Introduction

A defining feature of human intelligence is the ability to apply knowledge and techniques learned on one problem to other problems which have similar components. This ranges from the simplest examples (e.g. playing chess with different-colored pieces than you learned with) to the most complicated (like applying concepts from one domain of mathematics to another). This class of problems is referred to variously as “transfer learning”, “multi-task learning”, or “lifelong learning”, with minor variations implied by each name.

To date, machine learning algorithms have struggled with transfer learning tasks. In some cases existing techniques perform so poorly as to exhibit “negative transfer”, in which seeing examples across new domains causes the learner to perform worse than it did before.[1]

With the recent success of deep learning methods in many fields, efforts have been made to apply deep learning techniques to transfer-learning tasks. Deep learning is at the deepest level a method for hierarchically extracting good representations from complex data, with the higher levels of a network capturing increasingly abstract representations of the data. As such, deep learning seems naively to be a promising direction for transfer learning; abstract representations of the data should be useful for many related tasks, and the network should be able to simply not use any which are not helpful.

This theory has been borne out for simple, highly coupled tasks such as evaluating sentiment of reviews for different categories of products.[2] A more wide-ranging survey of deep learning methods for transfer learning shows that some classes of models are able to improve their performance on the original, clean dataset after being shown perturbed or distorted versions of the same data.[3]

However, even small changes in the task result in substantial changes to the optimal features, especially at high levels of the network.[4] This can lead to *catastrophic forgetting*, in which the network “unlearns” its original task as it trains on a new one.

Attentional models

Recently a very powerful new class of neural networks known as attentional models have shown success on a variety of difficult tasks. The Neural Turing Machine[5] (NTM) performs complex operations on sequences such as sorting or repeatedly copying by using a differentiable addressing mechanism to read and write to an external memory. DRAW[6] uses a differentiable attention system to selectively read from its input image and write to an output canvas.

These systems work by having a Long Short-Term Memory[7] (LSTM) controller which produces a weighting distribution over the cells of data structure it indexes. This weighting for each cell is, with some variations in each system, multiplied by the content of the cell and then summed across cells (when reading) or multiplied by the value to write and then added into the cell (when writing).

This attention system allows the network to sequentially

1. Decide on a region of the data to focus on, then
2. Read or edit just that region of the data.

By letting the network sequentially attend to a particular region of the input, this design gives a small network the same power as an enormous one with fixed connectivity. Furthermore, since such a system can decide from one timestep to the next what part of the data is the most important, it can effectively ignore the less diagnostic components of an input or output (e.g., never look at the background of an image).

These models take inspiration from the behavior of the primate visual system. In the brain, there are top-down connections from higher levels which act to strengthen or suppress the activation of the neurons in the lower layers.[8] These top-down connections allow the animal to selectively receive more information about an object of interest and less about the background or any distractors. Similar connections have been observed in higher regions of the brain, and these have been speculated to have a similar role in priming and inhibiting the activation of various thought processes.

