THE QUARKS OF ATTENTION: STRUCTURE AND CAPACITY OF NEURAL ATTENTION BUILDING BLOCKS

PIERRE BALDI AND ROMAN VERSHYNIN

Department of Computer Science, University of California, Irvine

Department of Mathematics, University of California, Irvine

Abstract. Attention plays a fundamental role in both natural and artificial intelligence systems. In deep learning, attention-based neural architectures, such as transformer architectures, are widely used to tackle problems in natural language processing and beyond. Here we investigate the most fundamental building blocks of attention and their computational properties within the standard model of deep learning. We first derive a systematic taxonomy of all possible attention mechanisms within, or as extensions of the standard model into 18 classes depending on the origin of the attention signal, the target of the attention signal, and whether the interaction is additive or multiplicative. Second, using this taxonomy, we identify three key attention mechanisms: additive activation attention (multiplexing), multiplicative output attention (output gating), and multiplicative synaptic attention (synaptic gating). Output gating and synaptic gating are proper extensions of the standard model and all current attention-based architectures, including transformers, use either output gating or synaptic gating, or a combination of both. Third, we develop a theory of attention capacity and derive mathematical results about the capacity of basic attention networks comprising linear or polynomial threshold gates. For example, the output gating of a linear threshold gate of n variables by another linear threshold gate of the same n variables has capacity $2n^2(1+o(1))$, achieving the maximal doubling of the capacity for a doubling of the number of parameters. Perhaps surprisingly, multiplexing attention is used in the proofs of these results. Synaptic and output gating provide computationally efficient extensions of the standard model enabling *sparse* quadratic activation functions. They can also be viewed as primitives for collapsing several layers of processing in the standard model into shallow compact representations.

Keywords: neural networks; deep learning; attention; gating; synaptic modulation; transformers; capacity; circuit complexity.

"Everyone knows what attention is... It is the taking possession by the mind in clear and vivid form, of one out of what seem several simultaneously possible objects or trains of thought..." William James, Principles of Psychology (1890).

1. Introduction

Everyone can focus their attention on an image, a sound, or a thought. But what is attention and how does it really work? Besides James's definition, other standard definitions of attention include: "*the ability to focus selectively on a selected stimulus, sustaining that focus and shifting it at will*"; or, linking attention to awareness: "*the concentration of awareness*

E-mail address: pfbaldi@uci.edu, rvershyn@uci.edu.

Date: March 14, 2023.

on some phenomenon to the exclusion of other stimuli". All such definitions are very coarse, imprecise, and based on introspective and phenomenological considerations. The all define attention in terms of other functionally obscure terms, such as "focus" or "awareness". Here, in order to better understand attention at the computational level, we study it within the simplified framework of artificial neural networks and deep learning by first identifying its most fundamental building blocks or quarks, using a physics-inspired terminology, and then rigorously analyzing some of their computational properties.

The motivation for working with artificial neural networks is two-fold. The first motivation is to avoid getting bogged down by the complexity of biological systems. There is of course a substantial literature on the neurobiology and psychophysics of attention (e.g., [15, 2, 21]) pointing to a variety of different phenomena and attention systems, leading some to conclude: "*The word"attention" is an inadequate, singular term for a multitude of inter-related processes. We use a host of adjectives to describe attention-for example, we say that attention can be divided, oriented, sustained, or focused, and many of these descriptions likely reflect underlying, dissociable neural processes. Complicating matters, attentional resources can be allocated to either external stimuli, or to internal stimuli such as thoughts and memories. Furthermore, we often confuse the regulation of attention (a covert behavior) with the regulation of movement (an overt behavior) when discussing an "attentional disorder""* [2]. In spite of this complexity and diversity of processes, we believe that at the most fundamental level attention mechanisms are built out of a small number of fundamental operations, which occur on time scales that are fast compared to the time scales of learning and long-term synaptic modifications. For instance, in order to exclude other stimuli, neuronal machinery must exist that is capable of dynamically suppressing the activity of subsets of neurons, or subsets of connections, or both, associated with the non-attended information. These fundamental operations may be easier to identify and study using artificial neural networks. In particular, one of our goals here is to produce a systematic nomenclature of all such possible operations, within the standard deep learning model. While this is not the place to discuss the relationship between artificial and biological neural networks, there is a body of evidence showing that, atleast at some level, the former can provide useful information about the latter (e.g., [28, 20, 27, 6]).

The second motivation, equally or even more important, is that attention plays an increasingly important role in deep learning, where various attention mechanisms such as content-based attention [14], speech recognition attention [10], or dot product attention [19], have been proposed and deployed in applications. Many of these mechanisms were initially developed for speech and natural language applications (NLP) (e.g., [4, 11, 22]), but they are now being adapted to other problems (e.g., [17, 13, 24]). The intuitive idea in NLP applications is that when, for instance, translating a sentence from one language to another, the underlying neural algorithm should be able to dynamically shift its focus on the relevant words and context, while filtering out the less relevant ones. For instance when translating "the red roof" into the French "le toit rouge" the machinery that produces the *third* word of the output ("rouge") should dynamically give more importance to the *second* word ("red") of the input, relative to the other neighboring words. The current pinnacle of attention-based architectures is the transformer architecture [26] which has led to state-of-the-art performance in NLP and large language models (LLMs). While these advances have even led some experts to speculate that attention mechanisms may be the key for achieving machine consciousness (!), with rare exceptions [12], there is little theory to help us better understand the nature,scope, and computational capabilities of attention.

Main Results and Roadmap. To address this gap and provide a systematic treatment of attention mechanisms in deep learning, we first note that in the standard neural network model comprised of McCulloch and Pitt neurons, there are three kinds of variables: synaptic weights (w), neuronal activations (S), and neuronal outputs (O). Thus, for any attention mechanism implemented in this model, there are three possibilities for the source of the attending signal, and three possibilities for the target of the attention signal. Considering two basic interactions–addition and multiplication–between the attending signal and the attended target yields a total of 18 possibilities. Next, the most reasonable assumption is that the source of the attending signal is of type O. This leaves six possibilities, among which we further identify three key attention mechanisms: additive activation attention (multiplexing), multiplicative output attention (output gating), and multiplicative synaptic attention (synaptic gating). Output gating and synaptic gating are proper extensions of the standard model and all current attention-based architectures use either output gating or synaptic gating, or a combination of both. We show how each attention module within a transformer architecture is built using both output gating in order to compute dot products, and synaptic gating in order to compute outputs. Finally, we use the notion of neural capacity to develop a theory of attention capacity. The capacity of a feed-forward neural architecture is the number of bits required to specify a given input-output function that can be implemented by the architecture. We derive mathematical results about the capacity of basic attention networks comprising linear or polynomial threshold gates. For instance, we show that the output gating of a linear threshold gate of n variables by another linear threshold gate of the same n variables has capacity $2n^2(1 + o(1))$. This proves that output gating is an efficient mechanism, since doubling the number of parameters leads to doubling the overall capacity, which is the maximum achievable. Perhaps surprisingly, multiplexing attention is used extensively in the proofs of these results.

Thus, as a roadmap, in Section 2 we first seek to identify and classify the most fundamental building block of all attention mechanisms within the standard deep learning model. In particular, we identify three key attentional mechanisms. In Section 3, we show how output gating and synaptic gating are used in all the current attention-based architectures, including transformers. In Section 4, we explore the functional capacity of output gating and synaptic gating. In Section 5, we provide a brief overview of the notion of capacity and the technique of multiplexing, which is a form of activation attention, for proving capacity lower bounds. In Sections 6 and 7, we prove several theorems about the capacity of activation, output, and synaptic gating, using multiplexing, first for single units (Section 6) and then for single layers (Section 7) of linear and polynomial threshold functions.

2. Sytematic Identification of Attention Quarks: Within and Beyond the Standard Model

We first introduce the formal neural network framework that we use in order to systematically organize and study the attention quarks, i.e., the most fundamental building blocks of attention. To borrow another term from physics, we call this framework the Standard Model.

2.1. **The Standard Model (SM).**

The Standard Model is the class of all neural networks made of what are generally called McCulloch and Pitt neurons. Neural networks in the SM consist of directed weighted graphs of interconnected processing units, or neurons. The synaptic strength of the connection from neuron j to neuron i is represented by a single real-valued number w_{ij}. Any non-input neuron i produces an output O_i by first computing an activation $S_i = \sum_j w_{ij} O_j$, i.e the activation corresponds to the dot product of the incoming

signal with the synaptic weights. In turn, the output of the neuron is produced in the form $O_i = f_i(S_i)$ where f_i is the transfer or activation function of neuron i. Typical activation functions include the identity function in the case of linear neurons, sigmoidal activation functions such as the logistic and tanh activation functions, and piece-wise linear functions ([25]), such as the Heaviside, sign, or ReLU functions. An encompassing, and more than sufficient, class of transfer functions for a formal definition of the SM is the class of functions that are monotonous and continuously differentiable everywhere except for a finite (and small) set of points [another alternative is the class of piece-wise linear functions with a finite (and small) number of pieces]. A fundamental, and easy to prove [6], property of the SM is that it has universal approximation properties: (1) Any Boolean function can be implemented exactly by a feed-forward network in the SM. This can easily be achieved using a feed-forward network, with a single hidden layer, of linear threshold gate neurons (which encompass the AND, OR, and NOT Boolean operations); and (2) For any small $\epsilon > 0$, any continuous function from $\mathbb{R}^n$ to $\mathbb{R}^m$ defined on a compact set C can be approximated within ϵ everywhere over C by a feedforward network in the SM. This approximation also can be achieved using a single hidden layer of linear threshold functions, and a linear output layer.

Several attention mechanisms described below can be viewed as extensions of the standard model, where new operations are added to the SM to obtain a richer model. Extending the SM is not a new procedure. For instance, using softmax layers is already an extension of the SM since the softmax is not a proper, single-neuron, activation function. Another example is the use of polynomial activation functions (e.g., [9]). Due to the universal approximation properties of the SM, these extensions are not meant to increase the approximating power of the SM. Rather, their value must be established along other dimensions, such as circuit size or learning efficiency. In the digital simulations of neural networks, these extensions correspond to new software primitives. In physical neural networks, these extensions must come with actual wires and physical mechanisms. For instance, a softmax operation is a new software primitive in a neural network software library but it requires a new physical mechanism for its physical implementation. It can be replaced by a network of depth 3 within the SM (Section 4) with fixed weights set to ± 1 (Figure 10), provided the logarithm and exponential activation functions are available.

2.2. **Systematic Taxonomy.** In the SM, there are only three kinds of variable types: S (activations), O (outputs), and w (synaptic weights). At the most fundamental level, we can organize attention mechanisms (and more broadly new SM interactions) depending on: the type of variable associated with the *source* of an attention signal (3 possibilities), the type of variable associated with the *target* of an attention signal (3 possibilities), and the *mechanism* of the interaction, i.e., on the algebraic operation used to combine the attending signal and the attended target. While many algebraic operations can be considered, the two most basic ones are addition and multiplication (2 possibilities). Note that both addition and multiplication are differentiable operations, and thus can easily be incorporated into the backpropagation learning framework. In total, this results into $18 = 3 \times 3 \times 2$ possibilities. We now discuss these possibilities, reducing them down to the 6 most important ones.

(1) **Source:** It is reasonable to assume that the source of the attending signal is a variable of type O corresponding to the output of one attending neuron, or a group (layer) of attending neurons. While the other possibilities can be explored systematically, e.g., a synapse directly attending another synapse or an activation attending another activation, in a physical implementation these would require new complex mechanisms. Furthermore, these do not occur in current attention-based deep learning

models. Equally unlikely would be the case of mixed schemes where the attending signal would emanate from, for instance, both neuronal outputs and synapses. In short, while other options can be studied, it is reasonable to assume that the attending signals emanate from neuronal outputs, allowing us to reduce the number of possibilities by a factor of three leaving 6 basic possibilities (Table 1).

(2) **Target:** For the target of an attention signal, we will study all three possibilities. Thus attention signals can target activations (S), outputs (O), or synapses (w). We will call these three forms of attention activation attention, output attention, and synaptic attention respectively.

(3) **Mechanism:** The most simple operations one can think of for combining the attending signal with its attended target are addition and multiplication. Attention requires excluding all other stimuli and possibly enhancing the attended stimulus (here we do not distinguish between external stimuli or internal representations). Intuitively, at the fundamental level, these exclusions and enhancements correspond to multiplicative operations where, for instance, the signals associated with non-attended stimuli are inhibited–i.e., multiplied by zero, and the attended stimuli are enhanced, i.e., mutliplied by a factor greater than one. We will reserve the term "gating" for multiplicative interactions. Thus, for instance, multiplicative synaptic attention will also be called synaptic gating. All multiplicative interactions, with the exceptions of terms of the form $w_{ij}O_j$, are not part of the SM and thus correspond to potential extensions of the SM. While multiplicative interactions seem the most important and novel ones, we will also consider the case of additive interactions. In particular, in the case of activation attention, for several common activation functions such as logistic or ReLU, inhibition (and thus suppression of stimuli) can be achieved additively by sending a large negative signal towards the attended neuron. Unlike gating, additive activation attention is contained in the SM. This is the kind of attention mechanism that will be used in the proofs of the theorems in Sections 6 and 7.

In the case of each attention mechanism, it is useful to keep in mind the distinction between digital simulations and actual implementations in a physical neural network, i.e., machine learning versus learning in the machine [6]. For instance, different mechanisms may be equivalent at the level of the algebraic expressions they lead to, but very different in terms of their physical implementations. As we shall see in Figure 2, both synaptic and output gating can produce the same quadratic term, but through different physical implementations. Finally, for completeness, two other important issues that could be taken into account in a taxonomy of basic attention mechanisms are multiplicity issues, at both the source and the target, as well as time scales issues, as briefly described below.

(1) **Multiplicities:** In each possible case, one must take into account multiplicity issues both at the level of the source and at the level of the target. For instance, in synaptic gating, can the attending output of a neuron gate more than one synapse? Can the attending output of several neurons gate the same synapse? And so forth. In the most simple cases, we will assume that the multiplicity is one both at the source and at the target, but greater multiplicities will also be considered, for instance in some of the theorems in Sections 6 and 7.

(2) **Time Scales:** Finally, for simplicity, and in line with current deep learning attention models, we assume that the attention mechanisms operate on the time scale of individual inputs. Different inputs create different attention signals. Alternative possibilities are briefly discussed in Section 2.5.

In summary, we are left with six main cases, corresponding to two different mechanisms $(+, \times)$ and three different target types (S, O, w). We now examine the three additive interactions and the three multiplicative interactions that are left, one by one, and show that they can be further reduced to three most important cases, which will be the focus of the following sections.

2.3. **Quark Identification: Additive Interactions.** In the case of additive interactions, the attention signal is added to three possible targets of type S, O, or w.

2.3.1. *Additive Activation Attention: Multiplexing.* In this case, consider an attended neuron i. It activation S has the form $S = S_1 + S_2$ where S_1 is the "normal" activation (without attention) and S_2 is the attending signal originated from one, or multiple, attending neurons. The terms multiplexing simply refers to the combination or superposition of two signals over the same channel. Depending on the transfer function f_i of neuron i, the attending signal can be used to control the output $O_i = f_i(S_1 + S_2)$. The typical case is when f_i is the logistic or Heaviside function: then a large negative signal S_2 (much larger than S_1) will override any S_1 and force the output of neuron i to be zero. If, on the other hand, $S_2 = 0$ then $O_i = f_i(S_1)$ and the normal signal will be propagated. If attention must be able to both suppress and enhance signals, this mechanism allows the suppression, but it does not provide a direct way for the multiplicative enhancement of signals. Formally this mechanism is entirely within the SM and does not require extending it. If the attending signal must come from a single neuron (source multiplicity one), this can easily be achieved by connecting the output of the attending neurons to a single linear neuron whose output is equal to S_2. Although not new, this attention mechanism is interesting because it will play a central role in the methods for proving various technical results about the new gating mechanisms presented below.

