kaustubh Supekar, Carlo de los Angeles, Srikanth Ryali, Kaidi Cao, Tengyu Ma, and Vinod Menon. 2022. Deep learning identifies robust gender differences in functional brain organization and their dissociable links to clinical symptoms in autism. *The British Journal of Psychiatry*, 220(4):202–209.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Elena Voita and Ivan Titov. 2020. Information-theoretic probing with minimum description length. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 183–196, Online. Association for Computational Linguistics.

Martin J. Wainwright and Michael I. Jordan. 2008. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1–2):1–305.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online.

Gal Yona and Daniel Greenfeld. 2021. Revisiting sanity checks for saliency maps. *arXiv*.

A Implementation of Top Gradient Path

In practice, we implement the top gradient path by storing 4 additional fields to each node in the graph: the most positive gradient of the node, a pointer to the child node which contributed this most positive gradient, the most negative gradient of the node, and a pointer to the child node which contributed this most negative gradient. In this way, each node tracks the paths containing the most positive gradient (top_pos) and most negative gradient (top_neg) from itself to the output node. To dynamically extend the path from v_k to v_j ($j < k$):

$$v_j.\text{top\_pos} = \begin{cases} v_k.\text{top\_pos} \cdot \frac{\mathrm{d}v_k}{\mathrm{d}v_j} & \text{if } \frac{\mathrm{d}v_k}{\mathrm{d}v_j} \geq 0 \\ v_k.\text{top\_neg} \cdot \frac{\mathrm{d}v_k}{\mathrm{d}v_j} & \text{otherwise} \end{cases}$$

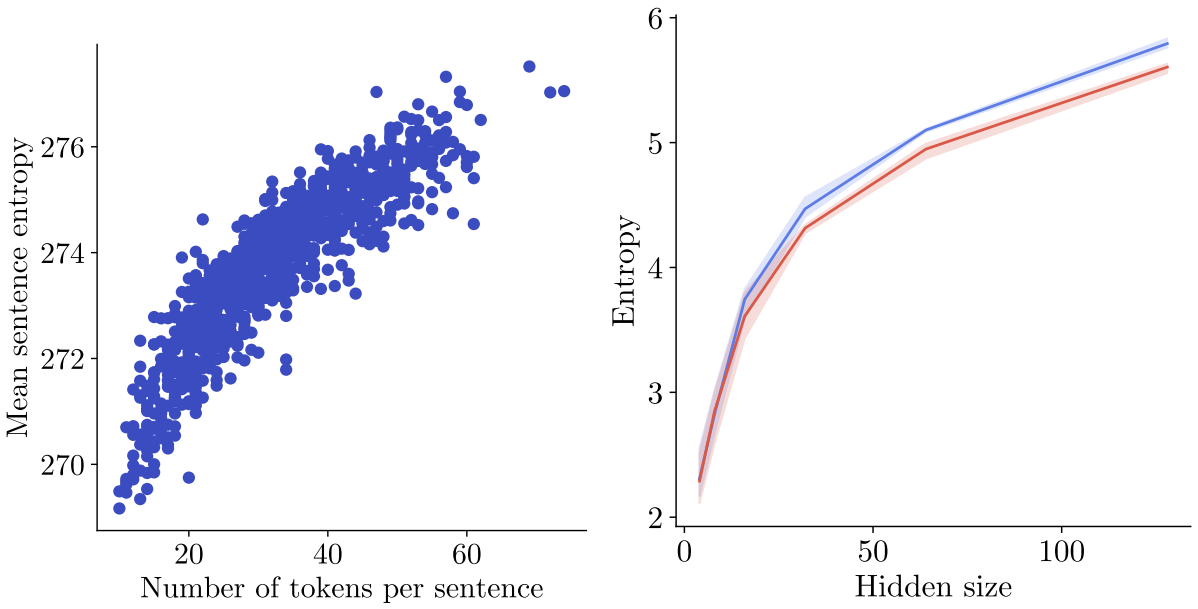
$$v_j.\text{top\_neg} = \begin{cases} v_k.\text{top\_neg} \cdot \frac{\mathrm{d}v_k}{\mathrm{d}v_j} & \text{if } \frac{\mathrm{d}v_k}{\mathrm{d}v_j} \geq 0 \\ v_k.\text{top\_pos} \cdot \frac{\mathrm{d}v_k}{\mathrm{d}v_j} & \text{otherwise} \end{cases}$$

