# One-shot Learning with Memory-Augmented Neural Networks (Deepmind)

* Abstract: - when new data is encountered, the models must inefficiently relearn their parameters to adequately incorporate the new information without catastrophic interference.
* Architectures with augmented memory capacities, such as Neural Turning Machine (NTMs), offer the ability to quickly encode and retrieve new information, and hence can potentially obviate the downsides of conventional models.
* In this paper the researchers demonstrate the ability of a memory-augmented neural network to rapidly assimilate new data, and leverage this data to make accurate predictions after only a few samples.
* The researchers also present a new method this method use memory location based focusing mechanisms.
* Introduction: - Many problems of interest require rapid inference from small quantities of data. In the limit of “one-shot learning”, single observations should result in abrupt shifts in behavior.
* 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.
* Previous work does suggest one potential strategy for attaining rapid learning from sparse data, and hinges on the notion of *meta-learning*.
* Meta-learning generally refers to a scenario in which an agent learns at two levels, each associated with different time scales.
* Rapid learning occurs *within* a task, for example, when learning to accurately classify within a particular dataset. This learning is guided by knowledge accrued more gradually *across* tasks, which captures the way in which task structure varies across target domains.
* Neural networks with a memory capacity provide a promising approach to meta-learning in deep networks.
* A scalable solution has few necessary requirements: first, information must be stored in memory in a representation that it both stable and element-wise addressable.
* Second, the number of parameters should not be tried to the size of the memory.
* Recent architectures like Neural Turing Machines and memory networks, meet the requisite criteria.
* The researchers revisit the meta-learning problem and setup from the perspective of a highly capable memory-augmented neural network (MANN).
* The researchers demonstrate that MANNs are capable of meta-learning in tasks that carry significant short- and long-term memory demands.
* It manifests as successful classification as never-seen-before with small amount of data. The researchers also outline a memory access module that emphasizes memory access by content, and not additionally on memory location, as in original implementations of the NTM.
* The researchers’ approach combines the best of two worlds: - the ability to slowly learn an abstract method for obtaining useful representation of raw data, via gradient descent, and the ability to rapidly bind never-before-seen information after a single presentation, via an external memory module.
* Meta-Learning Task Methodology: - the researchers to choose parameters to minimize a learning cost across some dataset *D*. For meta learning they choose parameters to reduce the expected learning cost across a distribution of datasets *p*(*D*):



* In the setup, a task, or episode, involves the presentation of some dataset *D* = =.
* The classification, is the class label for an image and for regression, is the value of a hidden function for a vector with real-valued elements or simply a real-valued number .
* In the setup, is both a target, and is presented as input along with , in a temporally offset manner; i.e., the network sees the input sequence (), (), … , ().
* And so, at time *t* the correct label for the previous data sample () is provided as input along with a new query .
* The network is tasked to output the appropriate label for at the given timestep. The labels are shuffled from dataset-to-dataset.
* For a given episode, ideal performance involves a random guess for the first presentation of a class, and the use of memory to achieve perfect accuracy thereafter.
* Ultimately, the system aims at modelling the predictive distribution , inducing a corresponding loss at each time step.
* This task structure incorporates exploitable meta-knowledge: a model that meta-learns would learn to bind data representations to their appropriate labels regardless of the actual content of the data representation or label, and would employ a general scheme to map these bounds representation to appropriate classes or function values for prediction.
* Memory-Augmented Model: - Neural Turning Machines (NTM)– the Neural Turning Machine is a fully differentiable implementation of a MANN.
* It has a controller, like feed-forward network or LSTM, it interacts with an external memory module using a number of read and write heads.
* The memory encoding and retrieval in a NTM external memory is rapid, with vector representations being placed into or taken out of memory potentially every time-step.
* It makes NTM a perfect candidate for meta-learning and low-shot prediction, it is capable of both long-term storage via slow updates of its weights, and short-term storage via slow updates of its weights, and short-term storage via its external memory module.
* The controllers used in the model are either LSTMs, or feed-forward networks.
* Given some input, , the controller produces a key, , which is then either stored in a row of a memory matrix , or used to retrieve a particular memory, *i*, from a row; i.e., .
* When retrieving a memory, is addressed using the cosine similarity measure,