2.3.2. *Additive Output Attention.* In this case, using multiplicities of 1, we consider a neuron i connected to a neuron k in the main network, and an attending neuron j. In this case, the output O_j is simply added to the output O_i (Figure 1, Left) producing the terms $O_i + O_j$ (or $O_i + w_{ij}O_j$). This terms is nothing new in the SM and is equivalent to having an additional linear neuron with two incoming connections originating in neurons i and j, both with synaptic weight 1, and the same outgoing connections as neuron j in the original network. This mechanism alone does not provide much in terms of attentional functionalities and therefore it will not be considered here any further.

2.3.3. *Additive Synaptic Attention.* In this case, using multiplicities of 1, we consider a neuron i connected to a neuron k in the main network, and an attending neuron j. In this case, the output O_j is simply added to a synaptic weight, i.e., to w_{ki}. (Figure 1, Right), producing a new synaptic weight $w_{ki} + O_j$, which in turn creates a contribution equal to $(w_{ki} + O_j)O_i$ st neuron k. This contribution contains a new multiplicative term of the form O_iO_j which is not part of the SM. Since O_iO_j falls under the multiplicative category, it is subsumed by the analyses below of multiplicative interactions; thus additive synaptic interactions will not be considered any further in the rest of this work.

In summary, there are three kinds of additive interactions. Only multiplexing (additive activation attention) will be used in the rest of this work, and primarily as a tool in the proofs of some theorems.

2.4. **Quark Identification: Multiplicative Interactions or Gating.** In the case of multiplicative interactions, the attention signal is multiplied with three possible targets of type S, O, or w.

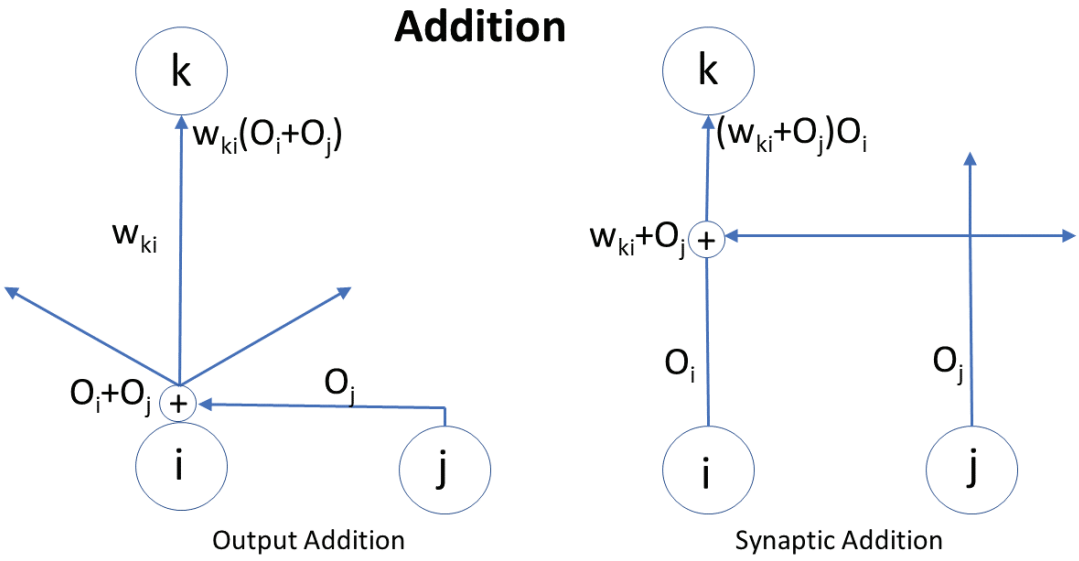

Figure 1. Additive Output (Left) or Synaptic (Right) Interactions. Left: the output O_j of the attending neuron is added to the output O_i of the attended neuron, producing a term of the form $O_i + O_j$. Right: the output O_j of the attending neuron is added to the attended synaptic weight w_{ki}, *de facto* producing a new multiplicative term of the form O_iO_j as one of the input components to neuron k.

2.4.1. *Multiplication Activation Attention: Activation Gating.* In this case, using a source multiplicity of 1, consider an attended neuron i with activation $S_i = \sum w_{il}O_l$ and transfer function f_i and an attending neuron j. In this case, the attending signal O_j multiplies the activation S_i so that the final output of neuron i becomes $O_i = f_i(S_iO_j)$. If f_i is sigmoidal or a threshold function, a large positive value of the attention signal O_j could be used to drive the response of neuron i towards one of its extreme values (e.g., 0/1 or $-/+$ [1]). Note that $O_i = f_i(S_iO_j) = f_i(\sum_l w_{il}O_lO_j$, so formally this mechanism is equivalent to having O_j multiply the output O_l of all the neurons connected to neuron i, although in a physical implementation these two things could be very different. Because of this equivalence, we will consider that output gating subsumes this mechanism and we will not discuss it much further. Furthermore, at least in the case of attended and attending neurons with threshold transfer functions equal to the sign function, multiplication of activation and multiplication of output are directly equivalent at the algebraic level because: $\text{sign}(S_iO_j) = \text{sign}(S_i)\,\text{sign}(O_j) = \text{sign}\, S_iO_j = O_iO_j$.

2.4.2. *Multiplicative Output Attention: Output Gating.* In this case, using multiplicities of 1, we consider a neuron i connected to a neuron k in the main network, and an attending neuron j. In this case, the output O_i is multiplied by O_j (or $w_{ij}O_j$) producing the quadratic terms O_iO_j which is new in the SM, leading to an input component into neuron k equal to $w_{ki}O_iO_j$ (Figure 2, Left). Note that while the multiplication is commutative, the attention

[1] Everywhere we write -/+ to indicate -1/+1.

mechanism is not in the sense that only the axon emanating from neuron i carries the signal O_iO_j to all the targets of neuron i.

2.4.3. *Multiplicative Synaptic Attention: Synpatic Gating.* In this case, using multiplicities of 1, we consider a neuron i connected to a neuron k in the main network, and an attending neuron j. In this case, the synaptic weight w_{ki} is multiplied by O_j. This produces a new synaptic weight $w_{ki}O_j$, which in turn also creates a contribution equal to $w_{ki}O_iO_j$ into neuron k.

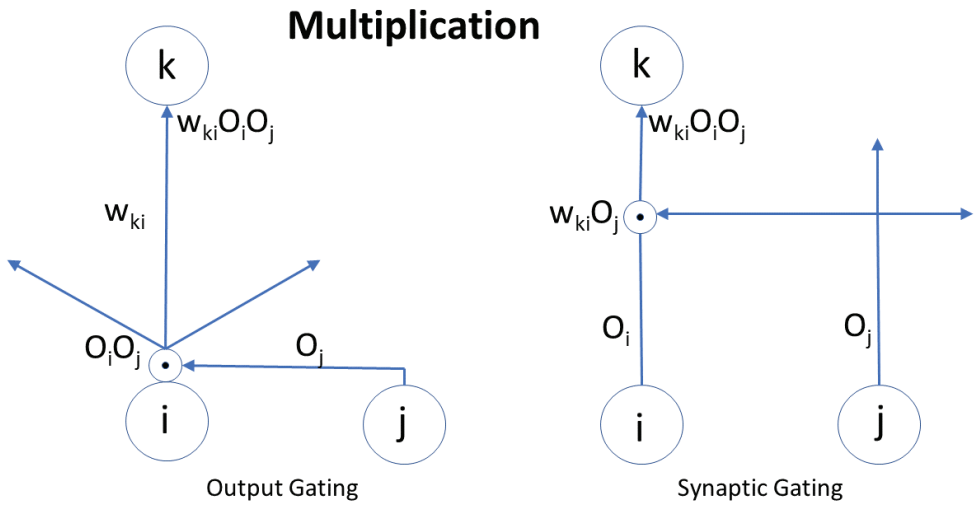

Figure 2. Multiplicative Interactions: Output and Synaptic Gating. Left: In output gating, neuron j gates the output of neuron i producing a new effective output O_iO_j. The signal O_iO_j is broadcasted to all the neurons downstream of neuron i, including neuron k. Right: In synaptic gating, neuron j gates the synapse between neuron i and neuron k, producing a new effective synaptic weight equal to $w_{ki}O_j$. In both cases, the signal O_j can be transmitted to other neurons and other synapses (higher multiplicity). In both cases, neuron k receives the same signal $w_{ki}O_iO_j$. However the effects of output versus synaptic gating on the rest of the network are different (see text).

2.5. **Synaptic Gating versus Output Gating.** When the gating signal O_j is close to zero, it will tend to suppress the gated signal O_i or the gated synaptic weight w_{ki}. The ability to dynamically suppress a synaptic weight or the signal flowing through it embodies the idea of "excluding other stimuli" associated with attention. Likewise, when the gating signal O_j is far from zero, it can dynamically enhance a synaptic weight or the signal flowing through it. Although equivalent circuits for output and synaptic gating can be found (see Figures 3 and 4), conceptually they are different.

Synaptic gating is a mechanism by which the gating neuron or network can dynamically change the synaptic weights of the gated neuron or network, thus effectively changing the program being executed by the attended network. This allows the same gated neuron or network to be modulated and to compute different functions, as a function of the gating neuron or network. Thus synaptic gating can also be viewed as a form of fast synaptic weight

mechanism [23, 3], where synapses with different time scales coexist and fast synapses are used to dynamically store information and modulate the function being computed by a given network. However, even for the fast synapses there could be different time scales. While here we assume that synapses change on the time scales of the inputs, fast synapses could also change on a lower time scale in the sense that a gated synapse could be reused over several inputs.

Although we have seen that both synaptic and output gating produce the same term of the form $w_{ki}O_iO_j$ at neuron k, this is true only for neuron k. Unlike synaptic gating, output gating affects all the neurons downstream of the gated neuron. In contrast, synaptic gating is more precise as it affects only the neuron downstream of the gated synapse, but it is more expensive, requiring one gating wire per gated synapse, rather than one gating wire per gated neuron. Nevertheless, if neuron i has only one outgoing connection, then gating of its output or its outgoing synapse are of course equivalent. For this reason, in the formal analyses, we will focus on output gating which covers also synaptic gating under the assumption of a single outgoing connection per gated neuron.

An observation that will become important in Sections 4– 7, is that in the case of binary units and output gating, it does make a difference whether one uses $0/1$ or $-/+$ representations. In particular, although $0/1$ or $-/+$ linear (or polynomial) threshold functions are equivalent, different forms of output gating are obtained with different combinations of such units. This is because multiplication of $x \in \{-1, 0, 1\}$ by 0 or by -1 leads to different results. In particular, multiplication of the outputs of two $0/1$ threshold gates is equivalent to applying a logical AND operation, whereas multiplication of two $-/+$ linear threshold gate is equivalent to applying a logical NXOR (the negation of an XOR). Multiplication of a $0/1$ threshold gate by a $-/+$ threshold gate produces a non-Boolean functions with outputs in $\{-1, 0, 1\}$. Nevertheless in many cases equivalent circuits can be found (see Example in Figure 7) using either multiplication between $0/1$ threshold gates or multiplication between $-/+$ threshold gates. This also suggests a more general question of studying all possible ways of combining two threshold functions using Boolean operators (see Section 6).

2.6. **Relations to Polynomial Neural Networks.** There are at least two important relationships between gating and polynomial neural networks. First, we have seen that both synaptic and output gating mechanisms produce quadratic terms of the form $w_{ki}O_iO_j$ contributing to the activation of neuron k. Thus gating can also be viewed as a special case of neurons with quadratic activations or, more generally, polynomial activations [9]. However, a full quadratic activation function of n inputs may need $n(n-1)/2$ 3-way synaptic weights (the quadratic component of the activation of a neuron i has the form $S_i = \sum_{jk} w_{ijk}O_jO_k$) associated with each possible pair of inputs. Synaptic gating or output gating produce only one new quadratic term. Thus, in short, gating creates quadratic terms but in a sparse way that avoids the combinatorial explosion associated with all possible combinations.

The second connection is that the same gating concepts can be applied to to more complex units, beyond the standard model, in particular to units where the activation is a polynomial function of degree d of the inputs (the standard model corresponds to $d = 1$). Thus for instance a neuron j with a quadratic activation function could gate the output of another neuron i with quadratic activation functions, or gate a synapse w_{ki} between neuron i and neuron k. Gating by neurons with polynomial activations, in particular gating by polynomial threshold units, will be studied in Sections 6 and 7.

2.7. **Summary.** In summary, the quarks of attention can be classified based on the origin, the target, and the interaction mechanism of the attention signal. Assuming that the origin

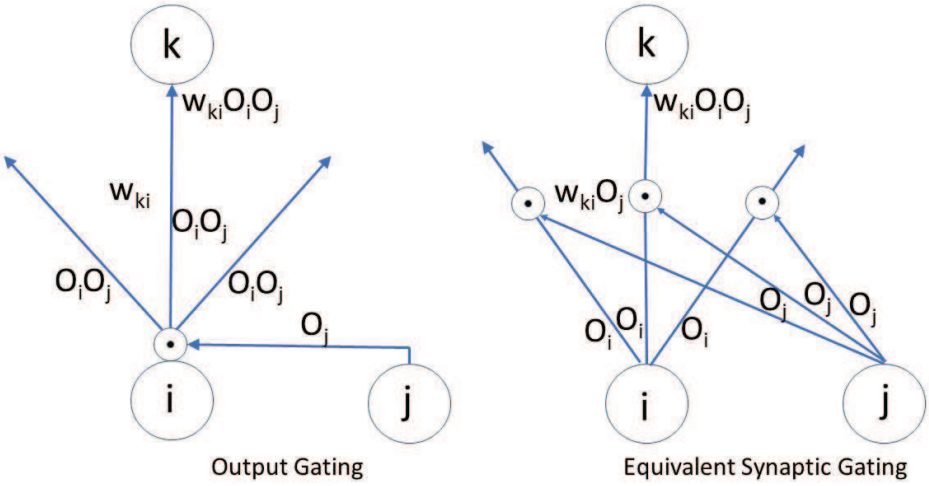

Figure 3. Output gating equivalence. Left: Output gating of neuron i by neuron j. All the connections emanating from neuron i carry the signal O_iO_j. Right: Equivalent network obtained using synaptic gating only. The gating neuron j must synaptically gate all the connection weights emanating from neuron i

.

TABLE 1. Organization of attention mechanisms. Assuming that the origin of the attention signal is the output of one or several neurons, there are 6 classes depending on the target of the signal and the interaction mechanism. We consider 3 kinds of targets: activation (S), output (O), and synapses (w). We consider 2 kinds of interaction mechanisms: addition and multiplication. Two of the classes (additive activation attention, or multiplexing, and additive output attention) are in the SM; the other 4 classes correspond to true extensions of the SM. The discussion in the text shows that further analyses can focus on three classes only: multiplexing, output gating, and synaptic gating (in bold).

	S	O	w
Addition	**multiplexing** (SM)	additive output att.(SM)	aditive synaptic att.
Multiplication	activation gating	**output gating**	**synaptic gating**

is in the output of one neuron, or a group of neurons, and that the interactions are either additive or multiplicative, this leads to six classes (Table 1). Within the additive group, two classes are already in the SM (additive activation and additive output attention) and only one class is of interest here for further studies (additive activation attention or multiplexing). Within the multiplicative group, all three classes correspond to true extensions of the SM and, at least formally, further analyses can be reduced to two main classes: output gating and synaptic gating. In all cases, the attending signal modulates the function computed by the attended network.