B Additional Entropy Sanity Checks and Experiments

B.1 Sanity Checks with Synthetic Data

To build intuition about the entropy of a model's computation graph, we run two sanity check experiments. First, we evaluate the entropy of a pretrained BERT model as the sentence length increases. Since larger sentence lengths result in more paths in the computation graph, we expect the entropy of the model to increase with sentence length. Our findings confirm this (Fig. 6a).

Second, we expect that the entropy of a trained model ought to increase with the model complexity, as measured by hidden size. In this experiment, we create a 4-featured artificial dataset with randomly generated values in the range $[0, 1]$, labeled by whether the first feature is greater than 0.5. We train multilayer perceptrons with varying hidden sizes on this dataset and find that the entropy of the input features increases with model complexity as expected (see Fig. 6b).



(a) Entropy vs sentence length

(b) Entropy vs model complexity (relevant & irrelevant features)

Figure 6: Fig. 6a shows that the more tokens a sentence contains, the more gradient paths are naturally involved and consequently the higher the overall entropy. Fig. 6b shows that as the hidden size of the MLP increases, so too does the model entropy. By comparing the entropy of the relevant feature line and the irrelevant features line, it also appears that the entropy is consistently higher for the relevant feature than irrelevant features, especially as model complexity increases.

B.2 Entropy vs Example Difficulty in Subject–Verb Agreement

Setup. We investigate the relationship between the entropy of the gradient graph of BERT and input sentences in the task of subject–verb number agreement. In this task, we measure example difficulty by the number of attractors in a sentence (more attractors corresponds to greater difficulty). We sub-sample the dataset from Linzen et al. (2016) to 1000 sentences, balanced evenly by the number of attractors per sentence (ranging from 1 to 4 attractors). Then, using the entropy semiring, we compute the entropy of BERT's gradient graph for each sentence.

Results. Since sentences with more tokens will naturally have a higher entropy due to a larger computation graph (see Fig. 6a), we control by sentence length. We bin sentences of similar length for (10–20, 20–30, 30–40, and 40–50 tokens) before analyzing the effect that the number of attractors has on entropy. We present the results in Fig. 7 and additionally run a Spearman correlation test between the entropy of the input representations (averaged across all tokens in the sentence) and the number of attractors. For each group of sentence lengths, we find minimal correlation between number of attractors and entropy. Therefore, there is little evidence to support a relationship between entropy and example difficulty as measured by number of attractors. However, number of attractors is not necessarily a strong indicator of example difficulty, and recommend more rigorous comparison of entropy against a stronger metric of example difficulty in future work.

