Modularized Internal Attention Network: Draft

Morgan Bryant
Department of Psychology
Stanford University, Stanford, CA, 94305 USA

February 12, 2017

Abstract

I outline a first draft at a novel neural network that uses a hidden attention mechanism to selectively process information within the network. This contrasts with Recurrent Attention Models [3] by managing attention not in visual glimpses trained with reinforcement learning signals but instead on its internal layers. The *modules* unique to this architecture are differentiable and allow the network to be trained with standard gradient methods, without need for a reinforcement learning signal.

1 Introduction

This kind of modularized internal attention network has the potential for:

- Possibly make networks faster at learning;
- Speed up test-time computation time;
- More easily transfer;
- Shed intuitive light on the processes of the network;
- Reconcile an aspect of formalism and connectionism;

- Potentially indicate a style of model structuring and learning that can match current neural networks as well as provide a model that more realistically mimics neurobiology.

The components that yield these results are the in-network attention selector and the module table. The attention selector is a portion of a layer output within the network that is treated as a command for processing as opposed to a set of parameters for a fixed computation. The module table is a collection of *modules* or small fully-connected layers that act like mini networks by encoding often-repeated operations or simple programs. But, critically, these programs are hidden: unlike the programs in the *Neural Programmer Interpreter* [4], these programs are not explicit about the action that they perform.

The hypothesis of this report is that the modularity of the network will encourage a more collaborative and specialized utilization of its resources in parallel, while still preserving the extreme function learning that made neural networks famous by promoting hidden units as opposed to formal symbolic units.

Besides the above primary hypothesis, it might seem intuitive that this kind of network might also transfer better to new tasks or be wholly transplanted into newly initialized networks. Such a claim may be supported by the ability to freeze module layers, zero-out module weights, or simply transfer the module layers.

2 Formal Model

The model can be conceived as a collection of pairs of layers, where one layer is a module and the other is a fully connected layer. Call the output of the previous layer the matrix $X \in \mathbb{R}^{N \times (B+\theta)}$ for any whole constants N, B, θ with the following characterizations: θ is small and represents the parameters to modules; B is arbitrary and relatively large and is the number of modules; and N is somewhat small and indicates the number of modules that the subsequent layer will use.

Then, there are two approach that may work:

- ***Updated Approach*** The actual approach that works is: For a layer $X \in \mathbb{R}^{N \times (B+\theta)}$, split X into $[X_B \ X_\theta] = X$, where $X_B \in \mathbb{R}^{N \times B}$ and

$X_\theta \in \mathbb{R}^{N\times\theta}$. Take the Module tensor (order 3) $M \in \mathbb{R}^{\theta\times B\times\phi}$ and choose one of two routes:

1. Make N copies of M : $\underline{M} \in \mathbb{R}^{N\times B\times\theta\times\phi}$. Make $Z = \underline{M} \cdot \sigma(X_B)$ where σ is the softmax function, and then make $Y = ZX_\theta$. Here, $Z : \mathbb{R}^{N\times\theta\times B\times\phi} \times \mathbb{R}^{N\times B} \mapsto \mathbb{R}^{N\times\theta\times\phi}$ and $Y \in \mathbb{R}^{N\times\phi}$. This approach operates each module in M each of the N sets of parameters $\underline{x}_\theta$. The Z is the candidate X_θ-parameterized module instance for each possible module, agnostic of the respective choices of modules X_B. Then, Z is reduced and summed according to module selections with $\underline{x}_B$. Finally, Y is mapped via a fully-connected layer to the next X' via $FY = X'$, $F \in \mathbb{R}^{(N\times\phi)\times(N'\times(\theta+B))}$. (A reasonable extension is to append to Y the selector matrix X_B via $Y' = [Y\ X_B]$ such that $X' = F'Y'$, $F' \in \mathbb{R}^{(N\times(\phi+B))\times(N'\times(\theta+B))}$.)

   This method is highly distributed: while the typical interpretation of a module system selects modules first to send parameters second, the scheme outlined here allows for a sum of weighted partial-computations using multiple modules; while this can be thought of as a partially-learned, unstable state, alternatively this could also be an learned combination.