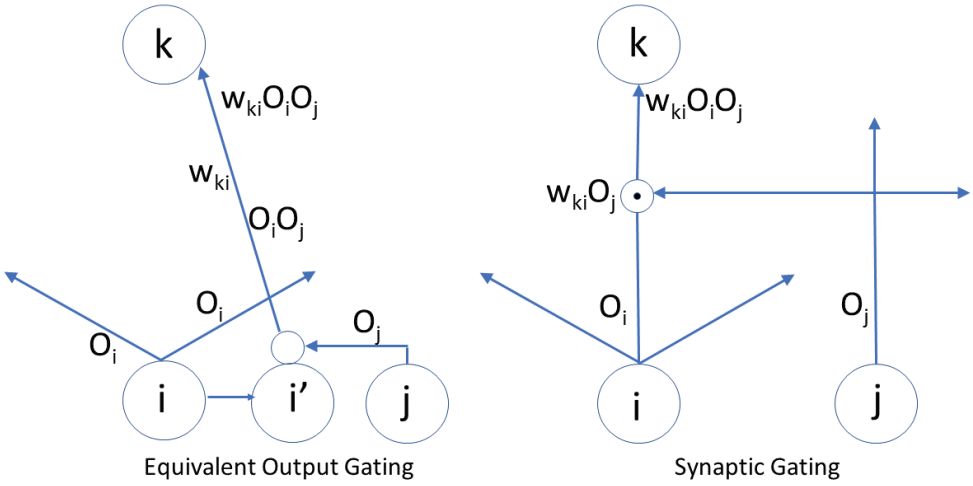

Figure 4. Synaptic Gating Equivalence. Right: Synaptic gating of synaptic weight w_{ki} by neuron j. Left: Equivalent network obtained using output gating only. Neuron i has an identical twin neuron i', i.e., neuron i connected to neuron i' through the identity function and thus both neurons produce the same output equal to O_i. Neuron j output gates neuron i' producing a signal O_iO_j which travels through the synapse w_{ki}. All other connections emanating from neuron i carry the signal O_i and are unaffected by the gating neuron j.

3. Transformers: All you Need is Gating

Although the descriptions of attention-based architectures in deep learning are often complex and tend to obfuscate the underlying neural architectures [14, 10, 19, 4, 11, 22], it can be checked that in all cases these architectures are built using the output and synaptic gating operations described in the previous section. For conciseness, here we demonstrate this fact in detail for the transformer architectures [26] (see also [18] for an MLP alternative to transformers). Transformers are typically described in terms of Query, Key, and Value vectors. For clarity, we briefly describe the inspirational meaning attached to these words originating from the area of databases.

3.1. Query, Key, and Value.

In a database, a key is a single attribute, or a group of attributes, that can uniquely identify each record (row) in a table. A key is used to establish relationships between tables. For instance, a database of car drivers may include the following attributes: first name, last name, home address, driver license number, car model driven, and license plate. The driver license number, or the name combined with the license plate are examples of keys. In a traditional database, when a properly formatted query comes in, it is compared with all the keys. When a perfect match is found, the content of the corresponding row (the value) is retrieved. This basic schema can be extended in several directions, which are used by transformers. First, the query and the key may not be properly formatted in the same "space" and a linear (or possibly non-linear) transformation may be used to project them in the same space, where they can be properly compared. Second different notions of similarity (kernels) may be used to compare them, as opposed to a binary identity score.

One of the most simple similarity comparisons can be done via a dot product, or a scaled dot product. And third, the value to be retrieved can be constructed from a combination of multiple values and multiple similarities. The most simple approach is to combine all the values linearly, using the similarities of the query to the corresponding keys to weigh each contribution. When the similarities form a probability distribution (via a softmax) as in the case of transformers, the ouputs are convex combinations of the input values. Armed with these notions, we can now take a close look at transformers and their attention mechanisms.

3.2. **Transformers and Gating.** Transformer architectures consist of stacks of similar encoder and decoder modules, with attention mechanisms in each module. The details of an encoder module are shown in Figure 5. As the Figure shows, a shared and typically linear network is first applied to each of n input vectors. At the bottom of the architecture, these input vectors could represent for instance vectors encoding successive words from a sentence. and can include positional information, as discussed below. At higher levels of the stack, these vectors could be associated with the outputs of the previous encoder or decoder module and correspond to more abstract representations, or to final outputs. For each input vector, the shared network typically produces a triplet of vectors of the same size m: Q (Query),K (key), and V (value), for a total of $3n$ triplets. The subsequent attention mechanism is drawn in a concise way in the Figure and is based on three operations: (1) taking all n^2 pairwise dot products of the n query vectors with the n key vectors; (2) applying a softmax to each row of dot products; and (3) using the output of the softmax operations as weights for linearly combining the value vectors to produce the corresponding output vector at each position. The first operation can be built using output gating, each dot product involving m gating operations, to multiply the proper Q and K components together. As a side note, these dot products can be viewed as similarity measures between the Q and K vectors, especially when these are normalized, and this suggests other kinds of transformer architectures where different similarity kernels are used. The softmax operation is a standard extension of the SM (Figure 10). The third operation corresponds to synaptic gating of the connections between the V vectors and the outputs. The convex combination of the value vectors by the corresponding softmax weights determines how much each value vector influences each output vector, based on the corresponding similarities between Q vectors and K vectors. This is where the influence of some of the value vectors can be enhanced, while the influence of others can be suppressed. Thus in total there are mn^2 output gating operations, and n^2 synaptic gating operations (assuming n output vectors). Thus, in short, the entire encoder module is based on a large number ($O(mn^2)$) of gating operations, both of the output and synaptic type. Thus, in this form, it can only be applied when n is not very large. The basic transformer decoder module (not shown) is very similar. One important property of the encoder module conferred by the attention mechanisms is that the output is invariant under permutation of the inputs. This is because any permutation of the inputs, results in a corresponding permutation of the Q,K, and V vectors due to the weight sharing. This in turn induces a corresponding permutation in the dot products and softmax outputs, so that in the end the weighted contribution of any V vector into any output vector remains the same. This permutation invariance may seem surprising for an architecture that was originally developed for NLP tasks, where the order of the words obviously matter. Indeed, in NLP applications of transformers, positional information is added to each input vector. However, the permutation invariance of transformers is particularly beneficial for applications of transformers outside of NLP, in particular applications where the input consists of *sets* of data vectors, where the order of the data vectors does not matter (e.g., [17, 13, 24]). For

example, in physics applications one often has to deal with a set of measurements, rather than a sequence of measurements; likewise, in chemistry, the order of the reactants or the products in a chemical reaction does not matter.

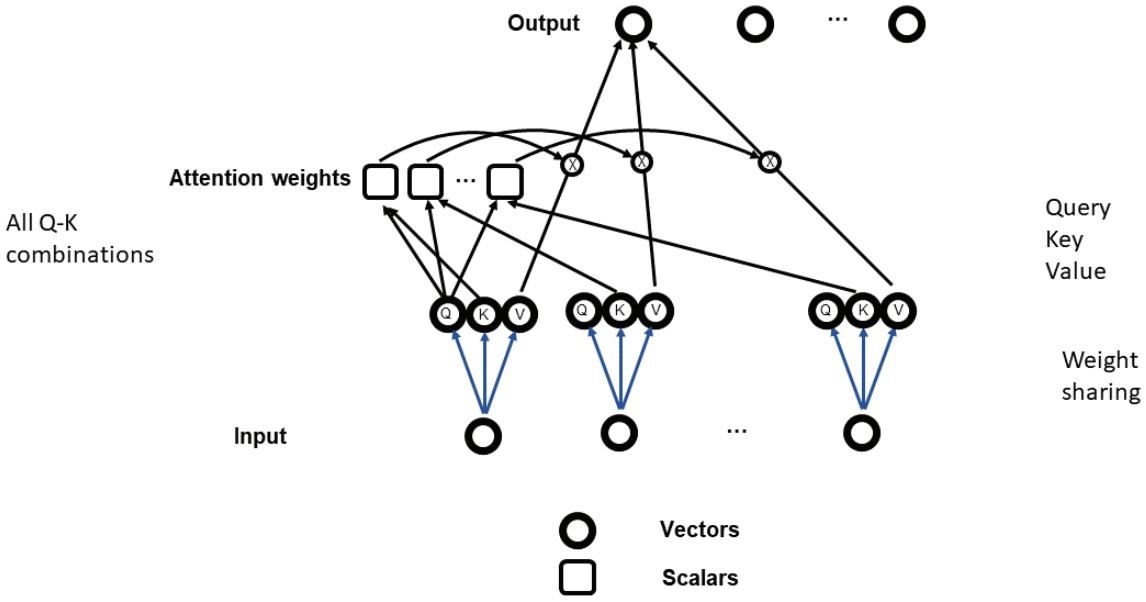

Figure 5. Neural network representation of the basic encoder module of a transformer architecture. Each input vector is converted into three vectors Q (Query), K (key), and V (value) using weight sharing (blue weights). All n^2 pairwise dot products $K(k)Q(l)$ are computed in the attention layer, which corresponds to a set of output gating operations. This is followed by row-wise softmax operations on these dot products to produce the weights that are used to linearly combine the value vectors into each corresponding output. These linear combinations correspond to synaptic gating.

3.3. **Transformers and Recurrent Neural Networks.** From a high level, transformer architectures are not that different from the recurrent or recursive neural network (RNN) architectures that were used previously for NLP tasks. By unfolding RNNs in time, it is easy to see that both kinds of architectures: correspond to feed-forward architectures; are capable of seeing the entire input sequence; use weight sharing; use attention mechanisms; and use some form of local connectivity. The major differences seem to reside in the details of the attention blocks with the use of Query, Key, and Value vectors in transformers and in the way positional information is handled. In transformers, positional information is added dynamically to the input vectors encoding each word, and information from the entire sequence is readily available already in the first layers of the architecture. In RNNs, positional information is not added to the inputs, but is present statically in the connectivity: sequential

words are connected to deeper and deeper layers of the architectures and information about the entire sequence is available only to the deeper layers.

4. Functional Aspects of Attention

Next we study though several examples how certain functionalities can be implemented using attention mechanisms, beginning with the effect of attention on single units.

4.1. **Single Unit Output Gating: Shaping the Activation Function.** First, for simplicity, we consider output gating of a unit by another unit with the same inputs and the same weights, hence the same activation S. The two units may have two different activation functions f and g. Through output gating, the final output of the gated unit will be given by: $f(S)g(S) = fg(S)$. Thus, in this case, output gating is equivalent to changing the activation function of the gated unit from f to fg. Examples of this effect are shown in Figure (Figure 6) where both f and g are piecewise linear, and centered at the origin. Note that in the case of a linear unit gated by another linear unit, the final output is a quadratic function of the n inputs, but with only $O(n)$ parameters as opposed to $O(n^2)$. The ReLU activation function emerges naturally, through the gating of a linear function by a $(0, 1)$ threshold function, or vice versa. Finally, the symmetric wedge activation function [25] emerges also naturally through the gating of a linear function by a $(-1, 1)$ threshold function.

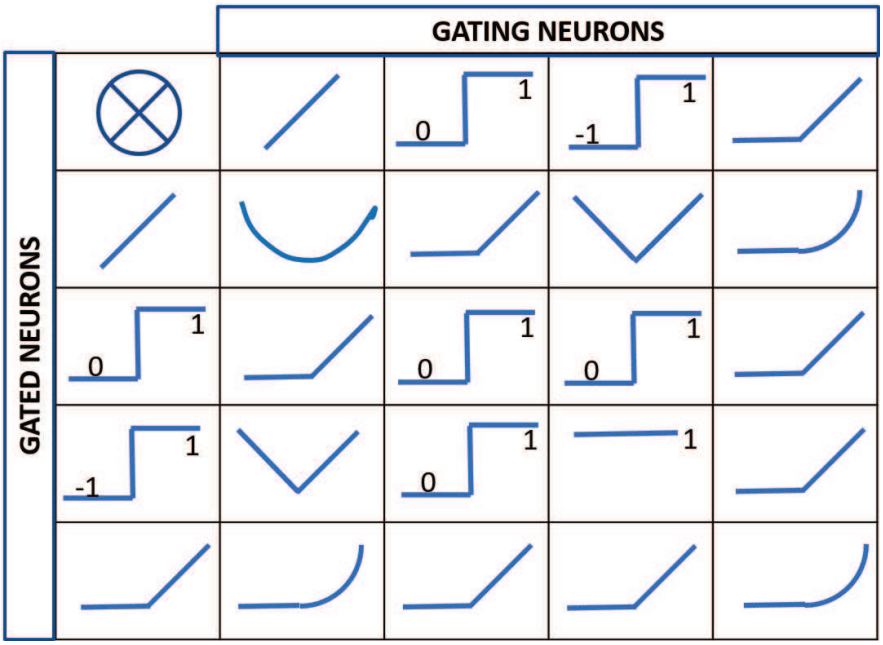

Figure 6. Effect of gating on activation functions. For simplicity we consider four main activation functions: linear, threshold (0,1) [Heaviside function], threshold (-1,1) [sign function], and ReLU.

4.2. **Single Unit Attention: XOR.** Next, we look at the simple XOR function. It is easy to show that the XOR function cannot be computed in a shallow way by a single linear threshold gate (or sigmoidal) neuron. Its computation requires at least one hidden layer. However, as shown in Figure 7 using 0/1 outputs, the XOR function can be computed by a

shallow network with a single linear threshold unit output-gated by another linear threshold unit. To see this, note that any corner of the hypercube can always be isolated by a hyperplane from the other corners of the hypercube, i.e., there is always a linear threshold gate that has value 1 (resp. 0) for one Boolean setting of its inputs, and 0 (resp. 1) for all the other possible inputs (see Lemma 5.1). In particular, both the OR and NAND functions are of this kind and thus can be implemented by a linear threshold gate. Gating the OR by the NAND (or vice versa) produces the desired XOR function without using any hidden layers, assuming that the ouput gating operation is an integral part of the layer where it occurs.

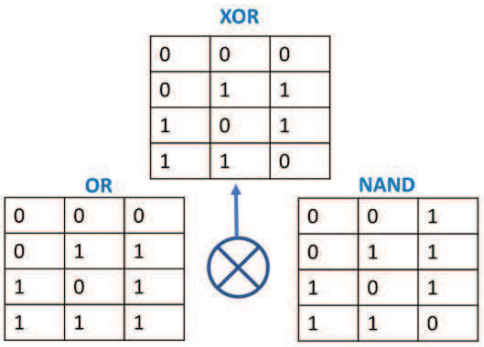

Figure 7. Shallow computation of XOR by a single unit with attention. The left unit computes an OR, which can be implemented by a single linear threshold gate. The right unit computes a NAND, which can also be implemented by a single linear threshold gate. The gating of one unit by the other produces the XOR function. The XOR function cannot be implemented using a shallow (no hidden layer) network of linear threshold gates. In this particular examples, the gated and the gating $0/1$ units can easily be replaced by $-/+$ units, since it is always possible to linearly separate any point on a hypercube from all the other points with a hyperplane, and the component wise product of $(-1,1,1,1) \times (1,1,1,-1)$ gives $(-1,1,1,-1)$, which corresponds to the $-/+$ version of XOR.

4.3. **Attention Layers: Universal Approximation Properties.** Next, we look at how universal approximation proofs are affected if output gating is allowed, both in the Boolean and continuous cases.

4.3.1. *The Boolean Case.* Every Boolean function of n variables can be computed by a feed-forward network of linear threshold gates, since AND, OR, and NOT can be implemented by linear threshold gates. By expressing the function in disjunctive or conjunctive normal form, the implementation can be achieved with a single hidden layer of exponential size. If we allow output gating, and its iterations, we have the following theorem.

Proposition 4.1. *Every Boolean function of n variables can be expressed as the product of at most 2^{n-2} linear threshold gates, both in the $0/1$ and $-/+$ representations.*

Proof. Let f be a Boolean function of n variables, using $0/1$ to denote false and true respectively. If f is 0 everywhere, it can immediately be expressed as a linear threshold gate.

Likewise, if f is 0 everywhere but one point of the n dimensional cube, then it can be immediately expressed as a single linear threshold gate. Thus we can assume that f is 0 on at most 2^{n-2} points. Let $x_1, \ldots, x_L (L \leq 2^{n-2})$ denote the inputs where f is zero. For each index i, let g_i denote the linear threshold gate which has value 0 on x_i and 1 everywhere else. Then it is easy to check that $f(x)$ can be written as the product $f(x) = f_1(x) \ldots f_L(x)$ (alternatively, on can express f in conjunctive normal form). The proof is the same in the $-/+$ case, letting $g_i(x)$ be the linear threshold gate with value -1 for x_i, and $+1$ everywhere else. Obviously the same result holds for polynomial threshold gates of degree d. □

In the $-/+$ case, the set B_n of all Boolean functions with the multiplication operation forms a commutative group, and each Boolean function is its own inverse. The subset of all linear threshold gates contains the identity, and each linear threshold function gate is its own inverse. However it does not form a subgroup because it is not closed. By the theorem above, the multiplicative closure of the set of all linear threshold gates is the set B_n of all Boolean functions.

