{% feed_meta %}

Explanation of One-shot Learning with Memory-Augmented Neural Networks

Santoro et al. 2016.

I've found that the overwhelming majority of online information on artificial intelligence research falls into one of two categories: the first is aimed at explaining advances to lay audiences, and the second is aimed at explaining advances to other researchers. I haven't found a good resource for people with a technical background who are unfamiliar with the more advanced concepts and are looking for someone to fill them in. This is my attempt to bridge that gap, by providing approachable yet (relatively) detailed explanations. In this post, I explain the titular paper - One-shot Learning with Memory-Augmented Neural Networks.

In my last post, I criticized this paper as poorly motivated. After taking time to crystallize my thoughts and to email the authors, I'm less critical of it now, but I still have a few concerns. My goal is to try explaining the paper and my concerns in tandem.

Motivation

In an earlier post, I wrote about the need for massive amounts of data to train deep neural networks. In contrast, humans require comparatively little data to learn a new behavior or to rapidly shift away from an old behavior. For example, after running into a handful of street signs, the modern teenager quickly learns to be careful texting while walking. As Santoro et al. write, "This kind of flexible adaptation is a celebrated aspect of human learning (Jankowski et al., 2011), manifesting in settings ranging from motor control (Braun et al., 2009) to the acquisition of abstract concepts (Lake et al., 2015). Generating novel behavior based on inference from a few scraps of information – e.g., inferring the full range of applicability for a new word, heard in only one or two contexts – is something that has remained stubbornly beyond the reach of contemporary machine intelligence."

The term **one-shot learning** has been introduced to capture this phenomenon of rapid behavior change following a small number of experiences, or even just one experience. In an earlier paper, a neural network was given an external memory and the ability to learn how to use its new memory in solving specific tasks. This paper classifies that previous model, the Neural Turing Machine (NTM), as a subclass of the more general class of **Memory-Augmented Neural Networks** (MANNs), and suggests an alternative memory system capable of outperforming humans in certain one-shot learning tasks.

Background

If you haven't read the NTM paper or my walkthrough of it, set this aside and go read one (or both). In order to understand the proposed change to a NTM's memory, it'll be helpful to understand a NTM.

Intuition

The goal is to modify a NTM to excel at one-shot learning. To accomplish this, the authors modify the NTM's controller's memory access capabilities. However, the paper is rather terse in justifying their specific change. A NTM controller is capable of using content-based addressing, location-based addressing or both, so when Santoro et al. suggest using a pure content-based memory writer, I was confused why this would offer any improvement. The only explanation the paper offers is that "[The NTM's memory access method] was advantageous for sequence-based prediction tasks. However, this type of access is not optimal for tasks that emphasize a conjunctive coding of information independent of sequence."

To their credit, three of the four authors I emailed wrote back in less than a day with more detailed explanations. The answer is that there's a tradeoff when training memory-augmented networks; more sophisticated memory access capabilities are more powerful, but the controller requires more training. Santoro et al. propose hamstringing the controller's ability to write to memory using location-based addressing so that the controller will learn more quickly. They point out that location-based addressing won't be necessary to excel at one-shot learning. This is because for a given input, there are only two actions the controller might need to do and both depend on content-addressing. One action is that the input is very similar to a previously seen input; in this case, we might want to update whatever we wrote to memory. The other action is that the input isn't similar to a previously seen input; in this case, we don't want to overwrite recent information, so we'll instead write to the least used memory location.

The authors call this hamstrung memory system the **Least Recently Used Access (LRUA)** module. What confused me at first is that Santoro et al. describe the LRUA as "a pure content-based memory writer," and then in the same sentence, state that the LRUA will write "to either the least used memory location or the most recently used memory location," which sounds an awful lot like location-based addressing. I reconciled the discrepancy when I realized that the most recently used memory location is determined by which memory location was most recently read, and reading is determined using content similarity.

Mathematics

Reading

Just like the NTM, for a given input x_t and memory matrix M_t with R rows and C elements per row, the controller will read from memory a length-C vector r_t that is a linear combination of the memory's rows $M_t(i)$ scaled by a normalized read-weight vector w_t^r.

$$r_t \leftarrow \sum_i^R w_t^r(i)\mathcal{M}_t(i) \tag{4}$$

The read-weight vector is almost identical to the NTM's content weight vector. For a given input x_t, the controller will produce a key k_t. As with the NTM, this key will be compared against each row $M_t(i)$ in memory using the cosine similarity measure $K(\cdot,\cdot)$. A similarity measure is "a real-valued function that quantifies the similarity between two objects" (in this case two vectors). Cosine similarity is based on the angle between the two vectors, and is thus advantageous because it doesn't depend on the magnitude of the vectors (which may vary).

$$K(k_t, M_t(i)) = \frac{k_t \cdot M_t(i)}{\|k_t\| \cdot \|M_t(i)\|} \tag{2}$$

The similarity measure is used to produce the read-weight vector w_t^r that specifies how much each row should contribute to r_t. The only difference between the LRUA and the NTM is that there is no parameter β_t to control the read-weight vector's concentration; I do not know why Santoro et al. decided to remove this parameter.

$$w_t^r(i) = \frac{exp\Big(K(k_t, M_t(i))\Big)}{\sum_j exp\Big(K(k_t, M_t(j))\Big)} \tag{3}$$

