Skinny Networks with top-down Attention

VENKATESH MEDABALIMI

In this short note I wish to bring to your attention how a certain kind of *skinny* but *deep* representations are universal, discuss their interesting properties and motivate methods for learning such architectures. This might be of benefit in resource(memory) limited devices and may lend well to on-chip implementations of deep models, more so whenever the length of the representation also scales well with that of the input. In a recent result Johnson[6] shows that deep, skinny neural nets are not universal approximators. While it was all along known from classical results that neural nets are universal approximators [2, 4],(and results for generative models [12, 8, 7]) this is a welcome inroad into understanding the representative power of neural nets in light of their shape and effectively says no amount of depth can compensate for a lack of width[1]. In contrast to this main result from [6] the representation we discuss uses constant width but *allows* for access to individual input variables at intermediate layers, like in a top down attention mechanism over the input. These representations are also of immediate relevance to neural nets with polynomial activations and alternate architectures called sum product networks(SPNs). The main insight we discuss arises from a theorem of Ben-Or and Cleve's [1] about algebraic formulas more familiar in TCS community. In only other related work we are aware of, exploring expressivity in the context of limited width neural nets a theorem of Rojas [11] from 2003 shows that by stacking one-dimensional perceptron layers, each of which is connected to all previous layers, one can obtain any 2-way *classifier*.

Rest of this write-up is organized as follows, first in section 1 we recall a celebrated result of Ben-Or and Cleve[1] on in-place computation of algebraic formulas using straight line programs operating on a constant number of registers and in section 2 describe an interesting characterization of such computations as an iterated matrix product of constant size near-identity matrices. By Stone-Weierstrass, since for any continuous function over a compact domain there exists a multivariate polynomial approximation it follows that these skinny representations are in fact universal. We conclude by listing out advantages of this architecture and a discussion of related works. An interesting challenge presents itself in figuring out how to learn these architectures.

1. Computation using constant number of registers

A couple of definitions before we recall Ben-Or and Cleve's result.

Date: September 25, 2019.

[1]checkout this Quanta article

Definition 1. *Straight-Line Program : A striaght line program over a ring $(\mathcal{R}, +, .)$ is a sequence of assignment statements of the form*

$$R_j \leftarrow R_j + c.R_i$$
$$R_j \leftarrow R_j - c.R_i$$
$$R_j \leftarrow R_j + x_u.R_i$$
$$R_j \leftarrow R_j - x_u.R_i$$

where $R_1, R_2, .., R_i, R_j, .., R_w$ are the registers and $x_1, x_2, ..x_u, ..x_n$ are the variables. Width w of a program is the number of registers it contains and length l is the number of statements or instructions.

Definition 2. *In-place computation: We say that a straight line program in-place computes $f(x_1, x_2, .., x_n)$ in a way that offsets R_j by $f.R_i$ if the values of the registers are transformed by the program as follows. The value of R_j is incremented by $R_i.f(x_1, x_2, .., x_n)$ and all other registers retain their initial value at the end of the program execution.*

Theorem 3 (Ben Or and Cleve [1])**.** *Over any ring $(\mathcal{R}, +, .)$ any formula $f(x_1, x_2, x_3, ..., x_n)$ of depth d can be in-place computed by a straight line program of width 3 and length 4^d.*

Proof. The essence of Ben-Or and Cleve's proof is to be able to perform in-place addition and multiplication using at most constant number of registers and then apply it recursively. Only 3 registers suffice. As an induction assumption assume we can in-place compute f in a way that offsets R_i by fR_j or $-fR_j$ as we desire and similarly we can in-place compute g in a way that offsets R_i by gR_j or $-gR_j$. Then we can in-place compute $f + g$ and $f \times g$ as follows.

- In-place computation of $f + g$

$$\text{Offset } R_1 \text{ by } fR_2 : \qquad R_1 \leftarrow R_1 + fR_2$$
$$\text{Offset } R_1 \text{ by } gR_2 : \qquad R_1 \leftarrow R_1 + gR_2$$

  This sequence of instructions offset R_1 by $(f + g)R_2$ leading to register states $h_1 + (f + g)h_2$, h_2 and h_3 if we start out with h_1, h_2 and h_3 in the register R_1, R_2 and R_3 respectively.