Since every Boolean functions can be written as a product of an exponential number of linear (or polynomial) threshold gates, it is natural to ask whether a smaller number of factors may be used. Can every Boolean function be written as the product of a linear or polynomial number of linear threshold gates? We will answer this question negatively in Section 6.1.1.

4.3.2. *The Continuous Case.* Next we look at the continuous case, using output gating to modify the basic universal approximation proof [6].

Theorem 4.2. *Let f be a continuous function from $[0, 1]$ to $\mathbb{R}$, and $\epsilon > 0$. Then there exists an integer $n = n(\epsilon)$ such that f can be approximated within ϵ everywhere over $[0, 1]$ by a network of n linear units, attended by n corresponding linear threshold gates with output gating. The final approximation corresponds to the dot product between the vector of linear unit outputs and the vector of attending unit outputs.*

Proof. Since f is continuous over the closed interval, it is uniformly continuous so that there exists $\delta > 0$ such that for any x_1 and x_2 in $[0, 1]$:

$$|x_2 - x_1| < \delta \Rightarrow |f(x_2) - f(x_1)| < \epsilon \tag{4.1}$$

Let us choose an integer n large enough so that $\delta > 1/n$. Next we slice the interval $[0, 1]$ into n slices of width $1/n$. Next we construct a network with n linear units and n linear threshold gate attention units with outputs in $\{0, 1\}$ (the proof can be adjusted to accommodate outputs in $\{-1, 1\}$). All the attention units are connected to the single input x by a weight equal to 1. Their threshold (bias) however are $0, 1/n, 2/n, \ldots, (n-1)/n$ so that when $x \in [0, 1/n)$ only the first attention unit is on, when $x \in [1/n, 2/n)$ only the first two attention units are on, and so forth. In other words, the slice containing x is encoded in the number of linear threshold gates that are on. For the linear units, they compute values $y_1(x), \ldots, y_n(x)$ as follows. The first linear unit approximates the function f in the first slice by producing the line that goes through $f(0)$ and $f(1/n)$, i.e., by implementing the function $y_1(x) = f(0) + n[f(1/n) - f(0)]x$. The second linear unit approximates the function f in the second slice by producing the line that goes through $f(1/n)$ and $f(2/n)$, but with the subtraction of the value produced by the previous unit. Thus in short: $y_2(x) = n[f(1/n) - f(0)]x - y_1(x)$. More generally, the output of the k-th linear unit approximates the function f in the k-th slice producing the line that goes through $f(k - 1/n)$ and $f(k/n)$, but with the subtraction of y_{k-1}. [Note: as an alternative construction, the linear units could also be taken to be constant, with $y_k(x) = f(k - 1/n) - y_{k-1}(x)$, and $y_1(x) = f(0)$.]

□

The same construction can be applied over any closed interval, as well over any finite union of closed intervals. Furthermore, if the range is $\mathbb{R}^p$, the same construction can be applied to each component. And finally, the same construction can be generalized if the input domain is of the form $[0,1]^m$. Thus in short:

Theorem 4.3. *Every continuous function f from a compact set $C \subset \mathbb{R}^m$ to $\mathbb{R}^p$ can be approximated to any degree of precision ϵ by a shallow attention network comprising linear units gated by corresponding linear threshold gate units, with a final dot product output.*

4.4. **Attention Layers: Dot Products.** As we have seen in the section on transformers and the universal approximation proof above, one place where attention mechanisms are particularly important is for computing the dot product between two activity vectors $u = (u_1, \ldots, u_n)$ and $v = (v_1, \ldots, v_n)$, associated with two corresponding layers of n neurons each. This can be achieved through output gating to first compute all the pairwise products $u_i v_i$ and then combine these products through a single linear output unit, with all its incoming weights set to 1, to compute the dot product $uv = \sum_i u_i v_i$. However, this dot product can equally be computed by synaptic gating, i.e., by using the vector v to gate the incoming weights of the linear unit above and compute the dot product in the form $uv = \sum_i (1.v_i)u_i$. This can be scaled up to tensors where there are multiple output vectors $u(k) = (u_i^k)$ and multiple attention vectors $v(l) = (v_i^l)$ of the same length, and all pairwise dot products $u(k)v(l)$ are computed, for any (k,l) pair, as in the transformer architectures. Of course the dot product can also be computed in the standard model (Figure 8) however this requires a deeper network with four layers of standard units with fixed connections all equal to 1, and both logarithm and exponential transfer functions. Thus output or weight attention create a new primitive, or compact circuit, for computing dot products. The same is true of other operators that are often introduced in neural network without being part of the standard model, such as for the already-mentioned softmax (Figure 10) or the normalization of a vector (Figure 9).

4.5. **Attention Layers: Attention Weights.** Synaptic gating of a connection can suppress or enhance the corresponding incoming signal. Synaptic gating all the incoming edges of a unit allows to assign different importance to its different inputs. In addition, it is often desirable that these degree of importance form a probability vector, as in the transformer architecture, and this can be achieved through a softmax operation. It is possible to apply a normalizing softmax either to the vector of pairwise products $u_i v_i$, or to the rows or columns of the tensor of dot products $u(k)v(l)$, as in the transformer architectures. The output of these softmax operations can then be used to gate other synaptic weights. These gated weights are often equal and set to one in order to compute convex combinations, as in the transformer architecture Thus, in short, in transformer and other architectures, attention mechanisms allow dot products, softmax, and synaptic gating operations to be combined into one macro operation, which would require a network of depth ~ 10 for its implementation inside the SM.

5. Cardinal Capacity Review

We have seen that attention mechanisms enable important functionalities with minimal depth compared to the equivalent SM circuits, at the cost of adding attention neurons and mechanisms. Here we want to better understand the trade offs between the computations

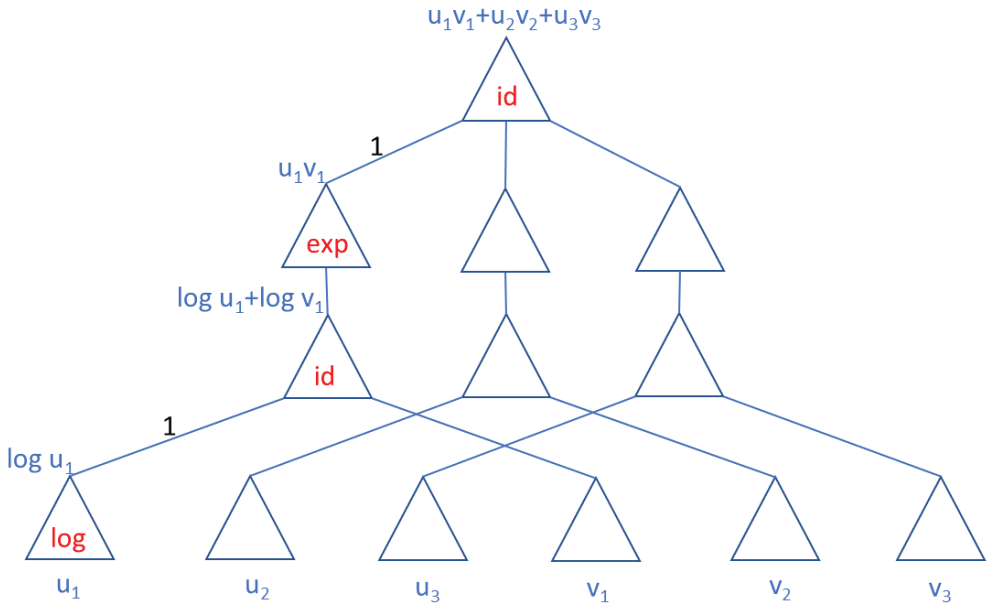

Figure 8. Standard model neural network for computing the dot product of two vectors (u_1, u_2, u_3) and (v_1, v_2, v_3) .The sub-circuit computing $u_i v_i$ corresponds to the implementation of output gating of u_i by v_i in the SM.

that are enabled and the corresponding costs. The key concept for doing so is the concept of cardinal capacity [8] which we briefly review below.

5.1. **Definition of Capacity:** Given a class of functions $\mathcal{A}$, we define its cardinal capacity $C(\mathcal{A})$, or just capacity, to be: $C(\mathcal{A}) = \log_2 |\mathcal{A}|$, where $|\mathcal{A}|$ is the cardinality of $\mathcal{A}$ in the finite case. In the continuous case, $|\mathcal{A}|$ can be defined as a volume, but here we will focus primarily on finite cases. The class $\mathcal{B}_n$ of all Boolean functions of n variables has capacity $C(\mathcal{B}_n) = 2^n$. Here we will consider subclasses of $\mathcal{B}_n$, in particular those implemented by feed-forward networks of linear or polynomial threshold gates, with attention mechanisms, and compute the corresponding capacity. Using linear or polynomial threshold functions is not particularly restrictive since these are reasonably good approximations of linear- or polynomial-activation neurons with steep sigmoidal transfer functions. Furthermore, the universal approximation properties of the SM can be established while using only linear (or polynomial) threshold functions in the hidden layers.

5.2. **Linear and Polynomial Threshold Functions.** A polynomial threshold functions of degree d has the form $\operatorname{sgn} p(x)$, where $p(x)$ is a polynomial of degree d using a $-/+$ output representation. Alternatively, for a $0/1$ output representation, we can use the form $H(p(x)$ where H is the Heaviside function equal to 0 for $x \leq 0$ and to 1 otherwise. Units with values in $0/1$ are similar to logistic sigmoidal units, and units with values in $-1/+1$ are similar to tanh sigmoidal units.

We let $\mathcal{T}(n; d)$ denote the class of polynomial threshold functions of degree d. Thus $\mathcal{T}(n; 1)$ denotes the class of linear threshold functions. When the inputs to a threshold function are binary, we use the term threshold gate. In the case of polynomial threshold gates, it does

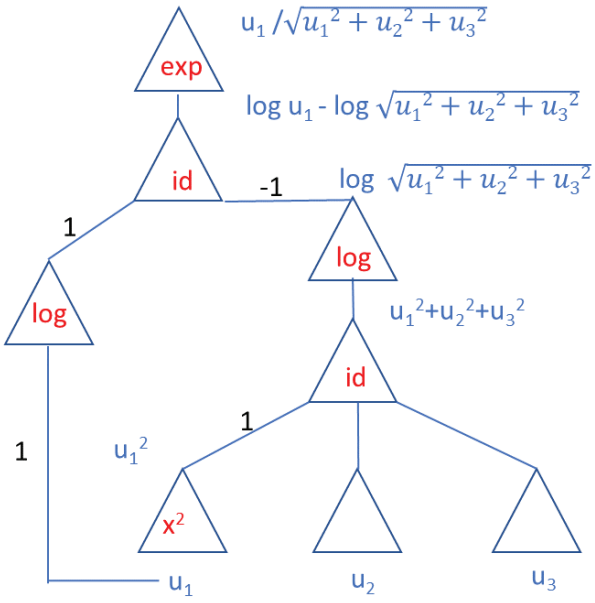

Figure 9. Standard model neural network for normalizing a vector (u_1, u_2, u_3) (for clarity, only the first normalized component is fully shown).

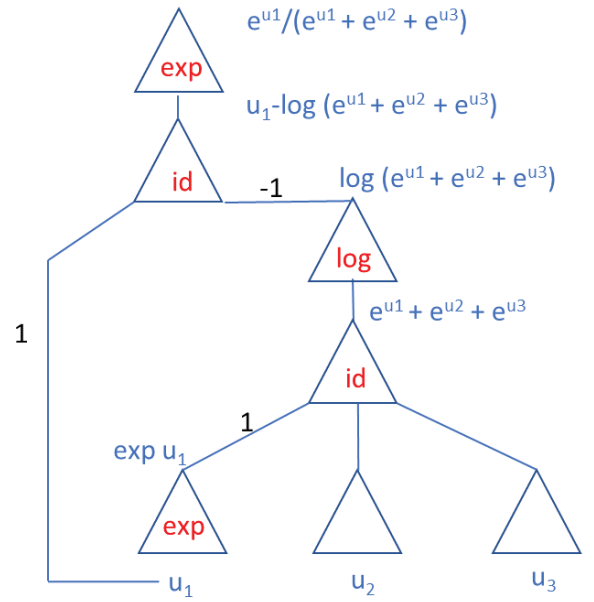

Figure 10. Standard model neural network for computing the softmax function for a vector (u_1, u_2, u_3)(for clarity, only the first component is fully shown).

not matter whether their input is encoded using $0/1$ or $-/+$ (or for that many any two

distinct real numbers). This is because there is an affine transformation between any two such encodings and the affine transformation can be absorbed in the synaptic weights, i.e., the coefficients of p. The same is generally true for the encoding of the output, however when attention gating is considered the $0/1$ and $-/+$ encodings behave differently. For instance, in the case of output gating, the product of two $0/1$ threshold gates behaves like an AND, whereas the output of two $-/+$ gates behaves like an NXOR.

Thus to derive more general results, we will consider the case where the gating mechanism is implemented by a Boolean function B, which could be an AND, an NXOR, or something else. In the most general setting, we let $B(z_1, \ldots, z_k) : \{-1, 1\}^k \to \{-1, 1\}$ be a Boolean formula in k variables. We are interested in the class of functions of the form $B(f_1, ..., f_k) : \{0, 1\}^n \to \{-1, 1\}$ where $f_j \in \mathcal{T}(n; d_j)$. We denote this class by $\mathcal{T}_B(n; d_1, \ldots, d_k)$.

5.3. **Why Capacity is Important.** The capacity $C(\mathcal{A})$ is a measure of what the class of functions $\mathcal{A}$ can do. As a single number, it is of course a very crude representation of the true functional capacity. However in the case of neural networks the capacity has a stronger significance. To see this, note first that the cardinal capacity is also the number of bits required to specify an element of $\mathcal{A}$. Thus in the case of neural networks, to a first order of approximation, the capacity is the number of bits that must be transferred from the training data to the synaptic weights during learning for the network to learn to implement a specific function in the class $\mathcal{A}$.

5.4. **Capacity of Single Units: Review.** Before we estimate the capacity of single units with attention mechanisms, we must review the known capacity results on single units without attention mechanisms. For a single linear threshold gate of n variables, we have [29, 30]:

$$\left(1 - \frac{10}{\log n}\right) n^2 \leq C(\mathcal{T}(n; 1)) \leq n^2 \tag{5.1}$$

This result was refined to the form [16]:

$$C(\mathcal{T}(n; 1)) = n^2 - n \log_2 n \pm O(n) \tag{5.2}$$

Similar results have been obtained for polynomial threshold gates of degree d [5, 9]. In particular, for any n and d satisfying $1 \leq d \leq n^\alpha$ (where α is fixed and $< \alpha < 1$) there exists a constant $D = D(\alpha)$ such that:

$$(1 - \frac{D}{\log n})^d n \binom{n}{\leq d} \leq C(\mathcal{T}(n; d)) \approx n \binom{n}{\leq d} \tag{5.3}$$

where:

$$\binom{n}{\leq d} = \sum_{k=0}^{d} \binom{n}{k} \tag{5.4}$$

For degree $d = o(\log n)$, including fixed degree d, Equation 5.3 yields:

$$C(\mathcal{T}(n; d)) = \frac{n^{d+1}}{d!}(1 - o(1)) \tag{5.5}$$

5.5. **Activation Attention and Muliplexing.** We now describe one of the main techniques that will be used in the attention capacity proofs for both synaptic and output gating. Perhaps surprisingly, this technique can be viewed as a form of attention, specifically a form of activation attention or multiplexing. It was developed and used in [7, 8]. First, we need the following lemma, stated for the 0/1 n-dimensional hypercube, but equally valid on the -/+ hypercube, or any other hypercube $[a, b]^n$. The lemma basically states that any vertex of the hypercube can be separated from the rest of the cube by a hyperplane with arbitrarily large margins.