Writing

To write to memory, the controller will interpolate between writing to the most recently read memory rows and writing to the least used memory rows. If w_{t-1}^r is the read-weight vector at the previous time step, and w_{t-1}^{lu} is a weight vector that captures the least used memory location (more on this in just a second), then the controller can choose a convex combination of the two weight vectors using a scalar parameter α and the sigmoid function $\sigma(\cdot)$ to create a write-weight vector w_t^w:

$$w_t^w \leftarrow \sigma(\alpha) w_{t-1}^r + (1 - \sigma(\alpha)) w_{t-1}^{lu} \tag{7}$$

Each row in memory is then updated using the write-weight vector and the key k_t issued by the controller.

$$M_t(i) \leftarrow M_{t-1}(i) + w_t^w(i) k_t \tag{8}$$

To create the least used weight vector w_{t-1}^{lu}, the controller maintains another weight vector, the usage-weight vector w_t^u, that is updated after every read and write step. A scalar parameter, γ, is used to determine how quickly previous usage values should decay:

$$w_t^u \leftarrow \gamma w_{t-1}^u + w_t^r + w_t^w \tag{5}$$

The least used weight vector w_{t-1}^{lu} is generated from the usage weight vector w_{t-1}^u by setting the index of the usage weight vector's minimum element in the least used weight vector to 1, and all other elements in the least used weight vector to 0. In other words, the least used weight vector is really just an indicator vector for the smallest element in the usage weight vector. For concreteness, if the usage weight vector is `[0.7, 0.25, 0.05]`, the least used weight vector will be `[0, 0, 1]`.

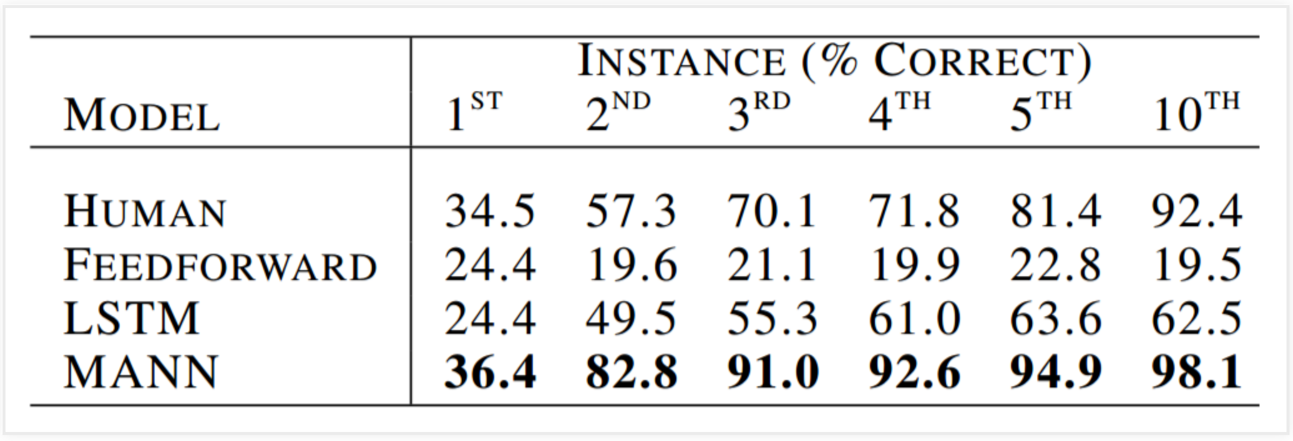
Experiments and Results

Omniglot Classification

The first test for the LRUA MANN was a classification test using the Omniglot dataset. If you're not familiar, it's a collection of symbols from a number of different written languages, with over 1600 separate classes and only a few examples per class, making it perfect for one shot learning. To reduce the risk of overfitting, the authors perturbed the symbol images with small, random translations and rotations. They also created new classes by rotating the initial classes' images by 90, 180 and 270 degrees.