* It is used to produce a read-weight vector, with elements computed according to a softmax:



* A memory, is retrieved using this weight vector:



* This memory is used by the controlled as the input to a classifier, like output layer, and as an additional input for the next controller state.
* Least Recently Used Access: - Location-based addressing was used to promote iterative steps, akin to running along a tape, as well as long-distance jumps across memory.
* It is good for sequence-based prediction tasks, but not good for tasks that emphasize a conjunctive coding of information independent of sequence.
* The researchers used a newly designed access module called the Least Recently Used Access (LRUA) module.
* The LRUA module is a pure content-based memory writer that writes memories to either that least used memory location or the most recently used memory location.
* New information is written into rarely-used locations, preserving recently encoded information, or it is written to the last used location, which can function as an update of the memory with newer, possibly more relevant information.
* The differences between these two options is accomplished with an interpolation between the previous read weights and weights scaled according to usage weights .



* Here, is a decay parameter and is computed as in eq. (3). The *least-used* weights, , for a given time-step can then be computed using .
* First the researchers introduce the notation to denote the *nth* smallest element of the vector v. Elements of are set accordingly:



* In the above equation *n* is set to equal the number of reads to memory. To get the write weights , a learnable sigmoid gate parameter is used to compute a convex combination of the previous read weights and previous least-used weights:



* In the above equation, is a sigmoid function, and is a scalar gate parameter to interpolate between the weights.
* The least location is computed from and is set to zero.