Lemma 5.1. *Let H be the n-dimensional hypercube, and $M > 0$ and $K \geq 0$. Fix any vertex $c = (c_1, \ldots, c_n)$ of the hypercube, and let $D = H - \{c\}$. Then there exists affine linear functions of the form $f(x) = a_0 + \sum_1^n a_i x_i$ such that: $f(c) = K$ and $f(d) \leq -M$ for any $d \in D$.*

Proof. First note that there are 1:1 affine maps between the different hypercubes, thus it is enough to prove the result for the 0/1 hypercube. Second, all the corners play a symmetric role so it is enough to prove it for the corner $c = (1, 1, \ldots, 1)$. It is easy to check that: $f(x) = \sum_1^n (M + K)x_i - (M + K)n + K$ satisfies the conditions of the Lemma. Note that by using $-f(x)$ the sign of the regions and corresponding margins can be exchanged ($f(c) = -K$ and $f(d) > M$ for all $d \in D$). See Figure 11. □

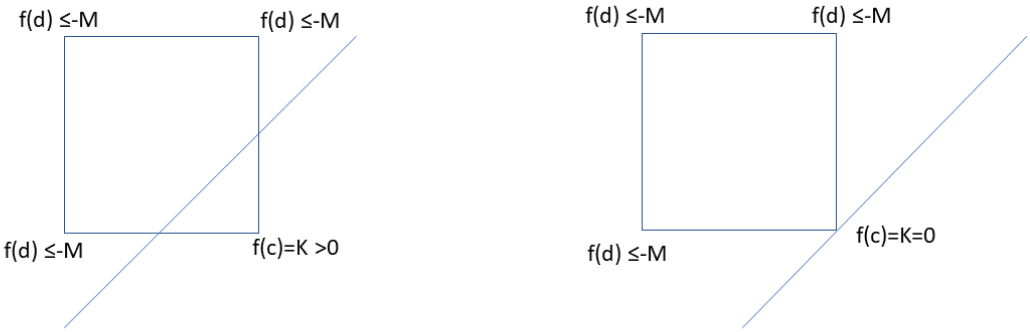

Figure 11. Any corner c of the n-dimensional hypercube can be separated from all other corners d of the hypercube by an affine hyperplanee with large margins defined by the parameters $K \geq 0$ and $M > 0$.

Now consider a neural network consisting of n inputs fully connected to a hidden layer of m linear or polynomial threshold functions (Figure 12) $f_0(x), \ldots, f_{m-1}(x)$. In the multiplexing approach, we add m, or even just $\lceil \log_2 m \rceil$ new binary inputs to the input layer. m different binary patterns over these inputs can be associated in one to one fashion with one of the m threshold functions in the hidden layer. Let i be any integer $0 \leq i \leq m$ and let $p(i)$ denote

the corresponding pattern of bits. For simplicity we can just use the binary representation of i, but any other representation works equally well.

This pattern $p(i)$ can be viewed as a corner of the corresponding hypercube of dimension $\lceil \log_2 m \rceil$ and thus we can apply Lemma 5.1 above to choose the weights connecting the attention units and the bias to hidden unit i accordingly. In particular, the weights can be chosen such that: (1) the attending signal originated from the attending bit patterns $p(i)$ is equal to 0; and (2) for all other settings of the attending bits, the attending signal is arbitrarily large and negative (alternatively arbitrarily large and positive). Ans similarly for all the other units and attention input patterns. As a result, whenever $p(i)$ appears in the attention bits, the i-th output of the hidden layer is equal to $f_i(x)$, and for all the other settings of the attention bits, the $i-th$ output is constantly equal to 1, or constantly equal to 0 (or -1 in the case of $-/+$ threshold hidden units). The pattern of constant bits is called the mask and different masks can be used for different proofs. Thus, in short, the attending signal emanating from the attention units is multiplexed with the regular signal and used to focus the attention of the hidden layer on the hidden unit encoded by the bits appearing in the attention units. The output of the hidden layer is equal to the mask except for the attended position, where it is equal to the corresponding function $f_i(x)$.

This form of activation attention is the key tool for proving capacity lower bounds. To see this, consider for instance the case where an OR operator is applied to the outputs of the hidden layer. With a mask consisting of 0s, when the attention bits are set to $p(i)$, the output of the OR applied to the hidden units is equal to $f_i(x)$. Thus the truth table of the overall input-output function of the original inputs plus the attention bits is uniquely equal to $f_i(x)$ when the attention bits are set to $p(i)$. Thus the capacity of the network with the expanded input of size $n + \lceil logm \rceil$ is lower bounded by the sum of the capacities associated with the functions f_i over the original input of size n.

6. Capacity of Single Unit Attention

We can now begin to estimate the capacity of various attention circuits. As in physics, where from quarks one proceeds to protons, neutrons, the hydrogen atom, and so forth, we begin with the simplest case, where the attention signal originates in a single gating unit, as shown in Figure 13. Attention blocks of increasing complexity are considered in the following sections.

6.1. Capacity of Single Attention Units: Output Gating.

We want to compute the capacity of the class of all functions that can be computed by one neuron gated by another neuron, corresponding to the left hand side of Figure 13. In the purely linear case,we have seen that this is the set of all quadratic functions of the form $O = (\sum_i w_i x_i)(\sum_j v_j x_l)$. To partially address this question in the non-linear case, we can consider first the case of a linear threshold gate gated by another linear threshold gate, and then similarly for polynomial threshold gates of degree d. Using $-/+$ linear threshold gates for the gated and the gating units, this is the class of Boolean functions of the form:

$$fg(x) = f(x)g(x) = \text{sign}(\sum_i w_i x_i)\,\text{sign}(\sum_i v_i x_i) = \text{sign}\left((\sum_i w_i x_i)(\sum_j v_j x_j)\right) \tag{6.1}$$

This class contains the identity and all the linear threshold gates. Thus, by Zuev's result (Equation 5.1) its capacity is at least $n^2(1 + o(1))$. However, intuitively, it must contain

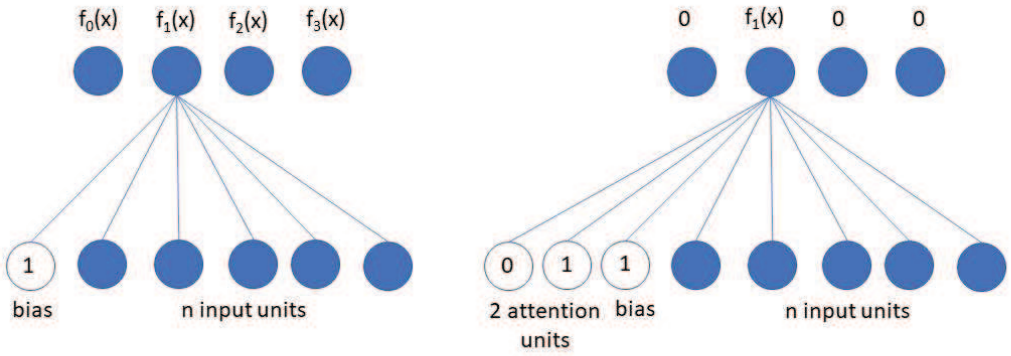

Figure 12. Left: A fully connected feedforward neural network with n inputs and $m = 4$ threshold gates computing the functions $f_0(x), f_1(x), f_2(x)$ and $f_3(x)$. The bias unit is constantly set to 1. Right: The same network with $2 = \lceil \log_2 m \rceil$ additional attention units in the input payer. The weights from the input units to the threshold gates are the same as in the left image. The attention units can be in 4 different states (00), (01), (10), and (11); these states can be associated in 1:1 fashion with the $m = 4$ threshold units. Assume for instance that the state $(1, 0)$ is associated with the hidden unit computing $f_1(x)$ (hidden unit 1). Then by Lemma 5.1 applied to the hypercube of dimension 2, it is possible to choose a set of weights from the attention units and the bias to hidden unit 1 providing: (1) an attention activation of 0 when the attention units are in the (0,1) state; and (2) an arbitrarily large negative (or an arbitrarily large positive) attention activation for all the other 3 states. As a result, when the attention units are in the (0,1) state the output of the attended hidden unit 1 is equal to $f_1(x)$. When the attention units are in any of the other three states, unit 1 is not attended and its output is constant and equal to 0 (or constant and equal to 1). And similarly, mutatis mutandis, for the other three hidden units. In other words, we can first choose a fixed pattern of 0s and 1s in the hidden layer, called a mask, and then connect the attention units to the hidden layer with such weights that the output of the hidden layer is equal to the mask, except for one position associated with the attended unit. If the attended unit is unit i in the hidden layer, the corresponding output is equal to $f_i(x)$ (and the attention units must be set to the corresponding values).

many other functions as shown in Figure 14 suggesting that in general the product of two linearly separable functions is not linearly separable. On the other hand, the upperbound on the capacity is at most $2n^2(1 + o(1))$, because the capacity is always bounded by the sum of the capacity of each individual component. Similarly considerations can be made for the $0/1$ which leads to the more general problem of estimating the capacity of the class of functions of the from $B(f, g)$ where B is any Boolean operator, and f and g are linear or polynomial threshold gates. And even more generality can be obtained by considering classes of Boolean

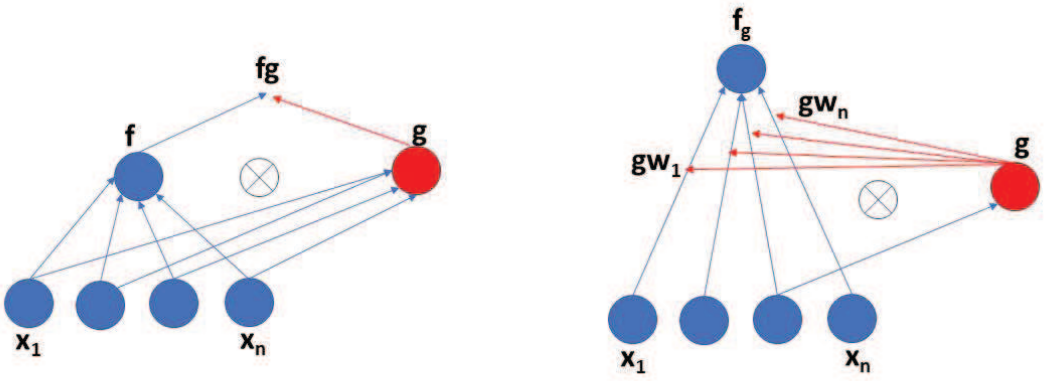

Figure 13. Left: Output gating with a single attention unit. Both the gated function f and the gating function g are linear (or polynomial) threshold gates of the same input vector $x = (x_1, \ldots, x_n)$. The overall network computes the function $fg(x) = f(x)g(x)$. Right: Synaptic gating with a single attention unit $g(x)$ gating all the incoming weights of the gated function $f(x)$. So the overall network computes the function $f_g(x)$. For instance, if f is a linear threshold gate $f(x) = \text{sign}(\sum_i w_i x_i)$, then $f_g(x) = \text{sign}(\sum_i g(x) w_i x_i)$.

TABLE 2. All possible Boolean combinations. There are 16 possible Boolean functions $B(p, q)$ of two variables p and q. Each row corresponds to a function $B(p, q)$ and its negation $\neg B(p, q)$. Ten Boolean functions are irreducible i.e., they cannot be expressed as a function of a smaller number of variables. Eight Boolean functions are symmetric ($B(p, q) = B(q, p)$). Fourteen Boolean functions can be implemented by a linear threshold gate (LTG). The functions are organized into four groups separated by horizontal lines. Within a group, all the functions in the same column are equivalent when their arguments are implemented by linear threshold gates. The last column correspond to the cardinal capacity $C(B(f, g))$ when f and g vary among all possible linear threshold functions of the same n variables.

$B(p,q)$	$\neg B(p,q)$	Irred. $(k=2)$	Sym	LTG	C{B(f,g)})
T	F	no	yes	yes	1
p	$\neg p$	no	no	yes	$n^2(1+o(1))$
q	$\neg q$	no	no	yes	
p OR q	$\neg p$ AND $\neg q$	yes	yes	yes	$2n^2(1+o(1))$
p OR $\neg q$	$\neg p$ AND q	yes	no	yes	
$\neg p$ OR q	p AND $\neg q$	yes	no	yes	
$\neg p$ OR $\neg q$	p AND q	yes	yes	yes	
p XOR q	$\neg(pXORq)$	yes	yes	no	$2n^2(1+o(1))$

functions of the form $B(f_1, \ldots, f_k)$ where B is a k-ary Boolean operator and $f_1, \ldots, f_k$ are polynomial threshold gates of respective degrees $d_1, \ldots, d_k$. We first address the case of $k = 2$ and then the general case.

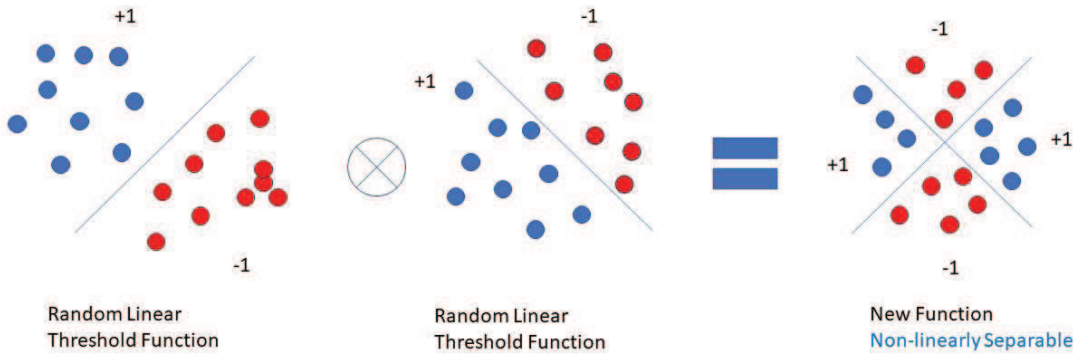

Figure 14. Two randomly selected -/+ linear threshold functions and their product. We can randomly pick such functions by randomly picking normal vectors of weights $w = (w_i)$ on the unit sphere S^{n-1} (or using i.i.d coordinates that are Normal or Uniform). When n is large, the normal vectors w and v are approximately orthogonal and the corresponding hyperplanes partition the space into four regions, each one containing approximately 2^{n-2} points of the hypercube. In general, the resulting function is not linearly separable.