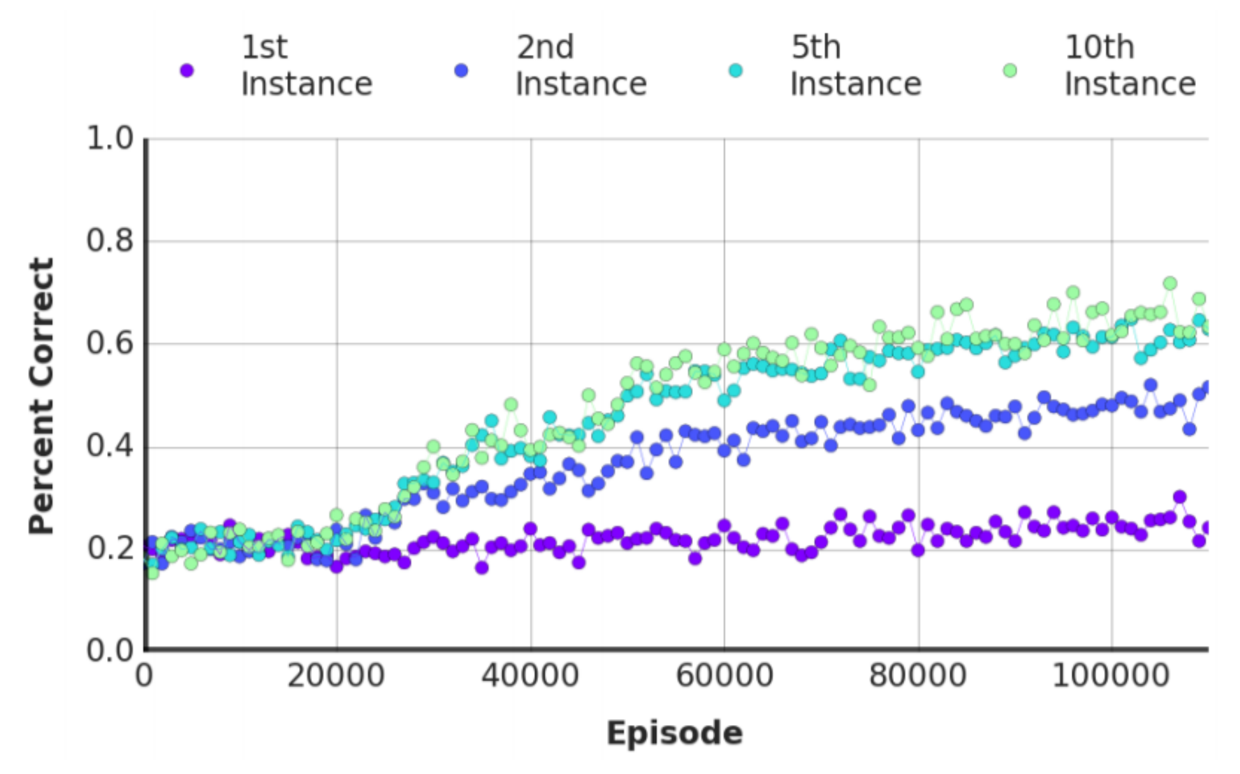
The first test involved randomly selecting five classes and then randomly assigning each class a label 1 through 5 (in practice, the class labels were one-hot vectors). The objective was to infer the label that corresponds to each class. For concreteness, the five classes could be the first five letters of the English alphabet `['a', 'b', 'c', 'd', 'e']` and the corresponding labels are `[2, 5, 4, 1, 3]`; each model would be shown an instance of one class, submit a guess as to what the correct label is, and then it would be informed what the true label is. This would be repeated until the model had seen 10 instances of each of the five classes.

A LRUA MANN, a LSTM and a vanilla feedforward neural network were all trained at this task. To establish a baseline, nine humans each performed the task twice. As the table below shows, the LRUA MANN outperformed its three competitors. It managed to correctly label over a third of class instances it had never seen before. The LRUA MANN learned that if a key was a poor match to keys it had seen previously, then it should try guessing a class label it hadn't already seen before e.g. if the controller sees 'd' and already knows that 'a' is 2, 'b' is 5 and 'c' is 4 and can recognize 'd' as distinct from 'a', 'b', and 'c', it should guess a new label like 1 or 3.

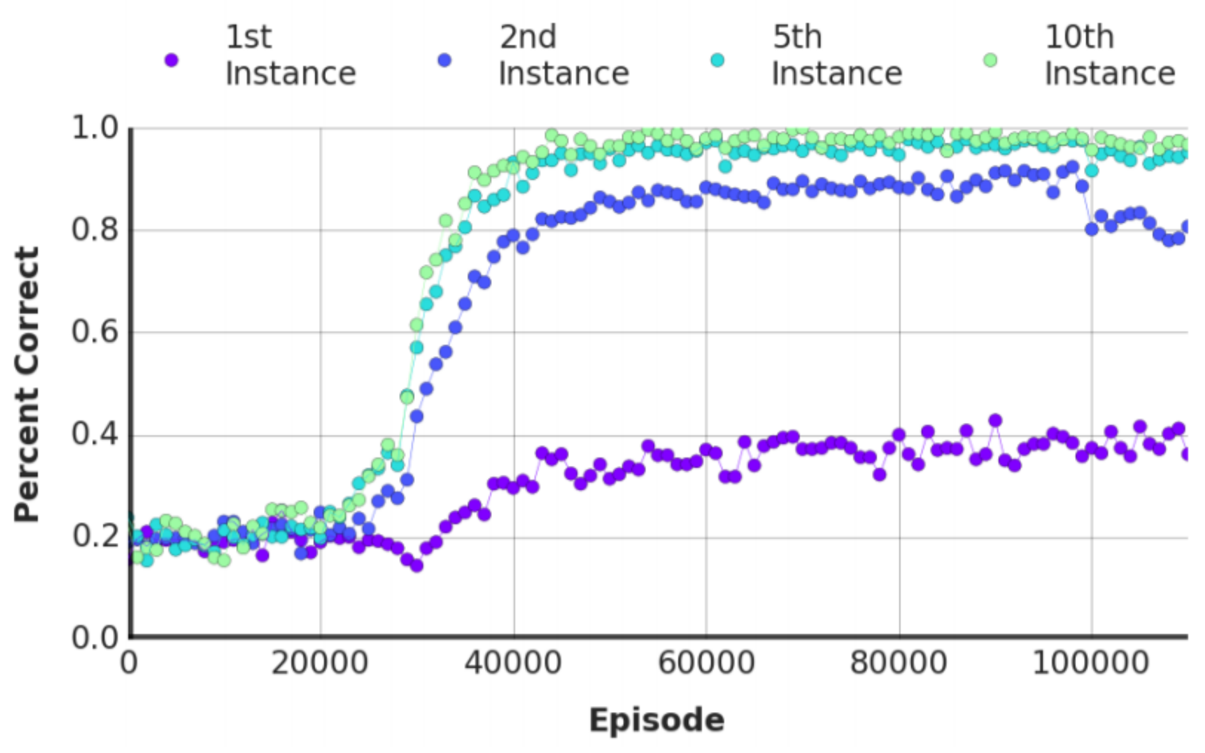


	Instance (% Correct)					
Model	1st	2nd	3rd	4th	5th	10th
Human	34.5	57.3	70.1	71.8	81.4	92.4
Feedforward	24.4	19.6	21.1	19.9	22.8	19.5
LSTM	24.4	49.5	55.3	61.0	63.6	62.5
MANN	**36.4**	**82.8**	**91.0**	**92.6**	**94.9**	**98.1**