* Experimental Results: - Data – two sources of data were used: Omniglot, for classification, and sampled functions from a Gaussian process (GP) with fixed hyperparameters, for regression.
* The Omniglot dataset consists of over 1600 separate classes with few examples per class.
* To reduce the risk of overfitting, the researchers performed data augmentation by randomly translating and rotating character images.
* The training of all models was performed on the data of 1200 original classes with the rest of the 423 classes being used for test experiments.
* Omniglot Classification: - the MANN was trained using one-hot vector representations as class labels. After training on 100,000 episodes wit five randomly chosen classes with randomly chosen labels, the network was given a series of test episodes.
* For classification using one-hot vector representations, one relevant baseline is human performance.
* The MANN displayed better than random guessing on the first instance within a class.
* It employed a strategy of educated guessing; if a particular sample produced a key that was a poor match to any of the bindings stored in external memory, then the network was less likely to choose the class labels associated with these stored bindings, and hence increased its probability of correctly guessing this new class on the first instance.
* For large classifier a different approach for labeling is used, in this new label technique new labels consisted of strings of five characters, with each character assuming one of five possible values.
* These characters are {‘a’, ’b’, ’c’, ’d’, ’e’}, it produces random strings such as ‘abcde’.
* Strings were represented as concatenated one-hot vectors, and hence were of length 25 with five elements assuming a value of 1, and the rest 0.
* To confirm that the network was able to learn using these class representations, the previously described experiment was repeated.
* A MANN with a standard NTM access module was unable to reach comparable performance to a MANN with LRU access.
* The researchers considered a set of baselines, such as a feed-forward RNN, LSTM, and a nonparametric nearest neighbors’ classifier that used either raw-pixel input or features extracted by an autoencoder.
* The autoencoder consisted of an encoder and decoder each with two 200-unit layers with leaky ReLU activations, and an output bottleneck layer of 32 units.
* The resultant architecture contained significantly more parameters than the MANN and, additionally, was allowed to train on three times as much augmented data.
* The highest accuracies from researchers’ experiments are reported it achieved this using a single nearest neighbor for prediction and features from the output bottleneck layer of the autoencoder.
* This kNN has unlimited amount of memory which give it an advantage to store previous examples also.
* It gives the kNN with a distinct advantage, even when raw pixels were used as input representations.
* The kNN baseline was clearly outperformed by the MANN.
* Persistent Memory Interference: - a good strategy to employ in this classification task, and the strategy that was artificially imposed thus-far, is to wipe the external memory from episode to episode.
* To check the effects of memory interference, the researchers performed the classification task without wiping the external memory between episodes.
* It proved predictably difficult, and the network was less robust in its ability to achieve accurate classification.
* In the case of learning one-hot vector labels in an episode that contained five unique classes, learning progressed much slower than in the memory-wipe condition, and did not produce the characteristic fast spike in accuracy seen in the memory-wipe condition.
* Curriculum Training: - the successful one-shot classification in episodes with fifteen classes, they employed a curriculum training regime to further scale the classification capabilities of the model.
* The network was first tasked to classify fifteen classes per episode, any every 10,000 episodes of training thereafter, the maximum number of classes presented per episode incremented by one.
* After training, at the 100,000 episodes mark, the network was tested on episodes with 50 classes.
* Similar tests continued, increasing the maximum number of classes to 100.
* The network generally exhibited gradually decaying performance as the number of classes increased towards 100.
* Regression: - the MANN architecture generated a broad strategy for meta-learning, we reasoned that it would be able to adequately perform regression tasks on never-before-seen functions.
* For testing, they generate functions using a GP prior with a fixed set of hyper-parameters and trained our network using unique functions in each episode.
* Each episode involved the presentation of *x*-values along with time-offset function values.
* A successful strategy involves the binding of *x*-values with the appropriate function values and storage of these bindings in the external memory.
* This task demands a broader read memory: the network must learn to interpolate from previously seen points, which most likely involves a strategy to have a more blended read-out from memory.
* Network performance was compared to true GP predictions of samples presented in the same order as was seen by the network.
* A GP is able to perform complex queries over all data points in one step. On the other hand, a MANN can only make local updates to its memory, and hence can only approximate such functionality.
* In the research the GP was initiated with the correct hyper-parameters for the sampled function, which give it an advantage in function prediction.
* The MANN predictions track the underlying function, with its output variance increasing as it predicts function values that are distance from the values it has already received.
* These results were extended to 2-D and 3-D cases, with the GP again having access to the correct hyper-parameters for the sampled functions.
* In both cases the log-likelihood predictions, of the MANN tracks appreciably well versus the GP, with prediction becoming more accurate as sampled are stored in the memory.
* Discussion & Future Work: - many important learning problems demand an ability to draw valid inferences from small amounts of data, rapidly and knowledgeably adjusting to new information.
* In the researcher’s technique they gradual, incremental learning encodes background knowledge that spans tasks, while a more flexible memory resource binds information particular to newly encountered tasks.
* These are deep-learning architectures containing a dedicated, addressable memory resource that is structurally independent from the mechanism that implement process control.
* The MANN examined here was found to display performance superior to a LSTM in two meta-learning tasks, performing well in classification and regression tasks when only sparse training data was available.
* A critical aspect of the tasks studied is that they cannot be performed based solely on rote memory.
* The new information must be flexibly stored and accessed, with correct performance demanding more than just accurate retrieval.
* MANNs are well-suited to meet these dual challenges, given their combination of flexible memory storage with the rich capacity of deep architectures for representation learning.



* Meta-learning is recognized as a core ingredient of human intelligence, and an essential test domain for evaluating models of human cognition.
* In informal comparisons against human subjects, the MANN employed in this paper displayed superior performance, even at set-sizes that would not be expected to overtax human working memory capacity.
* However, when memory is not cleared between tasks, the MANN suffers from proactive interference, as seen in many studies of human memory and inference.
* The preliminary observations suggest that MANNs may provide a useful heuristic model for further investigation into the computational basis of human meta-learning.
* The challenges need to solve in next-stages are - first researchers’ experiments employed a new procedure for writing to memory that was *prima facie* well suited to the tasks studied.
* Second, although they tested MANNs in settings where task parameters changed across episodes, the tasks studied contained a high degree of shared high-level structure.
* Finally, it may be of interest to examine MANN performance in meta-learning tasks requiring active learning, where observations must be actively selected.
* Supplementary information: - additional model details – the researchers’ model is a variant of a Neural Turing Machine (NTM) from Graves et al. It consists of a number of differentiable components: a controller, read and write heads, an external memory, and an output distribution.
* The controller receives input data directly, and also provides as input to the output distribution.