6.1.1. *Pairwise Composition ($k = 2$).* For completeness, consider Table 2 summarizing all 16 Boolean function $B(p, q)$ of two variables. We can substitute p and q with arbitrary linear (or polynomial) threshold functions f and g and compute the corresponding cardinal capacity. The first group in the table correspond to always true (T) and always false (F) functions, thus to a negligible total capacity of 1. The second group corresponds to a single linear threshold function, and thus its capacity is equal to: $n^2(1 + o(1))$ All the elements in the second group in the table are also found in the third group corresponding to the AND and OR operators, because $f \,\text{AND}\, f = f \,\text{OR}\, f = f$. Within the third group, all the OR expression are equivalent to each other, and all the AND expressions are equivalent to each other, when p and q are substituted with linear (or polynomial) threshold gates. This is because whenever f is a polynomial threshold gate of degree d, then $\neg f$ is also a polynomial threshold gate of degree d. The first three groups cover 14 Boolean functions $B(p, q)$ in total. These 14 Boolean functions can be implemented by a single linear threshold gate of p and q, and no other linear threshold gate of p and q exist. Thus the total aggregated capacity corresponding to all these cases, is given by the cardinal capacity $C(n, 2, 1)$ of a network of linear threshold gates with n inputs, 2 hidden units, and 1 output unit. This capacity is given by [7, 8]:

$$C(n, 2, 1) = 2n^2(1 + o(1)) \tag{6.2}$$

There is a one-to-one correspondence between the set $\{f \, AND \, g\}$ and the set $\{f \, OR \, g\}$ through the negation operator. Therefore:

$$C(\{f \,\text{AND}\, g\}) = C(\{f \,\text{OR}\, g\}) \tag{6.3}$$

[Note that any Boolean function that isolates once corner of the hypercube is irreducible. For such a function, knowing the values of the sequence $(B(f,g), B(f,\neg g), B(\neg f,g), B(\neg f,\neg g))$ uniquely determines the values of f and g]. The relevant result in [7, 8] is obtained using the attention multiplexing technique, applied in fact with the OR Boolean function and a mask of 0s, as described in Section 5.5. Thus, in short:

$$C(\{f \text{ AND } g\}) = C(\{f \text{ OR } g\}) = 2n^2(1+o(1)) \tag{6.4}$$

For the last row of the table, the output gating (multiplication) of two $-/+$ linear threshold functions correspond to applying the negation of the XOR Boolean operator. Note that: $f\,XOR\,\neg g \equiv \neg f\,XOR\,g \equiv \neg(f\,XOR\,g)$ and $f\,XOR\,g \equiv \neg f\,XOR\,\neg g$. As a result we have:

$$|\{f\,XOR\,g\}| = |\{\neg(f\,XOR\,g)\}| \tag{6.5}$$

when f and g vary over all possible linear threshold gates. Even more strongly, the corresponding sets of Boolean functions are identical:

$$\{f\,XOR\,g\} = \{\neg(f\,XOR\,g)\} \tag{6.6}$$

Now it is easy to see that

$$n^2(1+o(1)) \leq C(\{f\,XOR\,g\}) = C(\{\neg(f\,XOR\,g)\}) \leq 2n^2(1+o(1)) \tag{6.7}$$

The lower bound is obtained by noticing that for any Boolean function f, $f\,XOR\,F = f$. The upperbound is obtained by noticing that $f\,XOR\,g$ can be implemented by a network $A(n,2,2,1)$ of linear threshold gates (using the disjunctive normal form), and the capacity of such a network is always at most equal to the sum of the capacities of its individual gates. Finally, the attention multiplexing technique described in Section 5.5 applied with a mask of 1s (since $f\,NXORT = f$) shows that:

$$C(\{fXORg\}) = C(\{\neg(fXORg)\}) = 2n^2(1+o(1)) \tag{6.8}$$

Thus the product of $0/1$ or $-/+$ linear threshold gates have the same capacity, and a similar argument holds for polynomial threshold gates. These results can be summarized in the following theorem, which is true for both $0/1$ and $-/+$ threshold gates:

Theorem 6.1. *The capacity of a linear threshold gate output-gated by another linear threshold gate is given by:*

$$2n^2\left(1+o(1)\right) \tag{6.9}$$

Likewise, the capacity of a polynomial threshold gate of degree d output-gated by another polynomial threshold gate of the same degree is given by:

$$2\frac{n^{d+1}}{d!}\left(1+o(1)\right) \tag{6.10}$$

Remark 6.2. Furthermore, we have seen that every Boolean function can be written as a product of linear threshold gates with an exponential number of terms (Proposition 4.1). Theorem 6.1 shows that it is not possible to do so using only a polynomial number of terms, since this would result in an overall capacity that is only polynomial, whereas the capacity of B_n is 2^n.

Remark 6.3. The estimate in Equation 6.9 can be slightly refined using Equation 5.2 instead of 5.1.

Remark 6.4. These results can be extended to other interesting cases. For instance, if we assume that the weights of the gated and gating linear threshold neurons are binary with $-/+$ values, then the output gating capacity is equal to $2n\,(1+o(1))$. See Figure 15 for a partial depiction of the underlying hierarchy of Boolean functions.

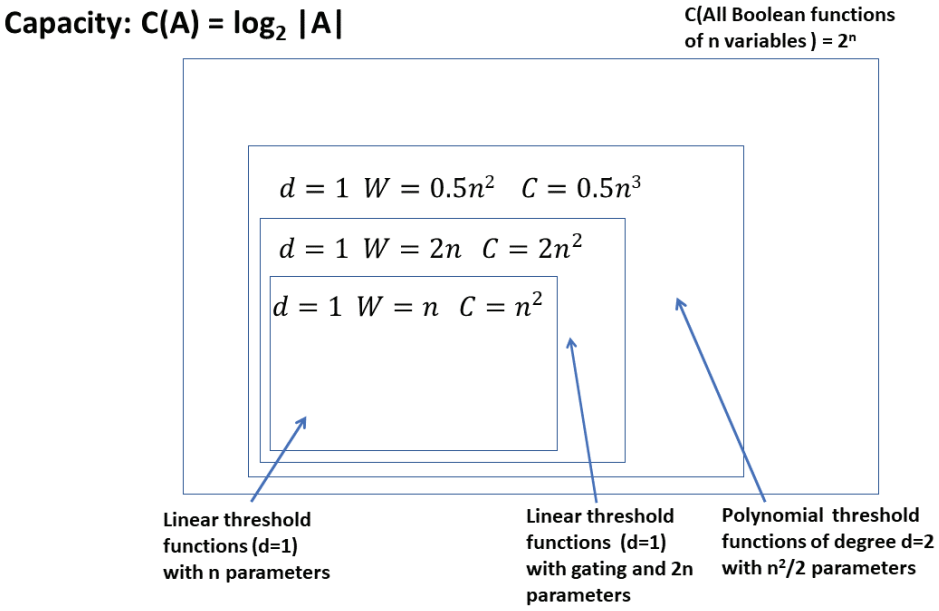

Figure 15. Hierarchy of classes of Boolean functions of n variables according to their asymptotic capacity C and number of parameters W. Linear threshold functions require n parameters and achieve capacity n^2. Linear threshold functions gated by linear threshold functions require $2n$ parameters and achieve larger capacity (e.g., they contain XOR) equal to $2n^2$. Quadratic threshold functions require $n^2/2$ parameters and achieve capacity $n^3/2$ [9]. The set of all Boolean functions correspond to an exponential capacity exactly equal to 2^n. Note that in all these cases $C = nW$.

6.1.2. *General Composition ($k \geq 2$).* The results in the previous section can be generalized to the class of functions of the form $B(f_1, \ldots, f_k)$, where B is a Boolean function of k variables and, for each j, $f_j \in \mathcal{T}(n; d_j)$. We denote this class by: $\mathcal{T}_B(n; d_1, \ldots, d_k)$.

Theorem 6.5 (Composition)**.** *Let B be an irreducible Boolean operator in k variables.*[2] *Then:*

$$\prod_{j=1}^{k} \left|\mathcal{T}(n-k+1; d_j)\right| \leq \left|\mathcal{T}_B(n; d_1, \ldots, d_k)\right| \leq \prod_{j=1}^{k} \left|\mathcal{T}(n; d_j)\right| \tag{6.11}$$

Furthermore, if B is the set of all irreducible Boolean functions of two variables (there are 10 of them), we have:

$$\left|\bigcap_{B} \mathcal{T}_B(n; d_0, d_1)\right| \geq \left|\mathcal{T}(n-1; d_0)\right| \left|\mathcal{T}(n-1; d_1)\right| \tag{6.12}$$

where the intersection is over the ten irreducible binary Boolean operators.

[2]Irreducibility means that B can not be expressed as a Boolean operator in fewer than k variables.

The complete proof of this theorem is given in the Appendix. The upper bound is easy and the lower bound relies on the attention multiplexing approach (Section 5.5). To check the special case, when $k = 2$, for two polynomial threshold gates of degree d_1 and d_2, Theorem 6.5 yields:

$$\left|\mathcal{T}(n-1;d_1)\mathcal{T}(n-1;d_2)\right| \leq \left|\mathcal{T}_B(n;d_1,d_2)\right| \leq \left|\mathcal{T}(n;d_1)\mathcal{T}(n;d_2)\right| \tag{6.13}$$

and when $d_1 = d_2 = d$:

$$\left|\mathcal{T}(n-1;d)\right|^2 \leq \left|\mathcal{T}_B(n;d,d)\right| \leq \left|\mathcal{T}(n;d)\right|^2 \tag{6.14}$$

Thus, in the case of output gating of two linear threshold gates, we have:

$$\left|\mathcal{T}(n-1;1)\right|^2 \leq \left|\mathcal{T}_B(n;1,1)\right| \leq \left|\mathcal{T}(n;1)\right|^2 \tag{6.15}$$

Substituting the estimates in Equations 5.1–5.5 in these inequalities gives immediately Theorem 6.1. Note that the intersection across all 10 irreducible Boolean functions is large.

6.2. **Capacity of Single Attention Units: Synaptic Gating.** We are now ready to compute the capacity for the case corresponding to the right hand side of Figure 13, where one threhsold unit synaptically gates the weights of another threshold unit. To begin with, we look at the case where all the weights of the gated unit are gated simultaneously. The main result is as follows:

Theorem 6.6. *Let $f(x)$ and $g(x)$ be two linear or polynomial threshold gates (not necessarily of the same degree), both with the same $-/+$ or $0/1$ output encoding and n binary input variables. Then full synaptic gating of f by g, where all the coefficients of f are multiplied by g, is equivalent to output gating of f by g. In particular, if both gates are linear threshold gates, then the corresponding capacity is given by:*

$$2n^2\left(1+o(1)\right) \tag{6.16}$$

and if both gates are polynomial threshold gates of degree d, then the corresponding capacity is given by:

$$2\frac{n^{d+1}}{d!}\left(1+o(1)\right) \tag{6.17}$$

Proof. We sketch the proof when f and g are linear threshold gates, but the argument extends immediately to polynomial threshold gates. Let us assume that $f(x) = \text{sign}(\sum_i w_i x_i)$ and $g(x) = \text{sign}(\sum v_i x_i)$. Then, with full synaptic gating, the gated function satisfies: $f_g(x) = \text{sign}(\sum_i g(x) w_i x_i) = \text{sign}(g(x)\sum_i (w_i x_i) = \text{sign}\, g(x)\, \text{sign}(\sum_i w_i x_i) = g(x)f(x)$. In the case of $0/1$ units, if H is the Heaviside function, then: $f_g(x) = H(\sum_i g(x) w_i x_i) = H(g(x)\sum_i w_i x_i)$. If $g(x) = 1$, this is the same as $f(x)g(x)$. Likewise if $g(x) = 0$, as long as we define $H(0) = 0$, then $f_g(x)$ is also equal to: $f(x)g(x)$. [Note that the gating is applied to the bias too]. □

Remark 6.7. In this particular case, to some extent, we can also consider the mixed case. If $f(x)$ is a $-/+$ gate and $g(x)$ is a $0/1$ gate, if we define $\text{sign}\, 0 = 0$ then we also have $f_g(x) = f(x)g(x)$ everywhere. If $f(x)$ is a $0/1$ gate and $g(x)$ is a $-/+$ gate, then when $g(x) = 1$ we also have $f_g = fg$. However, when $g(x) = -1$, then $f_g(x) = \neg f(x) = 1 - f(x)$.

Remark 6.8. In both Theorems 6.1 and 6.6 there is approximately a doubling of the capacity at the cost of doubling the number of parameters.

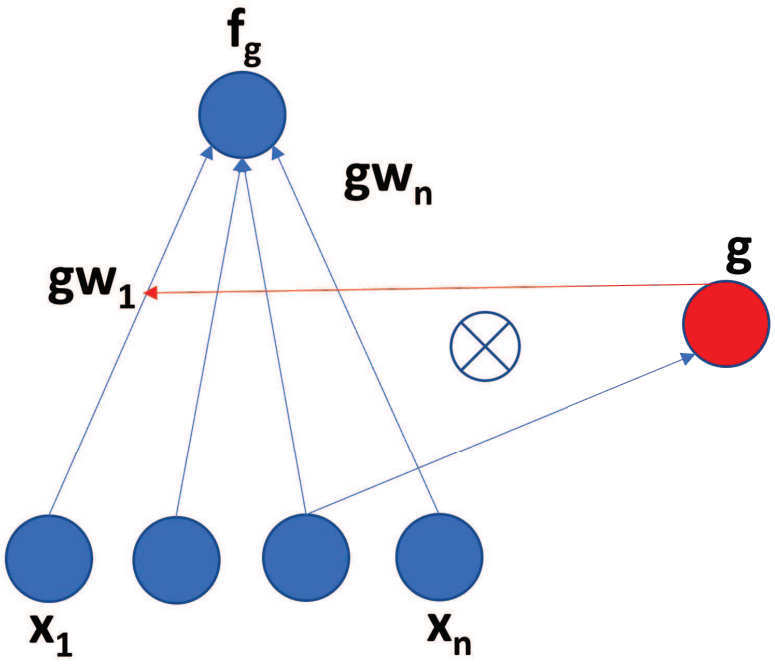

Figure 16. Synaptic gating with a single attention unit. Both the gated function f and the gating function g are linear (or polynomial) threshold gates of the same input vector $x = (x_1, \ldots, x_n)$. Through the synaptic gating operation, a single synaptic weight of the function f (w_1 in the figure) is multiplied by g.

Finally, we consider the synaptic gating case where the gating unit gates only *one* of the weights of the gated unit (Figure 16). The following Proposition provides bounds on the corresponding capacity.

Proposition 6.9. *Consider the case of a linear threshold gate f with n binary inputs, where one of the weights is synaptically gated by the output of a second linear threshold gate g of the same n inputs. Then the capacity C satisfies:*

$$n^2\left(1+o(1)\right) \leq C \leq 2n^2\left(1+o(1)\right) \tag{6.18}$$

If the linear threshold gates are replaced by polynomial threshold gates of degree d, the capacity C satisfies:

$$\frac{n^{d+1}}{d!}\left(1+o(1)\right) \leq C \leq 2\frac{n^{d+1}}{d!}\left(1+o(1)\right) \tag{6.19}$$

The same bounds hold for the case of additive activation attention between two linear polynomial threshold gates, or two linear polynomial threshold gates, of the same n inputs.

Proof. The result is true for both $0/1$ and $-/+$ encodings of the outputs. We provide the proof in the linear case but the technique is the same for polynomial threshold gates of degree $d > 1$. The lower bound results immediately from the fact that the gating unit could have an output constant and equal to 1 $g(x) = 1$). In this case the gated function is equal to $f(x)$ and the lower bound is the corresponding capacity estimate. The upperbound is simply the sum of the capacities. A similar argument applies for the case of additive activation attention. □

7. Capacity of Attention Layers

The previous attention results are obtained using only two neurons, a gating neuron and a gated neuron, with either output gating or synaptic gating. We now extend the capacity analysis to cases where there is a layer of gating neurons, as shown in Figure 17 for both output and synaptic gating.

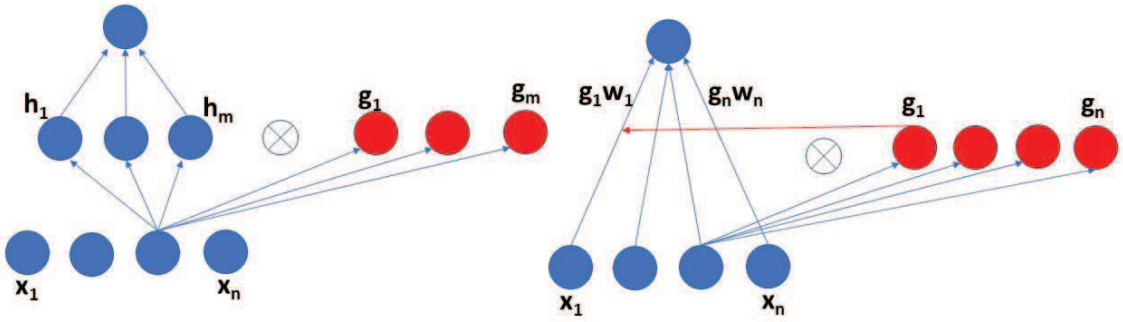

Figure 17. Left: output gating by a gating layer. For the same n dimensional input vector x, there are m hidden units computing functions $h_1(x), \ldots, h_m(x)$, and m corresponding gating units computing functions $g_1(x), \ldots, g_m(x)$. With the gating, the effective output of the hidden units is given by $h_1(x)g_1(x), \ldots, h_m(x)g_m(x)$. The final output unit produces an output of the form $f(h_1(x)g_1(x), \ldots, h_m(x)g_m(x))$. In the capacity analysis, we assume that the functions h, g, and f are linear threshold gates. Right: synaptic gating by a gating layer. In this case, there is a unit computing a function $f(x)$ with n weights $w_i, \ldots, w_n$. There are n gating functions $g_1(x), \ldots, g_n(x)$, each one multiplicatively gating one of the weights w. If $f = \text{sign}(\sum_i w_i x_i)$ then $f_g(x) = \text{sign}(\sum_i g_i(x) w_i x_i)$.