Below are two plots comparing the LSTM's performance against the LRUA MANN's performance (the paper doesn't explicitly state this, but the LRUA MANN used a LSTM controller) as functions of training episodes. Note that the MANN quickly learns to outperform the LSTM.

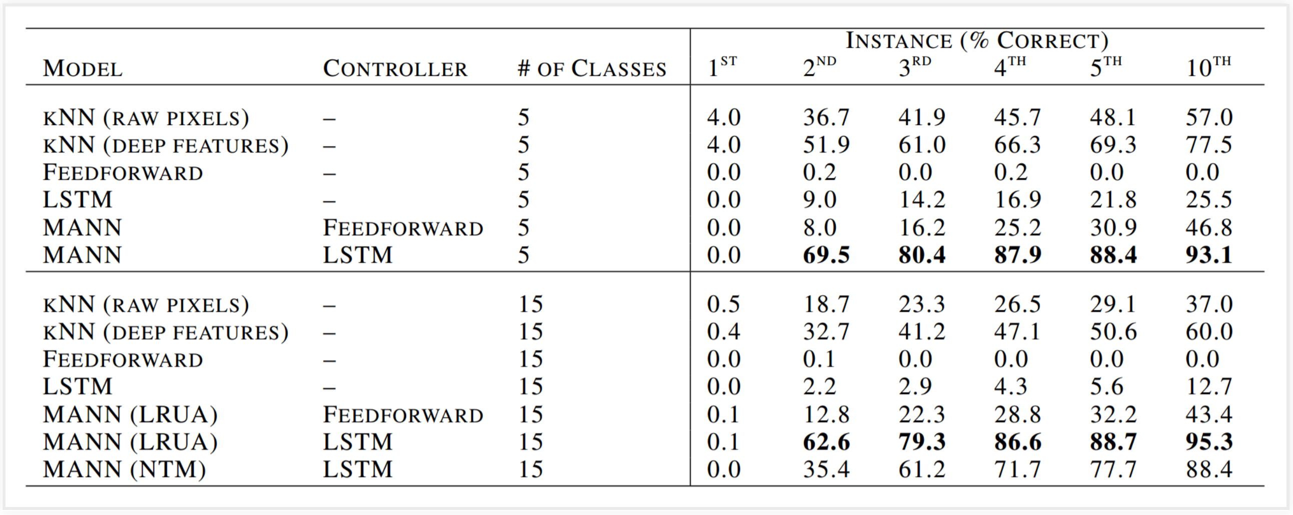


(a) LSTM, five random classes/episode, one-hot vector labels



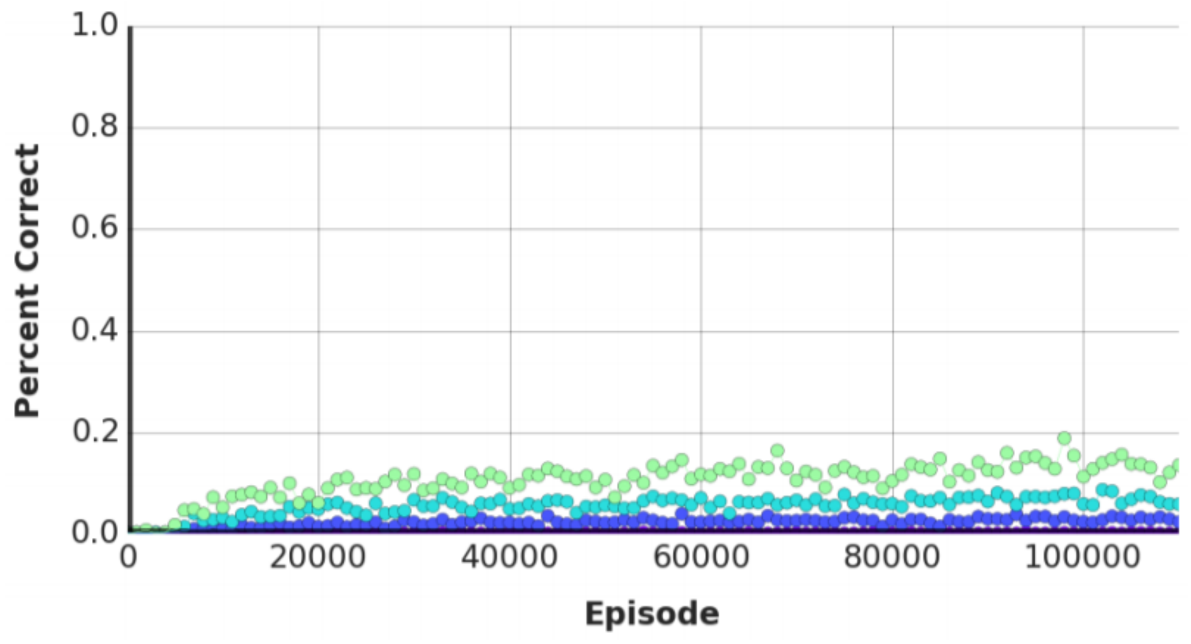
(b) MANN, five random classes/episode, one-hot vector labels

The same experiment was repeated, but with fifteen classes instead of five classes. Humans weren't included as a baseline because human working memory struggles to hold more than seven items and as the authors put it, "[trying to learn fifeen or more classes] proved much too difficult and highly demotivating." Instead, two additional models were considered. The first was a k-nearest-neighbors non-parametric model that used the raw input pixels. The second was a k-nearest-neighbors non-parametric model that instead used features extracted by an autoencoder; the autoencoder's encoder and decoder each had two 200-unit layers and a bottleneck layer of 32 units.