2. The above system is highly distributed and permits computations to be compiled in pieces, intentionally or not, for final results. But the intuitive understanding is that modules will be fine-tuned for specific tasks and the network sends queries into individual modules. That is, the softmaxed module selector vectors $\sigma(X_B)$ are expected to be disproportionately one-hot. Due to this expectation [to be confirmed via tests] and the fact that the version (1) above is expensive per layer, a reasonable variant is the following. This version is explicitly less general due to the space of post-normalized X_B being the space of one-hot vectors instead of the space of categorical distributions, but if the two are reasonable approximations to each other, then $Y \approx W$. This approximation weakens as B grows.

   For each n from 1...N, take $i = \underset{i}{\text{argmax}}(\underline{x}_n)$ be the index of the largest module-selecting value in, using numpy index notation, $X_B[n]$. Take M_i^n be the i^{th} module in M, $M_i^n \in \mathbb{R}^{\theta\times\phi}$. Concatenate all these $M' = [M_i^1\ M_j^2\ ...\ M_k^N]$, for $i, j, ..., k$ each

drawn from $[1...B]$. Note that $M' \in \mathbb{R}^{N \times \theta \times \phi}$. Make $W = M' X_\theta$, where $W : \mathbb{R}^{N \times \theta \times \phi} \times \mathbb{R}^{N \times \theta} \mapsto \mathbb{R}^{N \times \phi}$. Alternatively, think of the N distinct $W_n = M_i^n X_n$ where i varies by n. Both versions happen via N distinct $[\theta \mapsto \phi]$ $2^{nd}-$order matrix multiplications. Finally, this W is fully connected to the next X' with an $FC \in \mathbb{R}^{(N \times (\phi + B)) \times (N' \times (\theta + B))}$ when you include the hardmaxed or softmaxed X_B in the operand as well as W.

A complexity analysis for one layer pair Version (1) is constrained by the operation $\underline{M} X_B^\sigma$ which is $O(N \cdot Tens(B, \theta, \phi))$, where $X_B^\sigma = \sigma(X_B) \in O(NB)$ of N softmaxes over B elements each, and where $Tens(B, \theta, \phi)$ is runtime complexity of a $3^{rd}-$order general Tensor multiplication, which at worst is naively implemented in $B \cdot \theta \cdot \phi$, meaning version (1) is, for general values, $O(N \cdot B \cdot \theta \cdot \phi)$. The memory complexity is $O(B \cdot \theta \cdot \phi)$ since the N desired module operations can be processed in parallel or in sequence, and are independent of each other.

Version (2) is constrained by $M' X_\theta$ which is $O(N\theta\phi + B)$. The gains in this version is in a multiplicative factor of B, substituted for an additive correlate, due to the argmax operations. Besides this, the concatenation operations in (2) may have overhead but they do not increase complexity besides an additive (hence, negligible) factor of N. The memory complexity is $O(\theta\phi + N)$.

Notably, the complexity gain in B signals two points: First, that B processing costs can be traded off when desired, such as: use version (1) during train time to learn smoothly and (2) at test time in a real-time application. Or, alternatively, (2) could be used for faster training per amount of time, and (1) is used at test as a "bag of experts". Both versions are also constrained by the fully-connected multiplication, which can be done in $O(N \cdot N' \cdot (\theta + B))$ naively or in $O(N^{2.376} \cdot (\theta + B))$ if $N = N'$ using the Copper-Winograd algorithm [10]. Both versions have a memory complexity here of $O(N \cdot N' \cdot (\theta + B))$ naively.