7.1. **Capacity of Attention Layers: Output Gating.** We now examine the capacity of a network with one attention layer with output gating, as depicted on the left hand side of Figure 17. Thus we consider an architecture with n inputs, m hidden linear threshold units gated by m corresponding linear threshold units, and one final linear threshold output gate. All the linear threshold gates have $-/+$ outputs, although the following theorem is unchanged, and the method of proof is similar, if the gates have $0/1$ outputs. We denote by $\mathcal{T}(n, m, 1; \times)$ the corresponding set of Boolean functions. Note that this is the same architecture for computing the dot product of the gated and the gating hidden layer outputs, except that the final unit is non-linear with variable weights, instead of being linear with fixed weights equal to one. We will also let $\mathcal{T}(n, 1; \times)$ denote the set of Boolean functions corresponding to one linear threshold gate of n variables output-gated by another linear threshold gate of the same variables.

Theorem 7.1. *The capacity $C(\mathcal{T}(n,m,1;\times))$ of the set of Boolean functions corresponding to n inputs, m hidden linear threshold gates output-gated by m hidden linear threshold gates of the same inputs, followed by one linear threshold gate output satisfies:*

$$mn^2 \leq C(\mathcal{T}(n,m,1;\times)) \leq 2mn^2\left(1+o(1)\right) \tag{7.1}$$

for $n \to \infty$, and for any choice of $m \in [1, 2^{o(n)}]$. Furthermore:

$$C(\mathcal{T}(n,m,1;\times)) = mC(T(n,1;\times))\left(1+o(1)\right) \tag{7.2}$$

Thus:

$$C(\mathcal{T}(n,m,1;\times)) = 2mn^2\left(1+o(1)\right) \tag{7.3}$$

Proof. Let us denote by f the map between the input layer and the hidden layer with gating, and by ϕ the map from the hidden layer to the output layer. For the upper bound, we first note that the total number of possible maps f is bounded by $2^{mC(\mathcal{T}(n,1;\times))} \leq 2^{2mn^2(1+o(1))}$, since f consists of m threshold gates gated by m threshold gates, and thus each gated unit corresponds to at most $2^{C(\mathcal{T}(n,1;\times))} \leq 2^{2n^2(1+o(1))}$ possibilities by the Theorems in Section 6. Any fixed map f, produces at most 2^n distinct vectors in the hidden layer. It is known [1] that the number of threshold functions ϕ of m variables defined on at most 2^n points is bounded by:

$$2\binom{2^n-1}{\leq m} = 2^{nm(1+o(1))} \tag{7.4}$$

using the assumption $m \leq 2^{o(n)}$. Thus, under our assumptions, the total number of functions of the form $\phi \circ f$ is bounded by the product of the bounds above which yields immediately:

$$C(\mathcal{T}(n,m,1;\times)) \leq mC(\mathcal{T}(n,1;\times))\left(1+o(1)\right) \leq 2mn^2\left(1+o(1)\right) \tag{7.5}$$

For the lower bound, we can force the gating units to be the identity (i.e., with a constant output equal to 1). In this particular case, the gating units can be ignored and we need to count the number of Boolean functions that can be implemented in the remaining architecture. A theorem in [8] shows that this number is equal to $mn^2(1+o(1))$.

To prove the rest of the theorem, we use attention multiplexing. As a reminder, the basic idea is to have a small set of the input units act as attention units that can be used to select a particular function in the hidden layer. The same setting of the attention units will be used to select the corresponding functions in both the gating and gated layers. More formally, we decompose n as: $n = n^- + n^+$ where $n^- = \lceil \log_2 m \rceil$ corresponds to the attention units. Likewise, we decompose each input vector $x = (x_1, \ldots, x_n) \in \{-1,+1\}^n$ as: $x = (x^-, x^+)$, where:

$$x^- = (x_1, \ldots, x_{n^-}) \in \{-1,+1\}^{n^-} \quad \text{and} \quad x^+ = (x_{n^-+1}1, \ldots, x_n) \in \{-1,+1\}^{n^+} \tag{7.6}$$

For any gated Boolean linear threshold map f^+ from $\{-1,+1\}^{n^+}$ to $\{-1,+1\}^m$, we can uniquely derive a map $f = (f_1, \ldots, f_m)$ from $\{-1,+1\}^n$ to $\{-1,+1\}^m$ defined by:

$$f_i(x^-, x^+) = [x^- = i] \;\; AND \;\; [f_i^+(x^+)] \tag{7.7}$$

Here $x^- = i$ signifies that the binary vector x^- represents the digit i. In other words $x^- = i$ is used to select the i-th unit in the gated layer as well as in the gating layer, and filter f^+ by retaining only the value of f_i^+. By Lemma 5.1), this selection procedure can be expressed using a single linear threshold function of the input x^- for the gated layer, and similarly for the gating layer. We say that f is obtained from f^+ by multiplexing and f is a gated threshold map. It is easy to see that the filtering of two distinct maps f^+ and g^+ results

into two distinct maps f and g. Now let us use $\phi = OR$ in the top layer–note that OR can be expressed as a linear threshold function. Then it is also easy to see that $\phi \circ f \neq \phi \circ g$. Thus the total number of Boolean functions that can be implemented in this architecture is lower-bounded by the number of all gated Boolean maps f^+. This yields:

$$C(\mathcal{T}(n, m, 1; \times)) \geq mC(\mathcal{T}(n^+, 1; \times)) \left(1 + o(1)\right) = 2mn^2 \left(1 + o(1)\right) \tag{7.8}$$

using the fact that $n^+ = n - \lceil \log_2 m \rceil$, and $\lceil \log_2 m \rceil = o(n)$ by assumption. Thus: $C(\mathcal{T}(n, m, 1; \times)) = mC(\mathcal{T}(n, 1; \times)) \left(1 + o(1)\right) = 2mn^2 \left(1 + o(1)\right)$. □

Remark 7.2. In Theorem 7.1, we see again that both the capacity and the number of parameters approximately double at the same time.

7.2. **Capacity of Attention Layers: Synaptic Gating.** We now examine the capacity of a network with one attention layer with synaptic gating, as depicted on the right hand side of Figure 17, with each gating neuron gating a different weight of a gated neuron.

Proposition 7.3. *Consider the case of a linear threshold gate with n inputs and n weights, where each weight is synaptically gated by an independent linear threshold gate of the same n inputs. Then the capacity C satisfies:*

$$n^2 \left(1 + o(1)\right) \leq C \leq n^3 \left(1 + o(1)\right) \tag{7.9}$$

If the linear threshold gates are replaced by polynomial threshold gates of degree d, the capacity C satisfies:

$$\frac{n^{d+1}}{d!} \left(1 + o(1)\right) \leq C \leq \frac{n^{d+2}}{d!} \left(1 + o(1)\right) \tag{7.10}$$

Proof. The proof is similar to the proof of Proposition 6.9. The lower bound is obtained by constraining all the gating units to have a constant output equal to 1. The upperbound is simply the sum of all the capacities. □

Likewise, we can consider an architecture with n inputs, one layer of m gating units, and one parallel layer of m gated units. Each gating unit is uniquely paired with one gated unit (one to one) and synaptically gates one of the weights of the gated unit.

Proposition 7.4. *Consider the case of an architecture with n inputs, one layer of m gating units, and one parallel layer of m gated units. Each gating unit is uniquely paired with one gated unit (one to one) and synaptically gates one of the weights of the gated unit. Then the capacity C satisfies:*

$$mn^2 \left(1 + o(1)\right) \leq C \leq 2mn^2 \left(1 + o(1)\right) \tag{7.11}$$

If the linear threshold gates are replaced by polynomial threshold gates of degree d, the capacity C satisfies:

$$m\frac{n^{d+1}}{d!} \left(1 + o(1)\right) \leq C \leq 2m\frac{n^{d+1}}{d!} \left(1 + o(1)\right) \tag{7.12}$$

Proof. The proof is similar to the proof of Proposition 6.9. The lower bound is obtained by constraining all the gating units to have a constant output equal to 1. The upperbound is simply the sum of all the capacities. □

Remark 7.5. **Attention Capacity of Transformer Blocks:** Consider the problem of estimating the capacity of an encoder block (Figure 5) in a transformer architecture. The only adjustable parameters are in the shared maps used to produce the K, Q,and V vectors at each one of the n input positions. Most often, these maps are linear and thus to quantify

the capacity one can, for instance, quantify the corresponding matrix entries. This can be done by limiting the precision of the matrix entries. While this approach is reasonable, to focus the analysis on the attention mechanism itself, we can assume that the value vectors are fixed and count how many different outputs can be produced by the attention mechanism, as we vary the key and query vectors. For simplicity, we can assume that the query and key vectors are binary vectors. Assuming these vectors have length m, then there are 2^{mn} possible choices in total for the n key vectors (with repetition allowed), and similarly for the query vectors. Thus the Q and K vectors of the transformer encoder block can produce at most 2^{2mn} states for the overall attentional mechanism. Thus, in this sense, the attentional mechanism itself has a capacity of at most $2mn$.

8. Conclusion

In addition to the fundamental role attention plays in brain function, attention mechanisms have also become important for artificial neural networks and deep learning. Here we have taken the first steps towards building a theory of attention mechanisms by first identifying the quarks of attention, i.e., its smallest building blocks. Using the three variable types of the SM allows for the systematic identification and organization of possible attention building blocks based on their origin type, target type, and whether the mechanism of action is additive or multiplicative. Assuming that the attention signal originates from the output of some neurons, this yields six possibilities, which can then be reduced to three main cases: activation attention, output gating, and synaptic gating. Activation attention falls within the SM, whereas output gating and synaptic gating correspond to multiplicative extensions of the SM. Current attention-based architectures in deep learning, including transformers, are built out of attention modules which are themselves built out of output gating and synaptic gating operations. These operations and modules can be viewed as new primitives in the language of neural architectures in digital simulations and, because they are differentiable, the usual backpropagation learning framework can easily be extended to them. However, in a physical neural machine, these operations require additional connections (wires) and physical mechanisms for implementing multiplicative interactions.

Output gating can be used dynamically to directly silence unattended neurons, and to magnify the output of attended neurons. It can also be used as the main building block of a shallow module that can compute the dot product of two vectors of neuronal activities. The latter is a key, massively used, component of transformer architectures.

Synaptic gating is a fast synaptic mechanism that can be used dynamically to silence or weigh the attended synapses. It is often used in combination with a softmax operator to enable dynamic convex combinations of vectors, as in the transformer architectures. The concept of fast synapses that can vary their strengths on fast time scales is not new and has been associated with different roles, in different contexts. For instance, one potential role is the storage of transient information, such as intermediary results during mental reasoning, or simply the memorization of the beginning of a paragraph as the reading of the paragraph proceeds. A second potential role stems from viewing synaptic weights as computer programs, and thus fast synapses as enabling dynamic changes in the programs that are being executed and the implementation of parameterized functions. And a third role studied here is the enabling of attention. These three roles are not independent and raise interesting architectural questions for deep learning and neuroscience and the possible need for multiple synaptic time scales interacting in hierarchical ways.

To see this, as an example, consider the reading paradigm where information about the first sentence of a long paragraph is stored using a set of fast weights. If, as the reading proceeds, one must suddenly access a specific subset of this transiently stored information, attention must be directed towards certain particular words contained in the first sentence. In a deep learning architectures, this can be thought of in terms of a softmax synaptic gating, as is done in transformer and other NLP architectures. Thus somehow this fast weight attention mechanism must operate upon, and be faster than, the fast weight synaptic mechanism used to store information about the first sentence.

Another related question that is important for both deep learning and neuroscience is the issue of positional information and how it is encoded. The existence of topographic maps in the visual cortex and other areas suggests that at least in the case of 2D data like images, position is encoded in the brain topographically through the connectivity of the underlying architectures. Transformers can be, and have been, applied to image data, but in contrast they need positional information to be added to the value of individual pixels, or groups of pixels. This alone suggests to us that there may exist deep architectures for NLP problems, where position is encoded in the connectivity rather than the inputs, which perform as well if not better than transformers.

Attention mechanisms allow the attending network to modulate the function computed by the attended network, thereby expanding the scope of useful functions that can be efficiently implemented and trained in deep learning. Because the SM already has universal approximation properties, its extensions should not be evaluated in terms of which functions can be approximated, but rather in terms of other efficiencies. While attention blocks act as new primitives in standard deep learning software libraries, having access to output gating and synaptic gating mechanisms in a physical neural network can reduce its depth. Using the notion of cardinal capacity, and working with the approximation provided by Boolean neurons (linear or polynomial threshold gates), enables systematic investigations of the capacity of attentional circuits that were previously not possible. In particular, we have been able to estimate the capacity of basic attentional circuits involving linear, or polynomial, threshold gates. In many cases of interest, we found essentially a doubling of the capacity with a doubling of the number of parameters, which is a sign of efficiency.

Perhaps surprisingly, a key ingredient in the capacity proofs is the third form of attention, activation attention. Activation attention is used to prove capacity lower bounds by the multiplexing approach which selects a unit in a layer, as a function of the attending units, while driving the remaining units in the layer to low or high saturation. There is work left for tightening some of the capacity estimates and for extending them to other activation functions and other architectures. Although the capacity results stand on their own mathematical proofs, further corroborating them through computer simulations poses also unique challenges. This is already obvious for the simpler problem of using computer simulations to estimate the capacity of a single neuron, i.e., the number c of linear threshold functions of n variables. This is also equivalent to estimate the probability p that a Boolean function drawn randomly and uniformly from the set of all Boolean functions of n variables is a linear threshold function, or that a uniformly random coloring of the hypercube is linearly separable. Zuev's theorem states that $c \approx 2^{n^2}$. Thus a direct enumeration is faced with the fact that the number of linear threshold function grows exponentially fast and is prohibitively large even for relatively small values of n (e.g., $n = 1000$), whereas a simple Monte Carlo sampling approach is equally bound to fail since the probability $p \approx 2^{n^2}/2^{2^n}$ goes to zero

exponentially fast. The results in Section 6 and 7 build on Zuev's theorem and raise even harder challenges for computer simulations.

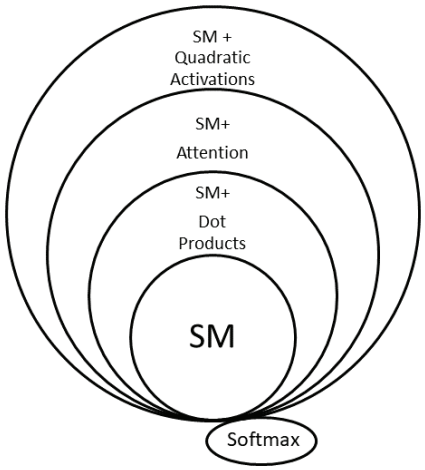

Figure 18. Standard model and some of its extensions.

Overall, both output and synaptic gating are extensions of the SM which introduce quadratic terms in the SM (Figure 18). Quadratic terms are powerful but expensive: a neuron with full quadratic activation over its n inputs requires on the order of n^2 synaptic parameters. Using quadratic activations everywhere in a large deep architecture leads to implementations that may not be efficient in terms of parameters and learning. Attention mechanisms are a way of introducing quadratic terms in a sparse way, in order to gain some of the benefits of quadratic activations, without paying the full price.