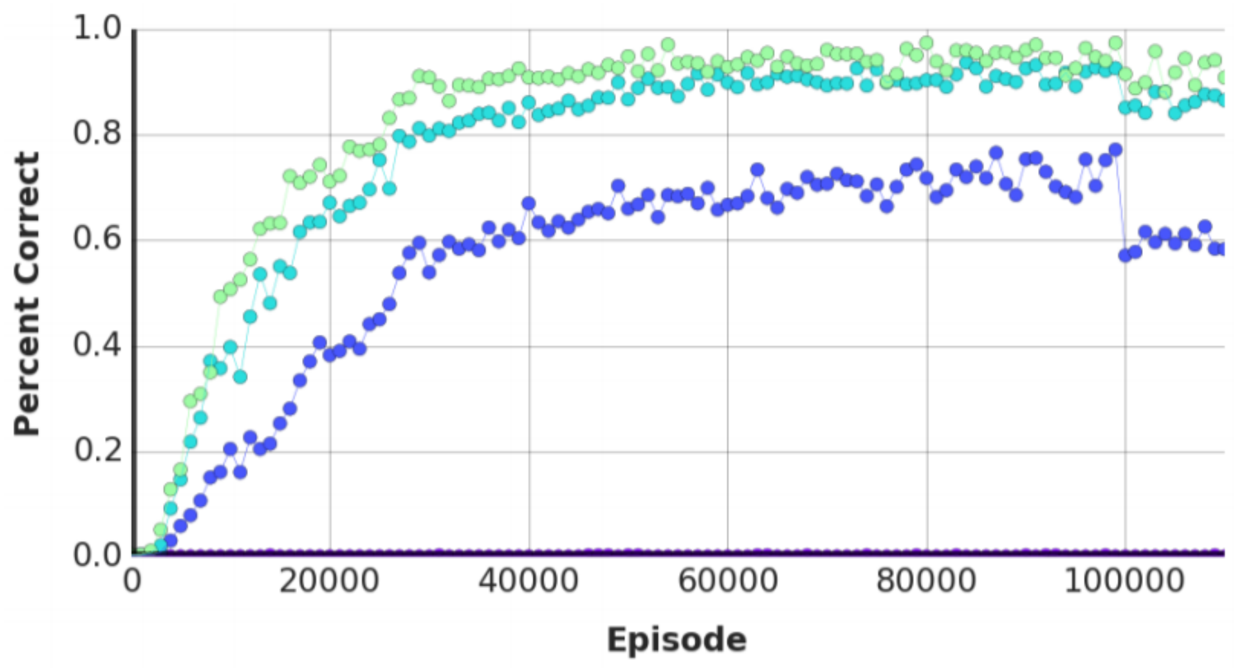


			Instance (% Correct)					
Model	Controller	# of Classes	1st	2nd	3rd	4th	5th	10th
kNN (raw pixels)	–	5	4.0	36.7	41.9	45.7	48.1	57.0
kNN (deep features)	–	5	4.0	51.9	61.0	66.3	69.3	77.5
Feedforward	–	5	0.0	0.2	0.0	0.2	0.0	0.0
LSTM	–	5	0.0	9.0	14.2	16.9	21.8	25.5
MANN	Feedforward	5	0.0	8.0	16.2	25.2	30.9	46.8
MANN	LSTM	5	0.0	**69.5**	**80.4**	**87.9**	**88.4**	**93.1**
kNN (raw pixels)	–	15	0.5	18.7	23.3	26.5	29.1	37.0
kNN (deep features)	–	15	0.4	32.7	41.2	47.1	50.6	60.0
Feedforward	–	15	0.0	0.1	0.0	0.0	0.0	0.0
LSTM	–	15	0.0	2.2	2.9	4.3	5.6	12.7
MANN (LRUA)	Feedforward	15	0.1	12.8	22.3	28.8	32.2	43.4
MANN (LRUA)	LSTM	15	0.1	**62.6**	**79.3**	**86.6**	**88.7**	**95.3**
MANN (NTM)	LSTM	15	0.0	35.4	61.2	71.7	77.7	88.4

As before, the LRUA MANN with a LSTM controller outperformed its competitors. Interestingly, the NTM almost performed as well as the LRUA MANN, confirming the early hypothesis that by reducing the ability of the controller to write to memory, the LRUA MANN will learn faster. The LRUA MANN also continued to outperform the LSTM. both in the limit and with fewer training episodes