- In place computation of $f \times g$ can be done as follows.

$$\text{Offset } R_3 \text{ by } -gR_2 : \qquad R_3 \leftarrow R_3 - gR_2$$
$$\text{Offset } R_2 \text{ by } fR_1 : \qquad R_2 \leftarrow R_2 + fR_1$$
$$\text{Offset } R_3 \text{ by } gR_2 : \qquad R_3 \leftarrow R_3 + gR_2$$
$$\text{Offset } R_2 \text{ by } -fR_1 : \qquad R_2 \leftarrow R_2 - fR_1$$

  This seqeunce of instructions offsets R_3 by $f.gR_1$ leading to register states h_1, h_2 and $h_3 + (f \times g)h_1$ if we start out with h_1, h_2 and h_3 in the registers R_1, R_2 and R_3 respectively.

The base case involving the leaves is easy to verify. Since the number of instructions used per level of recursion is at most 4, the length of the straight-line program is at most 4^d where d is the depth of the original formula. □

2. Narrow iterated matrix product reperesentation

For the purpose of this section the reader can think of being given a arithmetic formula of depth d that computes a multivariate polynomial f. Then lets say we have a straight-line program for this formula coming from Thm.3. The action of any instruction in the straight line program on the 3 registers can be captured by a matrix multiplication. For instance, the straight line instructions $R_1 \leftarrow R_1 + x_u.R_3$ and $R_1 \leftarrow R_1 + w_{i,j}.R_2$ correspond to multiplication by matrices $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_u & 0 & 1 \end{bmatrix}$ and $\begin{bmatrix} 1 & 0 & 0 \\ w_{i,j} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ respectively. If we start with R_1, R_2, R_3 initialized to the values $0, 1, 1$ respectively, then in-place computing the f in a way that offsets R_1 by $f(x) \times R_2$ amounts to computing $f(x)$ in the register R_1. Since we have a straight line program of instructions that achieves this, we can write it down as an iterated matrix product of the vector $[R_1\ R_2\ R_3] = [0\ 1\ 1]$ by the corresponding instruction matrices. Note that since each instruction uses only one of the variables x_i or weights $w_{i,j}$ apart from the registers as described above, each matrix is essentially one non-zero element away from identity as shown in the examples above. For a contrast, there are no explicit non-linear activations between layers as is the case with deepnets but, the input x is read potentially multiple times at different steps in the iterated matrix product, the same applies to individual weight parameters. In the same vein, a key distinction from deep *linear* nets is that some of these matrices depend on input variables $x_i, i \in [n]$ while the rest depend on the weight parameters w and these two kinds of near-identity matrices are interleaved as dictated by the order of appearance of corresponding instructions in the straight-line program.

One can interpret the model as an iterated matrix multiplication by matrices of the form $I + I_t.(a_t^w.(c_t.w) + b_t^x.((1 - c_t).x))$ where I_t is a 3×3 influence matrix with a single entry set to 1, in a way this selects which part of memory(internal) influences which other part in this step. a_t^w is an attention vector for selection over the parameter space w at time/layer t and similarly b_t^x is an attention vector for selection over the input space x at time t. See figure 1. The variable c_t denotes a choice parameter which when in $\{0, 1\}$ tells if the current layer wants focus on the input space x or the parameter space w to influence the computation. Note that the representation we used for showing universality is quite sparse and in fact the attention vectors would select a single parameter or input coordinate. The discrete nature of choice of non-zero entry in the inlfuence matrix I_t prompts us to think if methods like REINFORCE in conjunction with usual techniques are more appropriate. However it is a very interesting line to explore if one can facilitate learning these representations in other ways.Obviously one can define these representations with matrices of larger

dimension, allowing for the computation within to use more (internal)memory. Once trained, these representations also seem to present a potential opportunity for explaining or reasoning difference in classification of two different input instances x, y by identifying points of significant divergence in memory traces and the corresponding relevant input regions causing it via top-down attention. Since all computations start with the identically initialized registers this is always guaranteed to occur.