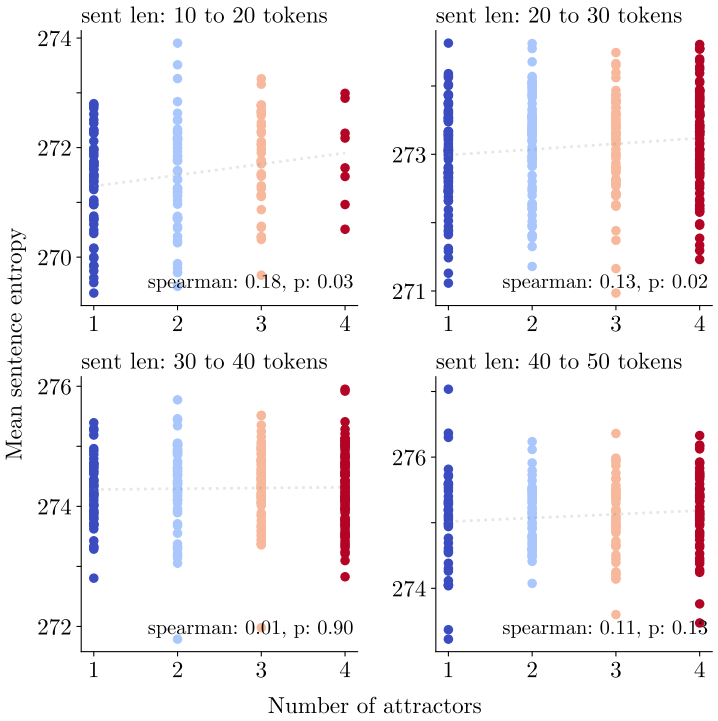


Figure 7: For all sentence length bins, there appears to be little to no correlation between number of attractors and entropy.

C Synthetic Datasets

C.1 Binary Datasets

We list in Tab. 1 descriptions and examples of all binary tasks constructed for our experiments.

C.2 BinCountOnes Datasets

We construct one family of multiclass classification datasets, `BinCountOnes`.

Parameterization. A `BinCountOnes` task is parameterized by the number of classes C, between 2 to S, such that C divides S. For example, when $S = 6$, C could be 3.

Task Name	Parameterized by:	Description	Positive Example	Negative Example
ContainsTokenSet	A set of tokens, T, e.g., `{1,2,3}`	Labeled `True` if X contains every token in T and `False` otherwise	`[1,3,4,2,5,2]`	`[1,5,9,2,2,4]`
Contains1	N/A	Labeled `True` if X contains the token `1` and `False` otherwise	`[1,3,4,2,5,2]`	`[6,5,9,2,2,4]`
FirstToken-RepeatedImmediately	N/A	Labeled `True` if the first two tokens in X are the same and `False` otherwise	`[3,3,2,6,7,8]`	`[5,3,2,6,7,8]`
FirstToken-RepeatedLast	N/A	Labeled `True` if the first and last tokens in X are the same and `False` otherwise	`[8,3,2,6,7,8]`	`[8,3,2,6,7,4]`
AdjacentDuplicate	N/A	Labeled `True` if two adjacent tokens in X are the same at any point in the sequence and `False` otherwise	`[1,3,6,6,7,8]`	`[1,3,6,8,7,8]`
FirstToken-RepeatedOnce	N/A	Labeled `True` if the first token in X is repeated at any point in the sequence and `False` otherwise. X is further constrained to have at most one repeat of the first token in X.	`[1,3,6,1,7,8]`	`[1,3,6,7,7,8]`

Table 1: Binary synthetic datasets used in §5.1 and §5.3. For all tasks, the input X is a sequence of S numbers (valued from 1 to vocab size). While for the examples in this table we use $S = 6$ to save space, in the actual experiments we use $S = 10$ (§5.1) and $S = 36$ (§5.3).

Description. Each example X is labeled between $[0, C-1]$ by the following formula: $\text{label}(X) = \left\lceil \frac{\text{Count1}(X)}{S/C} \right\rceil - 1$, where $C\text{Count1}(X)$ is the number of 1s that appear in X.

Examples. See Tab. 2.

Input	Label
`[1,3,4,2,5,2]`	0
`[1,3,4,2,3,1]`	0
`[1,3,4,2,1,1]`	1
`[1,3,1,1,5,1]`	1
`[1,3,1,1,1,1]`	2
`[1,1,1,1,1,1]`	2

Table 2: Example inputs and labels for the `BinCountOnes` task where sequence length $S = 6$ and number of classes $C = 3$.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
8

☒ A2. Did you discuss any potential risks of your work?
We foresee potential risks of our work, as our work aims at making models more interpretable. We hope to contribute to reducing biases inherent in model architectures, pre-trained model weights and tasks by increasing overall transparency.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Left blank.

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☒ Did you use or create scientific artifacts?

Left blank.

☐ B1. Did you cite the creators of artifacts you used?
No response.

☐ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
No response.

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
No response.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
No response.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
No response.

☐ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
No response.

C ☑ Did you run computational experiments?

5

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
8

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
5

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
5

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
5

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.