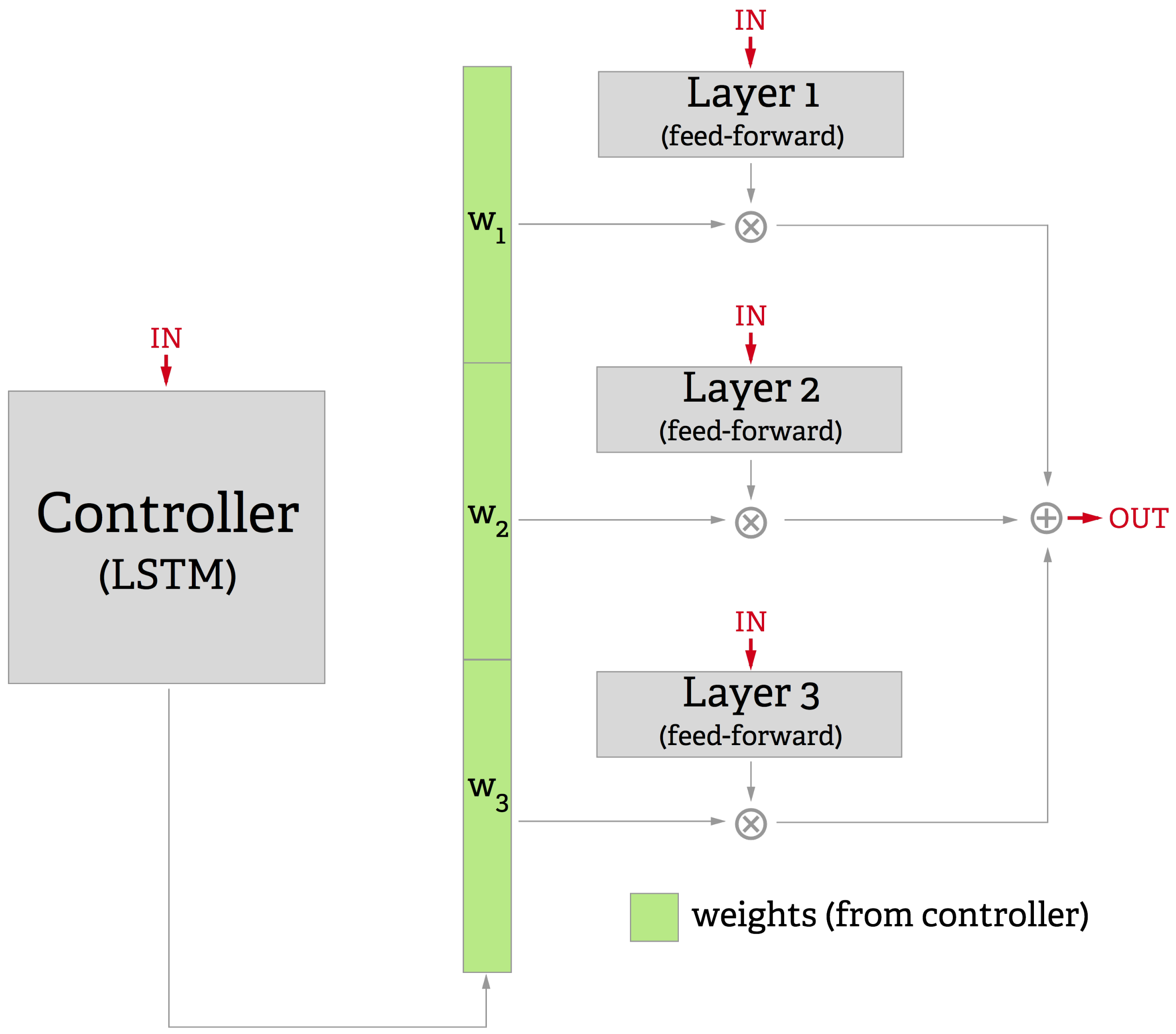
In this paper, the author proposes a model for multitask learning which addresses the challenge of reusing whichever subroutines are shared between several different tasks without forgetting those subroutines the tasks do not have in common. The proposed system takes the form of an attentional model over components of the network itself.

Model

The proposed model generates an output for a particular timestep via the following steps:

1. The input tensor is fed into the controller
2. The controller decides which layers are most appropriate for processing this input
3. The controller outputs a weighting vector reflecting how much output it wants from each of the layers

4. The input tensor is fed into each layer (in parallel)
5. The outputs from each layer are multiplied by their respective weights from the controller
6. The weighted outputs from all the layers are summed together and output. This is the output of the whole network for this timestep.



Essentially the idea is that at each timestep, the controller examines the input that it gets, then up- or down-regulates the activities of the various “functions” (single-layer NNs) to best deal with this input.

Since the controller is an LSTM, it can store information about the inputs it has received before, meaning that in a time series or language setting it can make weighting decisions contextually.