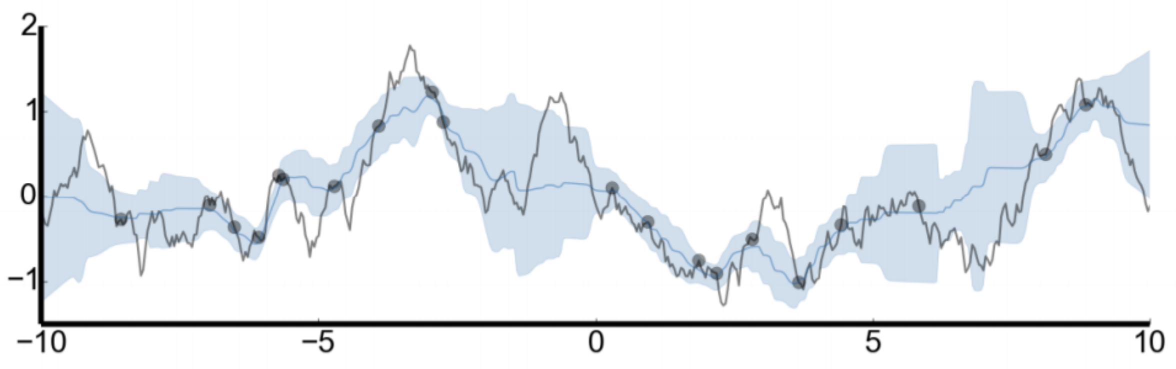
(c) LSTM, fifteen classes/episode, five-character string labels



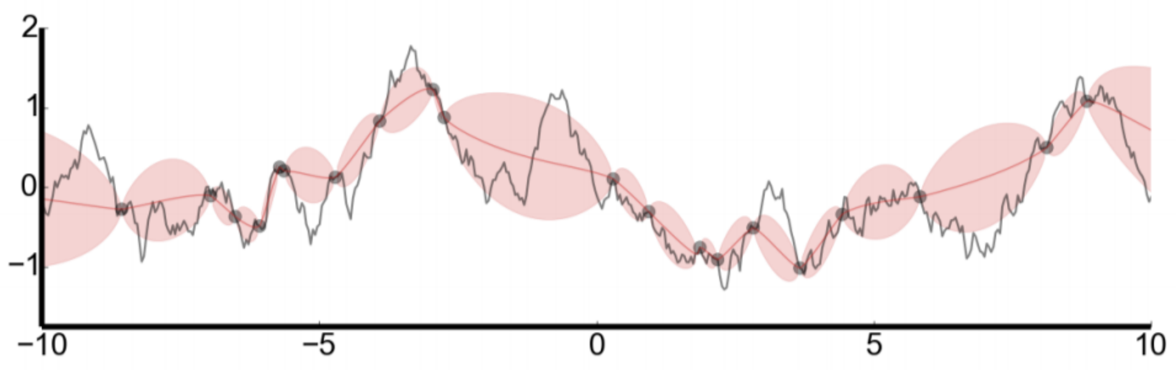
(d) MANN, fifteen classes/episode, five-character string labels

Gaussian Process

For the next test, the authors shifted from classification to regression. They sampled functions from a Gaussian Process (GP) with fixed hyperparameters, and then tested whether the LRUA MANN could learn to predict function values by interpolating from previously seen points. Each training episode consisted of the x-value x_t and the time-offset function value i.e. $f(x_{t-1})$. As a baseline, the LRUA was compared with a Gaussian process given access to the same hyper-parameters used to generate the function. The figure below shows how closely matched the two models' predictions are.