Finally, we can return to the quotes in the introduction linking attention to awareness and pointing to the inadequacy of having a single term. While subjectively we feel that we can control and direct our attention and be aware of its shifts, it should be obvious that attention mechanisms, such as output or synaptic gating, are computational mechanisms that do not require awareness. They can operate at all levels of a cognitive architecture, for instance to help implement dynamically whole-part hierarchies and ultimately awareness itself. Thus, in short, awareness is not necessary for attention, but attention may be necessary for awareness. Having a single term is indeed inadequate and, in time, it may have to be replaced with multiple terms to better capture the underlying complexities.

9. Appendix: Detailed Proof of Theorem 6.5

Here a polynomial threshold function is a function of the form $f = \text{sign}(p) : \{0,1\}^n \to \{-1,1\}$ where p is a polynomial in n real variables of degree at most d. The class of all such functions is denoted $\mathcal{T}(n;d)$.

Let $B(z_1, \ldots, z_k) : \{-1, 1\}^k \to \{-1, 1\}$ be a Boolean function in k variables. We are interested in the class of functions of the form $B(f_1, ..., f_k) : \{0, 1\}^n \to \{-1, 1\}$ where $f_j \in \mathcal{T}(n; d_j)$. Denote this class by $\mathcal{T}_B(n; d_1, \ldots, d_k)$. We want to prove the following theorem:

Theorem (Composition). *Let B be an irreducible Boolean operator in k variables. Then:*

$$\prod_{j=1}^{k} \left|\mathcal{T}(n-k+1; d_j)\right| \leq \left|\mathcal{T}_B(n; d_1, \ldots, d_k)\right| \leq \prod_{j=1}^{k} \left|\mathcal{T}(n; d_j)\right| \tag{9.1}$$

The upper bound is trivial from considering the total number of tuples $(f_1, ..., f_k)$ with $f_j \in \mathcal{T}(n; d_j)$. The lower bound is nontrivial except for $k = 1$ where both bounds become identical. The key to the proof is the multiplexing (activation attention) procedure, where k input components are viewed as attention units capable of producing a constant mask in the hidden layer, except for the attended function. Here for simplicity we use a sparse encoding in the k components, although dense encoding is also possible, as in the proof of Theorem 7.1. Dense encoding would lead to a reduction in the number of attending units from k to $\lceil \log_2 k \rceil$ as in Section 5.5. As a side note, using more attention units than the minimal number required, can be used to reduce the size of the attention weights, or to make the attention mechanism less sensitive to each individual attention bit.

To prove the lower bound in Composition Theorem 6.5, let us restate it equivalently as:

$$\prod_{j=0}^{k} \left|\mathcal{T}(n-k; d_j)\right| \leq \left|\mathcal{T}_B(n; d_0, \ldots, d_k)\right| \leq \prod_{j=0}^{k} \left|\mathcal{T}(n; d_j)\right|. \tag{9.2}$$

Irreducibility implies that if we select any input component i, the value of B cannot be determined entirely from the value of the remaining components alone. More formally:

Lemma 9.1. *Consider an irreducible Boolean operator $B = B(z_0, \ldots, z_k)$ and an index $i \in \{0, \ldots, k\}$. There exist signs $\theta \in \{-1, 1\}$ and $\theta_j \in \{-1, 1\}$, $j \in \{0, \ldots, k\} \setminus \{i\}$, such that:*

$$B(z_0, \ldots, z_k) = \theta z_i \quad \textit{whenever } z_j = \theta_j \textit{ for all } j \neq i. \tag{9.3}$$

Proof. Consider $B(z_0, \ldots, z_k)$ as a function of z_i. If this function is constant in the variable z_i no matter how we fix the other variables, then the value of $B(z_0, \ldots, z_k)$ is entirely determined by the values of these other variables, which contradicts irreducibility. Therefore, there exists some assignment $z_j = \theta_j$, $j \neq i$, so that the function $B(\theta_0, \theta_1, \ldots, z_i, \ldots \theta_k)$ is not constant in z_i. But there exists only two non-constant Boolean functions $f(x)$ in one variable: $f(x) = x$ or $f(x) = -x$, and this determines θ. □

The next lemma essentially states that we can fit an affine function of k variables to $k + 1$ points.

Lemma 9.2. *Let $e_0 = 0$ and $e_1, \ldots, e_k$ denote the canonical basis vectors in $\mathbb{R}^k$. Then, for any choice of index $j \in \{0, \ldots, k\}$ and signs $\theta_i \in \{-1, 1\}$, $i \in \{0, \ldots, k\} \setminus \{j\}$ there exists an affine function $q : \mathbb{R}^k \to \mathbb{R}$ such that:*

$$q(e_i) = \begin{cases} 0, & i = j \\ \theta_i, & i \neq j \end{cases} \tag{9.4}$$

for all $i \in \{0, \ldots, k\}$.

Proof. It is straightforward to check that the function:

$$q(z) = \theta_0 - \theta_0 z_j + \sum_{i \in \{0,\ldots,k\} \setminus \{j\}} (\theta_i - \theta_0) z_i \tag{9.5}$$

satisfies the required property. □

We can now use the previous lemma to derive a lemma for consistently extending a function of $n-k$ variables to a function of n variables. Here k components are used as selector of filter variables, as in the proof of Theorem 7.1.

Lemma 9.3. *Consider a function $f \in \mathcal{T}(n-k;d)$, an index $j \in \{0,\ldots,k\}$, and signs $\theta \in \{-1,1\}$ and $\theta_i \in \{-1,1\}$, $i \in \{0,\ldots,k\} \setminus \{j\}$. There exists a function $F \in \mathcal{T}(n;d)$ such that:*

$$F(e_i \oplus x) = \begin{cases} \theta f(x), & i = j \\ \theta_i, & i \neq j \end{cases} \tag{9.6}$$

for all $x \in \{0,1\}^{n-k}$. Here $\oplus$ denotes the concatenation operator.

Proof. Express the polynomial threshold function f as:

$$f(x) = \operatorname{sign}(p(x)) \quad \text{for } x \in \{0,1\}^{n-k} \tag{9.7}$$

where p is a polynomial in n variables and of degree at most d. Let q be a function that satisfies the conclusion of Lemma 9.2. Fix a number M large enough so that $M > \left|p(x)\right|$ for all $x \in \{0,1\}^{n-k}$, and define:

$$F(z \oplus x) = \operatorname{sign}\left(Mq(z) + \theta p(x)\right) \tag{9.8}$$

for all $z \in \mathbb{R}^k$ and $x \in \mathbb{R}^{n-k}$. By construction, F is a polynomial threshold function on $\{0,1\}^n$ of degree at most d as required.

Let us check that F satisfies the conclusion of the lemma. If $z = e_j$, we have $q(z) = 0$ due to our choice of q (per the conclusion of Lemma 9.2), and we get $F(z \oplus x) = \operatorname{sign}(\theta p(x)) = \theta f(x)$. If $z = e_i$ with $i \neq j$, then our choice of q implies $F(z \oplus x) = \operatorname{sign}(M\theta_i + \theta p(x))$. The choice of M guarantees that the term $M\theta_i$ dominates the term $\theta p(x)$ in magnitude, so we have $F(s \oplus x) = \operatorname{sign}(M\theta_i) = \theta_i$. □

We can now use Lemma 9.3 for the simultaneous extension and filtering of several functions of $n-k$ variables relative to an irreducible Boolean function B.

Lemma 9.4. *For any $(k+1)$-tuple of functions $(f_0,\ldots,f_k)$ where $f_j \in \mathcal{T}(n-k;d_j)$ there exists a $(k+1)$-tuple of functions $(F_0,\ldots,F_k)$ where $F_j \in \mathcal{T}(n;d_j)$ such that:*

$$B(F_0,\ldots,F_k)(e_i \oplus x) = f_i(x) \tag{9.9}$$

for all $i \in \{0,\ldots,k\}$ and $x \in \{0,1\}^{n-k}$.

Proof. Lemma 9.1 yields the existence of signs $\theta_i \in \{-1,1\}$ for $i \in \{0,\ldots,k\}$ and $\theta_{ij} \in \{-1,1\}$ for distinct $i,j \in \{0,\ldots,k\}$, such that:

$$B(z_0,\ldots,z_k) = \theta_i z_i \quad \text{whenever } z_j = \theta_{ij} \text{ for all } j \neq i. \tag{9.10}$$

Now consider the functions $f_j \in \mathcal{T}(n-k;d_j)$, $j \in \{0,\ldots,k\}$. Lemma 9.3 yields the existence of functions $F_j \in \mathcal{T}(n;d_j)$, $j \in \{0,\ldots,k\}$, such that:

$$F_j(e_i \oplus x) = \begin{cases} \theta_i f_i(x), & i = j \\ \theta_{ij}, & i \neq j \end{cases} \tag{9.11}$$

for all $i, j \in \{0, \ldots, k\}$ and $x \in \{0,1\}^{n-k}$.

For any fixed $i \in \{0, \ldots, k\}$ and $x \in \{0,1\}^{n-k}$, by construction the variables $z_j \coloneqq F_j(e_i \oplus x)$ satisfy the condition in (9.10). Therefore, (9.10) and (9.11) yield:

$$B(F_0, \ldots, F_k)(e_i \oplus x) = B(z_0, \ldots, z_k) = \theta_i z_i = \theta_i F_i(e_i \oplus x) = \theta_i^2 f_i(x) = f_i(x) \tag{9.12}$$

as claimed. □

Armed with this lemma, we can now prove Theorem 6.5.

Proof of Theorem 6.5. Lemma 9.4 demonstrates that for any tuple of functions $(f_0, \ldots, f_k) \in \prod_{i=0}^{k} \mathcal{T}(n-k; d_j)$ there exists a function $F \in \mathcal{T}_B(n; d_0, \ldots, d_k)$ such that $F(e_i \oplus x) = f_i(x)$ for all $i \in \{0, \ldots, k\}$ and $x \in \{0,1\}^{n-k}$. Thus, each component f_i of the original k-tuple can be uniquely recovered from F. Therefore, a map $(f_0, \ldots, f_k) \mapsto F$ (if there are multiple F corresponding to some f, select one arbitrarily) defines an injection from the cartesian product $\prod_{i=0}^{k} \mathcal{T}(n-k; d_j)$ into $\mathcal{T}_B(n; d_0, \ldots, d_k)$, completing the proof. □

As shown in Table 2, there are 16 binary Boolean operators B. Ten of them are irreducible, including AND, OR and XOR and their negations. For each such operator, the Composition Theorem 6.5 gives:

$$\left|\mathcal{T}(n-1; d_0)\right| \left|\mathcal{T}(n-1; d_1)\right| \leq \left|\mathcal{T}_B(n; d_0, d_1)\right| \leq \left|\mathcal{T}(n; d_0)\right| \left|\mathcal{T}(n; d_1)\right| \tag{9.13}$$

Surprisingly, the intersection of all ten classes is still as large.

Proposition 9.5. *We have:*

$$\left|\bigcap_B \mathcal{T}_B(n; d_0, d_1)\right| \geq \left|\mathcal{T}(n-1; d_0)\right| \left|\mathcal{T}(n-1; d_1)\right| \tag{9.14}$$

where the intersection is over the ten irreducible binary Boolean operators.

In particular, there are many functions f (specifically, $2^{2n^2(1-o(1))}$) that can be simultaneously expressed as: $f = f_1$ AND $f_2 = f_3$ OR $f_4 = f_5\, XOR\, f_6$ where all the f_i are linear threshold gates.

Proof. In the proof of the Composition Theorem 6.5 above, we showed that for each irreducible Boolean operator B and pair of functions $(f_0, f_1) \in \mathcal{T}(n-1; d_0) \times \mathcal{T}(n-1; d_1)$, there exists $F \in \mathcal{T}_B(n; d_0, d_1)$ such that:

$$F(0 \oplus x) = f_0(x), \quad F(1 \oplus x) = f_1(x) \tag{9.15}$$

for all $x \in \{0,1\}^{n-1}$. Obviously, this pair of equations defines F uniquely on $\{0,1\}$, and F is independent of B. Thus, F lies in the intersection of $\mathcal{T}_B(n; d_0, d_1)$ over all irreducible B. □

Acknowledgment

Work in part supported by ARO grant 76649-CS and NSF grant 1633631 to PB, and AFOSR grant FA9550-18-1-0031 to RV.

References

[1] Martin Anthony. *Discrete mathematics of neural networks: selected topics*, volume 8. Siam, 2001.

[2] Amy F.T. Arnsten and Francisco X. Castellanos. Neurobiology of attention regulation and its disorders. *Pediatric Psychopharmacology*, page 95, 2010.

[3] Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems*, pages 4331–4339, 2016.

[4] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. January 2015. 3rd International Conference on Learning Representations, ICLR 2015 ; Conference date: 07-05-2015 Through 09-05-2015.

[5] P. Baldi. Neural networks, orientations of the hypercube and algebraic threshold functions. *IEEE Transactions on Information Theory*, 34(3):523–530, 1988.

[6] P. Baldi. *Deep Learning in Science*. Cambridge University Press, Cambridge, UK, 2021.

[7] Pierre Baldi and Roman Vershynin. On neuronal capacity. In *Advances in Neural Information Processing Systems*, pages 7740–7749, 2018.

[8] Pierre Baldi and Roman Vershynin. The capacity of feedforward neural networks. *Neural Networks*, 116:288–311, 2019. Also: arXiv preprint arXiv:1901.00434.

[9] Pierre Baldi and Roman Vershynin. Polynomial threshold functions, hyperplane arrangements, and random tensors. *SIAM Journal on Mathematics of Data Science*, 1(4):699–729, 2019. Also: arXiv preprint arXiv:1803.10868.

[10] Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. *arXiv preprint arXiv:1506.07503*, 2015.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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, June 2019. Association for Computational Linguistics.

[12] Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. *arXiv preprint arXiv:2103.03404*, 2021.

[13] Michael James Fenton, Alexander Shmakov, Ta-Wei Ho, Shih-Chieh Hsu, Daniel Whiteson, and Pierre Baldi. Permutationless many-jet event reconstruction with symmetry preserving attention networks. *Physical Review D*, 105(11):112008, 2022. Also arXiv:2010.09206.

[14] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[15] Laurent Itti, Geraint Rees, and John K Tsotsos. *Neurobiology of attention*. Elsevier, 2005.

[16] Jeff Kahn, János Komlós, and Endre Szemerédi. On the probability that a random±1-matrix is singular. *Journal of the American Mathematical Society*, 8(1):223–240, 1995.

[17] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. volume 97 of *Proceedings of Machine Learning Research*, pages 3744–3753, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[18] Hanxiao Liu, Zihang Dai, David R So, and Quoc V Le. Pay attention to mlps. *arXiv preprint arXiv:2105.08050*, 2021.

[19] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[20] Bruno A Olshausen and David J Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607, 1996.

[21] Michael I Posner. *Cognitive neuroscience of attention*. Guilford Press, 2011.

[22] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[23] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

[24] Alexander Shmakov, Michael James Fenton, Ta-Wei Ho, Shih-Chieh Hsu, Daniel Whiteson, and Pierre Baldi. SPANet: Generalized Permutationless Set Assignment for Particle Physics using Symmetry Preserving Attention. *SciPost Phys.*, 12:178, 2022.

[25] Mohammadamin Tavakoli, Forest Agostinelli, and Pierre Baldi. Splash: Learnable activation functions for improving accuracy and adversarial robustness. *Neural Networks*, 140:1–12, 2021.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[27] Daniel LK Yamins and James J DiCarlo. Using goal-driven deep learning models to understand sensory cortex. *Nature neuroscience*, 19(3):356–365, 2016.

[28] David Zipser and Richard A Andersen. A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons. *Nature*, 331(6158):679–684, 1988.

[29] Yu A Zuev. Asymptotics of the logarithm of the number of threshold functions of the algebra of logic. *Soviet Mathematics Doklady*, 39(3):512–513, 1989.

[30] Yu A Zuev. Combinatorial-probability and geometric methods in threshold logic. *Diskretnaya Matematika*, 3(2):47–57, 1991.