- ***Fully Distributed Approach*** For each $1 <= i <= N$, take $X_i \in \mathbb{R}^{\theta + \mathbb{B}}$ and let $\underline{x}_\theta = X_{i,1:\theta}$, or the first θ entries of X_i, and let $\underline{x}_B = X_{i,\theta+1:\theta+B}$ be the last B entries of X_i. Make $Z_i = \underline{x}_\theta \cdot f(\underline{x}_B))^T$ such that $Z_i \in \mathbb{R}^{B \times \theta}$ and where f is a function that ensures that the vector is normalized and nonnegative, such as a normalize $\circ$ relu or a softmax, and preserves the dimensionality B. ([7] suggests that a softmax is a

valid choice for f by giving a probabilistic interpretation to the gating networks.) The outer product between the $\underline{x}_B$ and $\underline{x}_\theta$ are, intuitively, the weighted selections of where to send the given parameters. Then, make $Y_i = MZ_i$, where M is a 3-dimensional tensor of the B modules that each sends $\mathbb{R}^\theta \to \mathbb{R}^\phi$ and constitutes the "first" layer of the pair of layers.

The nonlinearity is useful since $\underline{x}_B$ represents the network's choice for which module to channel the parameters $\underline{x}_\theta$, and as such, may be best represented as if they were probabilities.

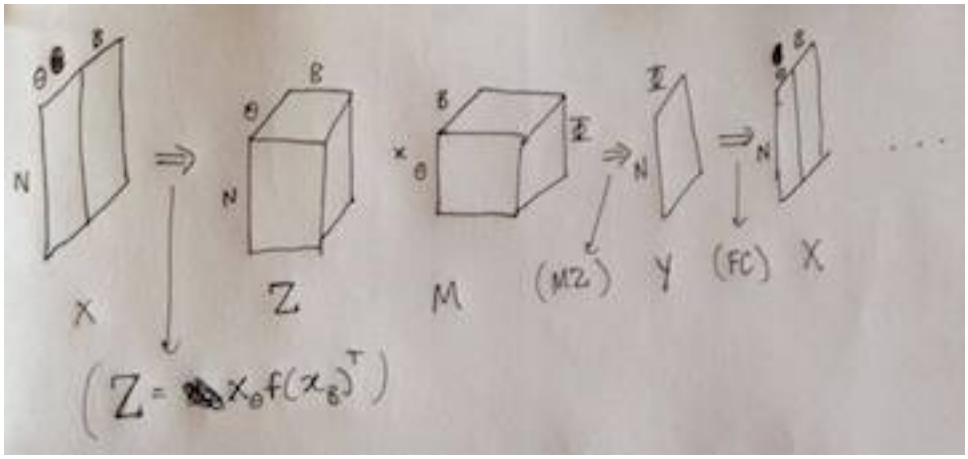

Figure 1: A visualization of the flow of computation used in the distributed approach

- ***Selective Approach*** The approach outlined above takes a distributed approach to selecting attention modules to direct computation flow. Alternatively, if instead modules are uniquely selected, these modules would be [both intuitively inclined towards being less hidden? ... and] more optimized for computation.

  For each $1 <= i <= N$, similarly take $X_i \in \mathbb{R}^{\theta+\mathbb{B}}$ and let $\underline{x}_\theta = X_{i,1:\theta}$ and $\underline{x}_B = X_{i,\theta+1:\theta+B}$. Make $Z_i = \underline{z}_i = f(\underline{x}_\theta)$, where f is a function that normalizes but is not necessarily make nonnegative. Then, make $Y_i = M_j \underline{z}_i$, where $j = \underset{1<=j<=B}{\operatorname{argmin}} \underline{x}_B$.

  Notice that this model replaces a rank-3 and rank-2 calculation with a rank-2 and rank-1 product and eliminates an outer product calcula-

tion. However, it sacrifices total distributivity, a property that tends to benefit neural networks, as well as complicating the training of unselected modules, since [unsure, but seems perhaps right:] it becomes statistically less likely for the least-selected modules to receive much selection or training after the primary selected modules are trained.

Finally, combine (and flatten) the Y_i into a vector $\underline{y}$ in $\mathbb{R}^{N\times\phi}$, and feed $\underline{y}$ to a fully connected layer $\mathbb{R}^{N\times\phi} \to \mathbb{R}^{N'\times(B'+\theta')}$ (the "second" layer of the pair) with a nonlinearity, the output of which is treated as the input to the next module.