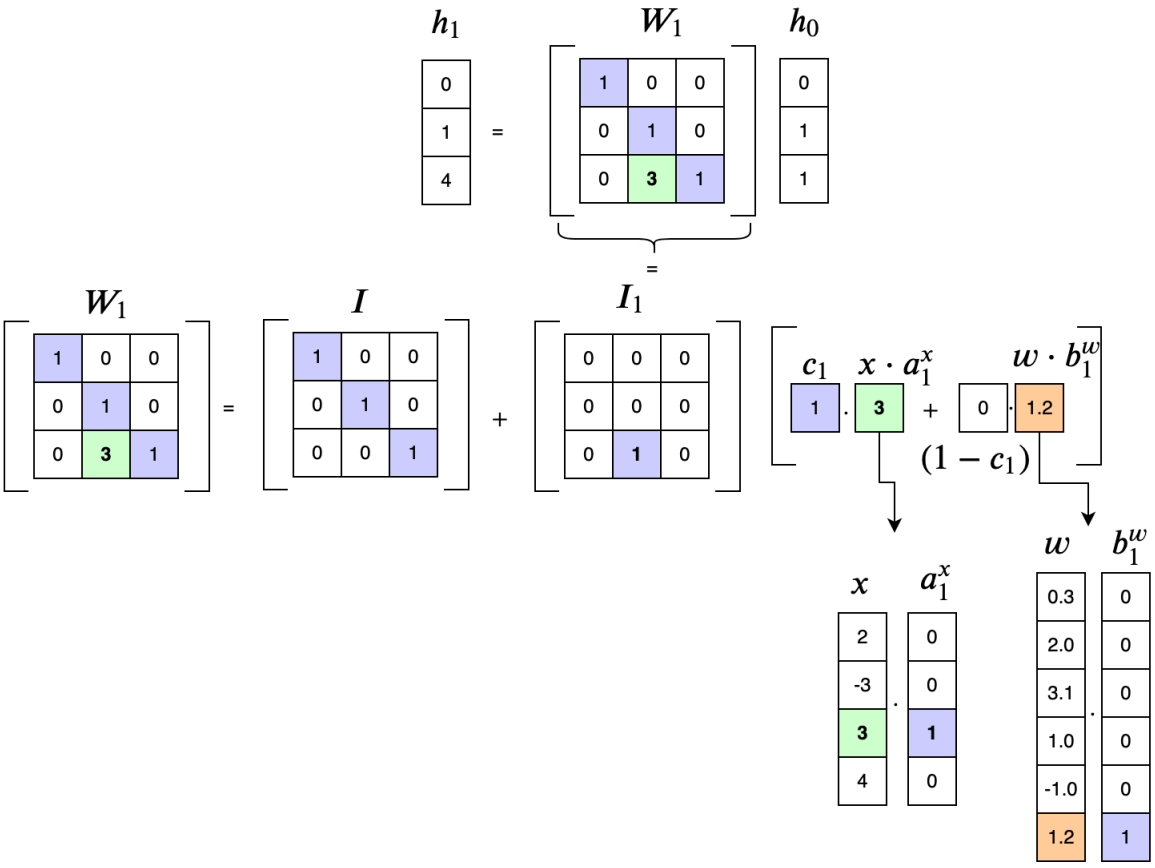

FIGURE 1. An illustration of the architecure's basic piece where each weight matrix is constructed using attention over weights and inputs. The influence matrix I_t can be determined as an action which is a function of the time step t by a policy network. Image Credit: Thanks to Harris Chan.

3. Advantages of our Architecture

We briefly summarize incentives for devising methods for learning a model of our kind below.

- Present a promise of interpretability via contrasting the input patches attended to when memory traces diverge significantly.

- Once trained the represenations are written solely as an Iterated Matrix multiplication, presence of no other intermediate non linearities makes it conducive for new hardware relying solely on matrix multiplications.
- Parallelism at inference time. Inference in these representations is parallelizable in the sense that the second half of the matrix product need not weight for the first half to be computed and so on recursively, since there's no need of "knowing" or waiting for activations once the transition matrices are available.
- The model is reversible.
- Space required for storing activations is rather small.