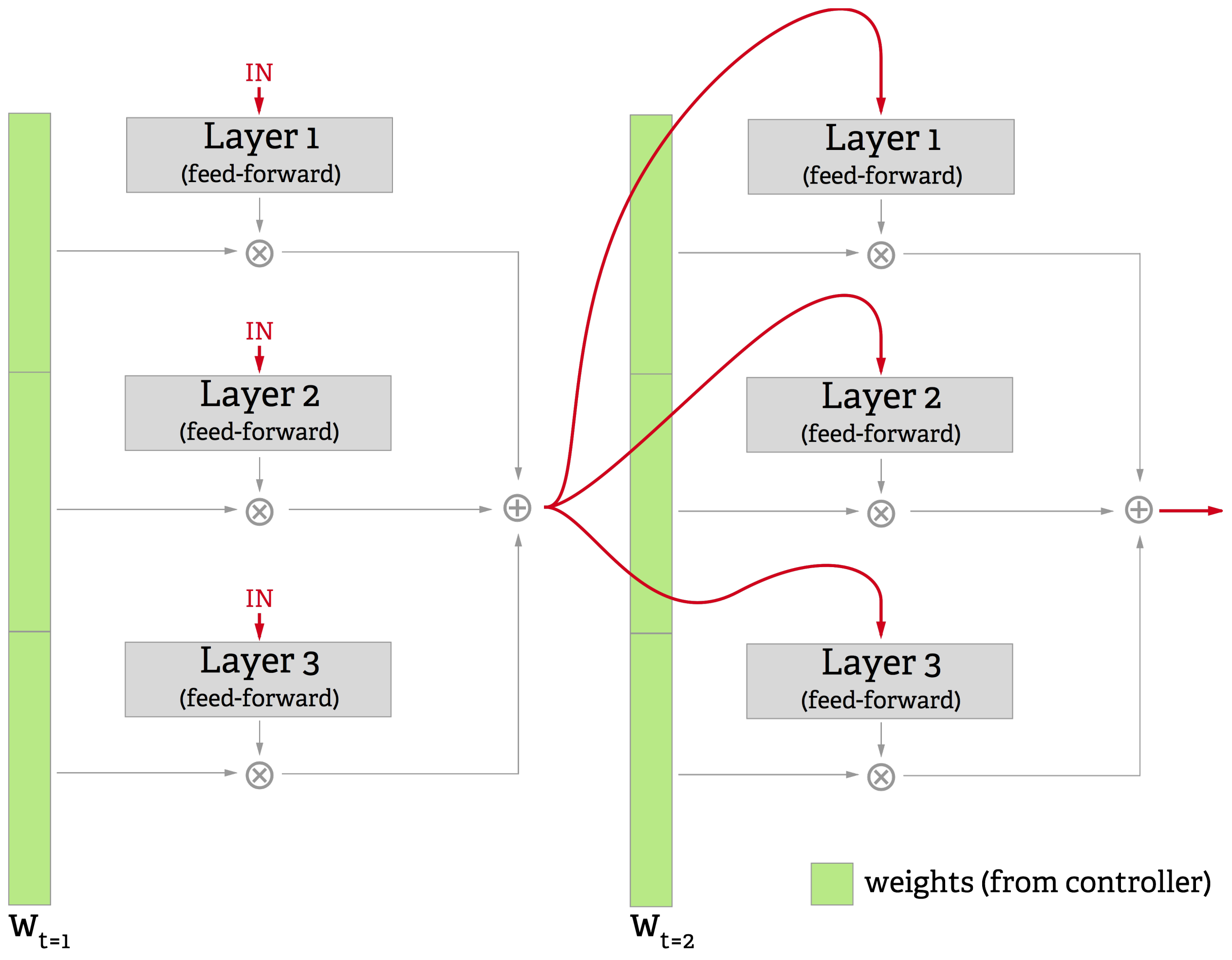
As this model is differentiable throughout, it can be trained with the standard backpropagation through time (BPTT) algorithm for stochastic gradient descent.

By setting weights over each of the layers in the network, the controller scales not only the output of each layer, but also the error gradient that it receives. This means that in a given timestep, the layers which have very low weights on their output will be nearly unchanged by the learning process. That is, functions which are not used are not forgotten.

In an ordinary feedforward neural network, the only way for the network to prevent learning in a particular node is for it to learn connection strengths very near zero for that node. This takes many training examples, and functionally removes that node from the computation graph.

This system, by comparison, can decide that a set of nodes is or is not relevant on an input-by-input basis.

Multi-step variant



One obvious question to consider about this model is, "What happens if the correct output function at a timestep is not computable with a linear combination of single-layer networks?" After all, there are functions computable by a polynomial-width network of depth `k` that require exponential width to compute with a network of depth `k-1`.[9]

To address this question, the system could be run for a predefined number of steps between outputs. That is,

1. Feed the system the input for "real-world" time `t` and save the output
2. Repeat `k` times:
    1. Feed the system its own most recent output
3. Take the `k`$^{\texttt{th}}$ output of the system as its answer for time `t`
4. Repeat from 1. for time `t+1`

This amounts to making the network deeper, in that more layers of computation and nonlinearity lie between the input and the output. This gives it the same computational power of a `k`-depth model.