In both models, a fully connected layer is useful in letting the network use the module outputs in concert. A reasonable modification, in fact, is to preserve the $X_B \in \mathbb{R}^{N\times B}$ as an additional residual input to the fully connected layer [9].

3 Discussion

This model effectively generates (in X) N module computations, where $\underline{x}_\theta$ are the parameters (of fixed size) and $\underline{x}_B$ indicate which module(s) each set of parameters should be fed to. $\underline{x}_\theta$ correspond to standard neural network parameters passed from layer to layer, so these easily fall in the backpropagation paradigm. But unlike other internal attention-selective models as in [2,3], the $\underline{x}_B$ module vector is simply another vector output by the system, orthogonal to the weights they manage, and thus can be trained "independently" of each of the parameters $\underline{x}_\theta$ that are fed to it, but remain sensitive to the types of parameters fed to it. These vectors are easily differentiable and thus can be backpropagated through – and without any special machinery. {Draft note: I still have to work out these derivatives.}

Another point of note is that the modules, as the word implies, are building blocks that can be somewhat independently considered. Benefits to this aspect include the following:

- If module learning was a desirable task, a multi-layer network could utilize the same module matrix/tensor for each of the computations and thus provide much more signal to the modules.
- The modules could be incorporated into a recurrent neural network rather naturally, since it implicitly changes the computation it performs

based on context, a problem that naive recurrent networks struggle with.

- If certain properties were going to be examined as bottlenecks in a network, any module set could be hand-designed to test certain characteristics (eg, a quick test to determine borderline expressivity could be zeroing out elements of modules by their index) or even wholly transplanted with an alternative system (ie, something more involved than a simple linear operation M such as another neural network). The module is only conditioned on being a function that does $\mathbb{R}^\theta \to \mathbb{R}^\phi$.

- The contents of trained modules can be examined in isolation for what they do: modules can be subjected to tests to determine the function they have – though, since they are presumably hidden unless they are hand-designed, these would probably not yield comprehensible results in isolation except coincidentally.

- As already stated, module tensors can be lifted out of a trained network and placed in a newly initialized network to potentially speed learning. In a different mode, a network can be retrained on learned modules, subject even to a genetic algorithm to improve the modules.

4 References and Related Literature

1. Michael I Jordan and Robert A Jacobs, "A Competitive Modular Connectionist Architecture", MIT, MA, USA, https://papers.nips.cc/paper/430-a-competitive-modular-connectionist-architecture.pdf

2. Marijn F. Stollenga, Jonathan Masci, Faustino Gomez, Juergen Schmidhuber, "Deep Networks with Internal Selective Attention through Feedback Connections", IDSIA, Manno-Lugano, Switzerland

3. Volodymyr Mnih, Nicolas Heess, Alex Graves, Koray Kavukcuoglu, "Recurrent Models of Visual Attention", Google Deepmind, Mountain View, CA, USA

4. Reed, Scott, and Nando De Freitas. "Neural programmer-interpreters." arXiv preprint arXiv:1511.06279 (2015), Google Deepmind, Mountain View, CA, USA

5. Gregor, Karol, et al. "DRAW: A recurrent neural network for image generation." arXiv preprint arXiv:1502.04623 (2015), Google Deepmind, Mountain View, CA, USA

6. Duda, R.O. & Hart, P.E. (1973) Pattern Classification and Scene Analysis. New York: John Wiley & Sons.

7. McLachlan, G.J. & Basford, K.E. (1988) Mixture Models: Inference and Applications to Clustering. New York: Marcel Dekker.

8. Jacobs, R.A., Jordan, M.I., & Barto, A.G. (1991) Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. Cognitive Science, in press. Found from (1) above; this is an alternative modular architecture on the same what/where tasks.

9. He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.

10. Coppersmith, Don and Winograd Shmuel, "Matrix multiplication via arithmetic progressions", Journal of Symbolic Computation, Volume 9, Issue 3, 1990, Pages 251-280, ISSN 0747-7171, <http://dx.doi.org/10.1016/S0747-7171(08)80013-2>