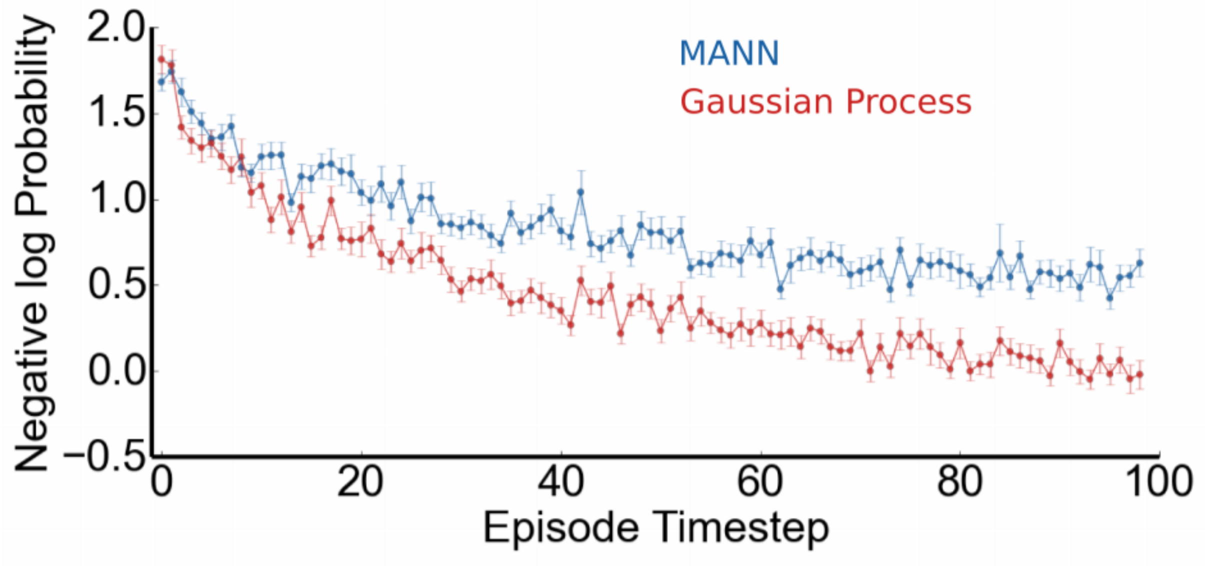


(a) MANN predictions along all x-inputs after 20 samples

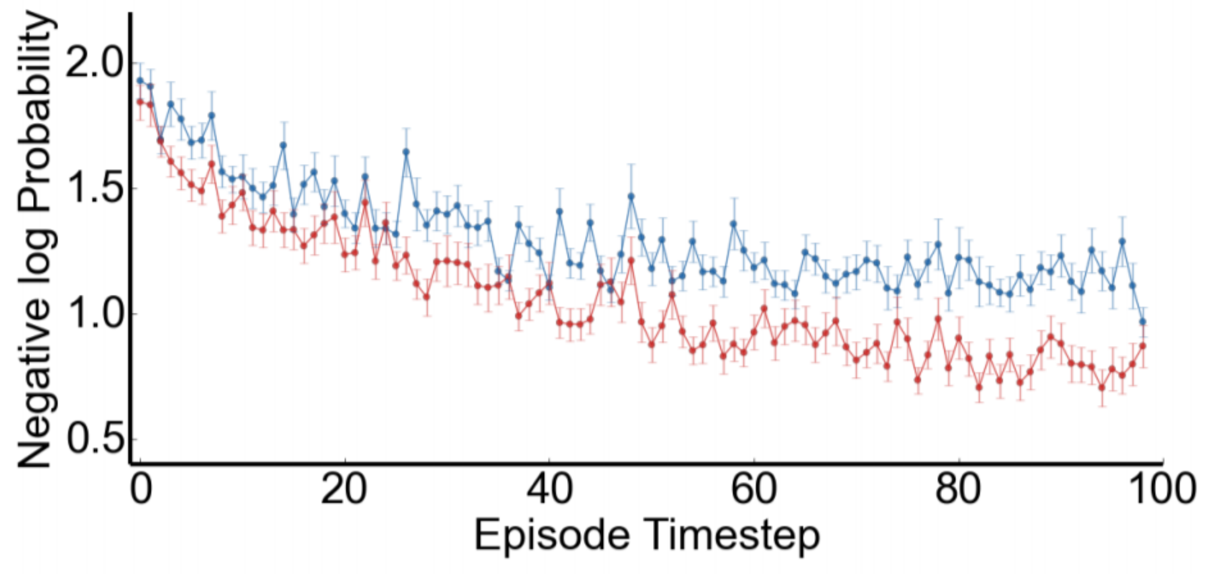


(b) GP predictions along all x-inputs after 20 samples

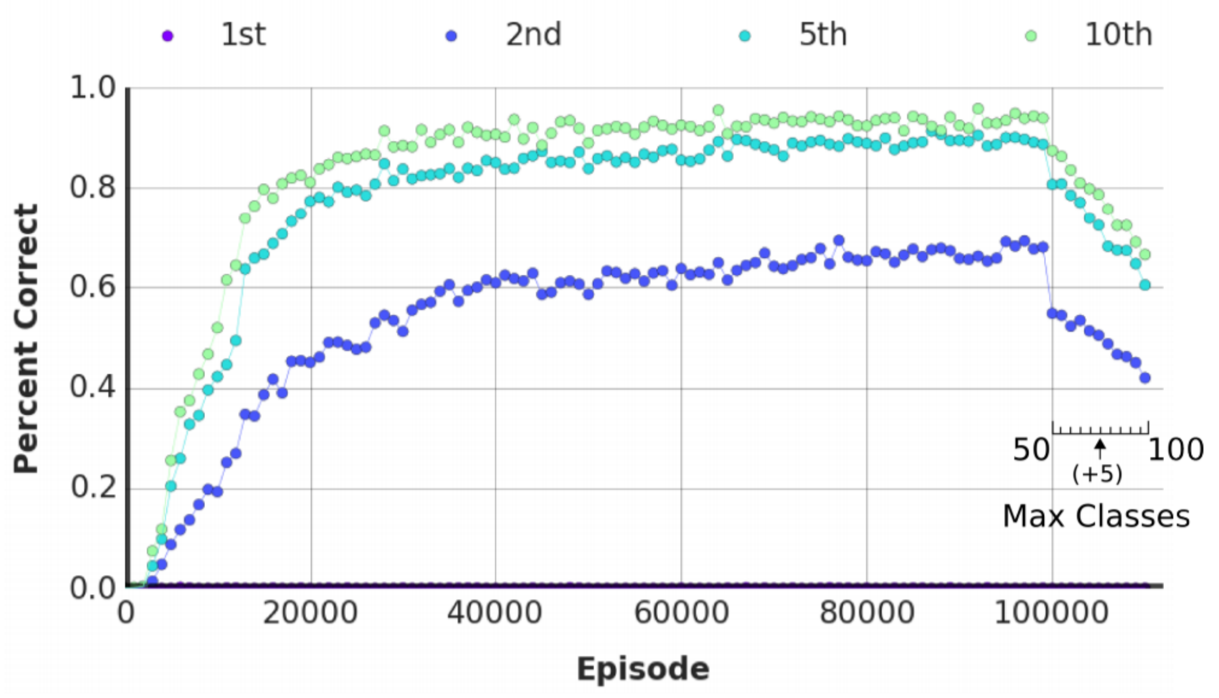
The LRUA MANN also performed well against the GP in two and three dimensions.