The **challenge** in devising learning methods for our architecture it appears lies in the interplay between two aspects. There are discrete choices to make at each layer which involve picking an input coordinate or a weight parameter and the influence matrix I_t. Then there's the question of optimizing over these parameters which appears to lend to more conventional methods like gradient descent. At first glance it seems sensible to optimize over one while the other is at the best possible setting and vice versa. Having these inter-dependent schematics appears to make things tricky. Probably one way is to learn a continuous embedding for discrete choices by training a performance evaluator and then perform alternating minimization by coordinate descent on the continuous embedding and weight spaces. But, its hard to imagine being able to train such a performance predictor.

4. Related Work

Our architecture motivated by the representation used in the theorem and depicted in figure1 is unique in many ways. It has interesting properties like weight and input reuse across layers and facilitation of multiplicative interaction of intermediate activations with the input by having some weight matrices be functions of the input. Hypernetworks[3] share the former aspect of weight sharing across layers in the form of a single net that generates weight matrices as a function of layer or time index, the input at the time and other information. Hypernets are inspired by methods in evolutionary computing like HyperNeat that seek to deal with large nets containing millions of weight parameters by having a small net to efficiently generate weight parameters. The mutliplicative interaction obtained by synthesizing weight matrices via attention over the input is remniscent of the mulitplicative RNN [13].

Architectures like Dense nets[5] make the input layer(and also all intermediate layers) available in all the subsequent layers via feature concatenation allowing for implicit multiplicative interactions. The difference lies in how in our case the input layer is used to construct matrices to multiply with-as opposed to have it "operated upon" to build new features.

To the best of our knowledge Lin and Jegelka [9] were the first to explore the power of narrow nets with skip connections. They show Resnet with one-neuron hidden layers is a universal approximator. Broadly speaking there are two main differences with the model in [9] and that motivated in this paper. As is the case with usual ResNets their model has skip connections between successive blocks but in the model discussed it is always the input layer that is reused. However this reuse is not

in the usual sense of a skip connection that gets added for the vector to be operated upon but to construct matrices using input layer to multiply the current features or memory with. More crucially the source of non-linearity in their model are the single RELU unit in their Residual block between wider layers whereas in the model I describe since there are two kinds of matrix multiplications that are interleaved - one with usual weight parameters and the second with matrices constructed using the input layer, non linearities can be seen as primarily arising out of the second type of multiplications.

There's a growing body of work in computer vision that is using attention in ingenious ways seeking inspiration from biological vision. For example works such as [10] seek to model vision as a sequential decision problem over fixations. Apart from the evident differences in these models to ours, a crucial difference with such sequential attention based processing systems is that the previous glimpses and computations upto time t do not influence the location of the next glimpse if we were to interpret the input dependent matrices as being derived from correspondng glimpses. In particular while the locations of sequence of glimpses can be task dependent it does not adapt to a specific input.

Acknowledgement Thanks to Harrris Chan for the illustration in page 4 and bringing some of the related works to our attention.

References

[1] Michael Ben-Or and Richard Cleve. Computing algebraic formulas using a constant number of registers. *SIAM Journal on Computing*, 21(1):54–58, 1992.

[2] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[3] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. *CoRR*, abs/1609.09106, 2016.

[4] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[5] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[6] Jesse Johnson. Deep, skinny neural networks are not universal approximators. *CoRR*, abs/1810.00393, 2018.

[7] Nicolas Le Roux and Yoshua Bengio. Representational power of restricted boltzmann machines and deep belief networks. *Neural computation*, 20(6):1631–1649, 2008.

[8] Nicolas Le Roux and Yoshua Bengio. Deep belief networks are compact universal approximators. *Neural computation*, 22(8):2192–2207, 2010.

[9] Hongzhou Lin and Stefanie Jegelka. Resnet with one-neuron hidden layers is a universal approximator. In *Advances in Neural Information Processing Systems*, pages 6169–6178, 2018.

[10] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[11] Raúl Rojas. Networks of width one are universal classifiers. In *Proceedings of the International Joint Conference on Neural Networks, 2003.*, volume 4, pages 3124–3127. IEEE, 2003.

[12] Ilya Sutskever and Geoffrey E Hinton. Deep, narrow sigmoid belief networks are universal approximators. *Neural computation*, 20(11):2629–2636, 2008.

[13] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.