Function reuse

The fundamental problem this model attempts to address is that of *function reuse*. Given that two tasks require some of the same computation to solve, can we build a system that only has to learn that computation once?

If this system has learned a function in one task which happens to also solve another task, the error gradients propagated to the controller will consistently increase the weight assigned to that function and decrease that assigned to the other functions. But this depends on a particular subroutine of one task being exactly the solution to another.

One result which provides hope for partial function reuse is given by Yosinski et al.,[4] who showed that a deep network trained to recognize one domain of objects learned to recognize a new domain of objects far faster than an untrained, randomly-initialized network. So clearly there are domains with substantial overlap in their subroutines.

In order to encourage this behavior, it may be necessary to train the network on multiple domains simultaneously, thus letting the functions "coevolve" on all domains and encouraging separation of shared resources from those that are domain-specific.

Whether or not this happens in practice in an experimental question, but the proposed model provides a framework in which it is possible.

Experiments

Games

One compelling domain for multitask learning is simple arcade games such as Pac-Man or Breakout. Models such as the DQN[10] have been recently shown to perform very well on a subset of these games when trained for one game only, but so far efforts to play multiple games with one network have failed. Even minor perturbations in the visual representation of a game (e.g. changing the colors) require substantial retraining.

The author proposes to test the multitask learning abilities of this system by training it with interleaved examples of two similar games at a time. Pairs of games with different levels of similarity can be constructed by everything from creating minor visual differences in the same game to using different source games (e.g. Breakout versus Space Invaders).

Additionally, it would be interesting to see whether the system can efficiently perform curriculum learning. Curriculum learning is a subset of transfer learning which involves transferring skills learned on easy problems in a domain to harder problems in the same domain. Puzzle games with levels of increasing difficulty, such as Rush Hour, would be an interesting domain for this. This would be an extension to the project if time is available.

Language

A standard task for recurrent neural networks is that of next-word prediction, where the network is given some initialization text (perhaps character by character), then asked to predict the next word (or character) in the sequence one by one.

It would be extremely appealing to have a single system capable of doing this task simultaneously for two very different domains while sharing some of the computation. For example, the network might be trained on interleaved examples from the Bible and Twitter. Any sharing of learned functions might then reflect shared structure in the grammar or even thoughts represented by the words in those different domains.

This could even be taken to its logical extreme by training a single network to predict text in two different languages. This would be an extension to the project if time is available.

Discussion

This paper proposes a system capable of taking on challenges in the domain of transfer or multitask learning using a brain-inspired model of functional attention. This model consists of an LSTM controller which allots attention and a set of layers which learn functions. By separating the task of *deciding what to execute* from the task of *execution*, this model will prevent the problems of catastrophic forgetting and negative transfer while allowing the reuse of subcomputations which are shared between tasks.

---

1. Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. In IEEE Transactions on Knowledge and Data Engineering, 22(10), 1345–1359.
2. Glorot, X., Bordes, A., & Bengio, Y. (2011). Domain adaptation for large-scale sentiment classification: A deep learning approach. In Proceedings of the 28th International Conference on Machine Learning (ICML–11) (pp. 513–520).
3. Bengio, Y. (2012). Deep learning of representations for unsupervised and transfer learning. In JMLR: Workshop and Conference Proceedings 27 (pp. 17–37).
4. Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks?. In Advances in Neural Information Processing Systems (pp. 3320–3328).
5. Graves, A., Wayne, G., & Danihelka, I. (2014). Neural Turing Machines. arXiv preprint arXiv: 1410.5401.
6. Gregor, K., Danihelka, I., Graves, A., & Wierstra, D. (2015). DRAW: A recurrent neural network for image generation. arXiv preprint arXiv:1502.04623.

7. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735–1780.

8. Baldauf, D., & Desimone, R. (2014). Neural mechanisms of object-based attention. Science, 344(6182), 424–427.

9. Hastad, J. (1986, November). Almost optimal lower bounds for small depth circuits. In Proceedings of the eighteenth annual ACM symposium on Theory of computing (pp. 6–20). ACM.

10. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529–533.