(a) 2D regression log-likelihoods within an episode



(b) 3D regression log-likelihoods within an episode



(a) One additional class per 10,000 episodes

Discussion

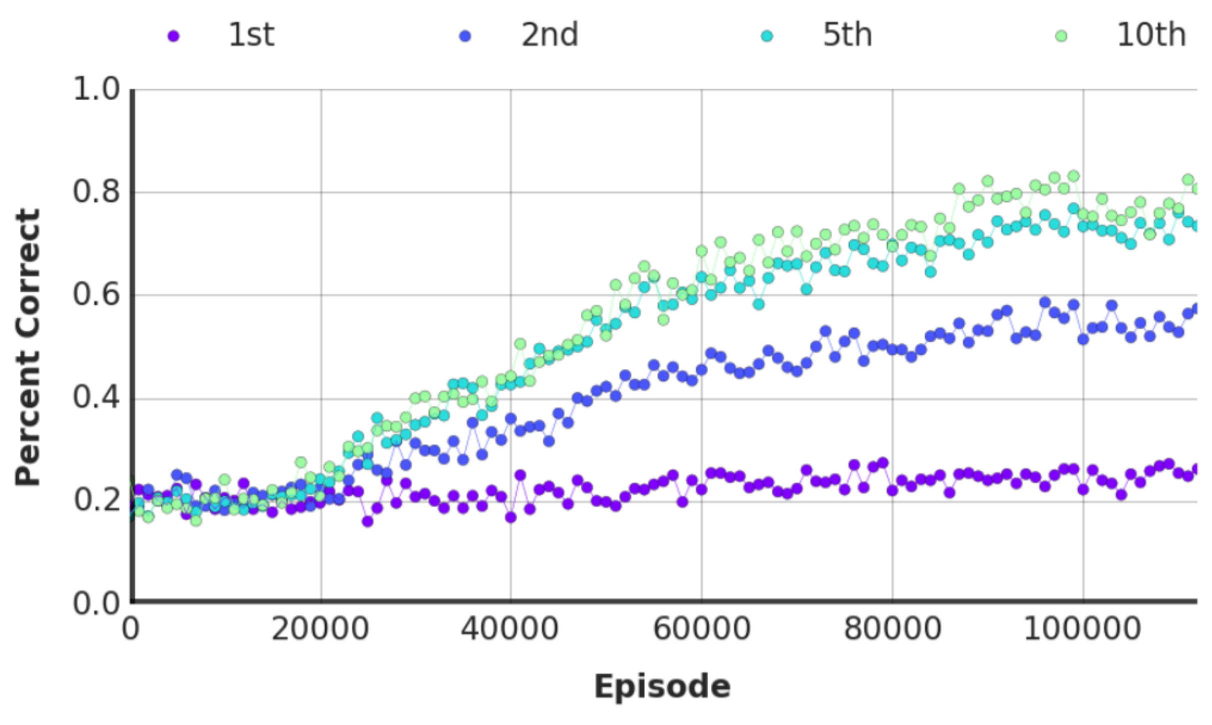
Is a MANN a meta-learner?

I think a large part of the cognitive dissonnance I experienced reading this paper was due to the introduction, which frames the MANN as a meta-learner. **Meta-learning** is the idea of a learning system with two collaborative learning processes that learn across different time scales: in the short term, one learning system quickly learns how to perform well at an immediate task, and in the long term, the other learning system slowly learns common structure across tasks.

The authors argue that meta-learning is a possible way to achieve one-shot learning, and then argue that memory-augmented neural networks are excellent candidates for meta-learning: "This ability [of rapid encoding, storage and retrieval] makes the NTM a perfect candidate for meta-learning and low-shot prediction, as it is capable of both long-term storage via slow updates of its weights, and short-term storage via its external memory module." However, when I think of "learning" in the context of neural networks or machine learning, I don't think of data storage; I think of updating model parameter values in accordance with newly observed data. I'm forced to ask whether storing data in external memory represents a form of short term learning, and I'm not at all confident that it does.

Memory Wiping

The earlier comparison of the LRUA MANN with the human test subjects wasn't entirely fair. In order to surpass human performance, the LRUA MANN's memory had to be wiped between episodes. Without that added code, the LRUA MANN was much less successful at beating the humans as the image below shows. The authors say that the LRUA MANN was still successful under certain setups (they specifically mention ten classes with episodes of length 75), but they don't provide the percentage of setups in which the LRUA MANN outperforms humans without the memory wipe.



(a) Five classes per episode

Future Directions

One of the coolest parts of writing this paper's authors is that they were willing to tell me a little about future research directions. Both Santoro and Lillicrap suggested reading their recent paper, and Santoro also suggested Google Brain's recent paper as well as FAIR's slightly older paper. I'm sharing these in case readers want to continue to learn more about the topic, as this might be my last post on MANNs in the short term.

Summary

- A key feature of human intelligence, one-shot learning, refers to the ability of humans to substantially shift behavior following as little as a single experience

- Deep learning has historically struggled to achieve one-shot learning
- Memory-augmented neural networks can display one-shot learning capabilities on par with humans in specific tasks
- More advanced memory-access capabilities appear to require more training

Notes

I appreciate any and all feedback. If I've made an error or if you have a suggestion, you can email me or comment on the Reddit or HackerNews threads. I still haven't created a mailing list or integrated RSS, but I promise I'll get to it after submitting my MRes applications.