* The controller in researchers’ experiments is feed-forward networks or Long Short-Term Memories (LSTMs).
* For the best performing networks, the controller is a LSTM with 200 hidden units. The controller receives some concatenated input (,) and updates its state according to:



* Here , , are the forget gates, output gates, and input gates, respectively, are the hidden state biases, is the cell state, is the hidden state, is the vector read from memory, is the concatenated output of the controller, represents element-wise multiplication, and (., .) represents vector concatenation.
* are the weights from the input (,) to the hidden state, and are the weights between hidden states connected through time.
* The read vector is computed using content-based addressing using a cosine distance measure.
* The network has an external memory module, , that is both read from and written to. For reading, the controller cell state serves as a query for .
* First, a cosine distance measure is computed for the query key vector and each individual row in memory:



* After, that these similarity measures are used to produce a read-weight vector , with elements computed according to a softmax:



* A memory, is then retrieved using these read-weights:



* Finally, is concatenated with the controller hidden state, , to produce the networks’ output . Four reads were chosen for the reported experimental results.
* To write to memory, they implemented a new content-based access module called Least Recently Used Access (LRUA).
* LRUA writes to either the most recently read location, or the least recently used location, so as to preserve recent, and hence potentially useful memories, or to update recently encoded information.
* Usage weights are computed each time-step to keep track of the location most recently read or written to:



* Here is a decay parameter. The *least-used* weights, , for a given time-step can then be computed using .
* First, the researchers introduce the notation to demote the smallest element of the vector v. Elements of are set accordingly:



* Here is set to equal the number of reads to memory.
* The write weights , a learnable sigmoid gate parameter is used to compute a convex combination of the previous read weights and previous least-used weights:



* Here is a dynamic scalar gate parameter to interpolate between the weights. The least used memory location is computed from and is set to zero.
* Writing to memory then occurs in accordance with the computed vector of write weights:



* Output distribution: - the controller’s output,, is propagated to an output distribution. In classification the task used one-hot labels, the controller output is first passed through a linear layer with an output size equal to the number of classes to be classified per episode.
* For one-hot classification, the output distribution is a categorical distribution, implemented as a softmax function.
* The categorical distribution produces a vector of class probabilities, , with elements:



* Here are the weights from the controlled output to the linear layer output.
* The linear output size is kept at 25, it allows for the output to be split into five equal parts of size five.
* Each of these categorical distributions independently predicts a ‘letter’, and these letters are then concatenated to produce the five-character-long string label that serves as the network’s class prediction.
* Similar implementation is used for regression tasks. The linear output from the controller outputs two values: and , which are passed to a Gaussian distribution sampler as predicted mean and variance values.
* After that the Gaussian sampling distribution then computes probabilities for the target value using these values.
* Learning: - for the one-hot label classification, given the probabilities output by the network, , the network minimizes the episode loss of the input sequence:



* Here , is the target one-hot or string label at time *t*. For string label classification, the loss is similar:



* Here, the (*c*) indexes a five-element long ‘chunk’ of vector label, of which there are a total of five.
* For regression, the networks’ output distribution is a Gaussian, and as such receives two-values from the controller output’s linear layer at each time-step: predictive and values, which parameterize the output distribution.
* The network minimizes the negative log-probabilities as determined by the Gaussian output distribution given these parameters and the true target .
* Classification Input data: - input sequences consist of flattened, pixel-level representations of images and time-offset labels .
* First *N* unique classes are sampled from the Omniglot dataset, where *N* is the maximum number of unique classes per episode. *N* have values of 5, 10, or 15.
* Samples from the Omniglot source set are pulled, and are kept if they are members of the set of *n* unique classes for that given episode, and discarded otherwise.
* 10*N* samples are kept, in this setup, the number of samples per unique class are not necessarily equal, and some classes may not have any representative samples.
* The image is flattened into a vector, concatenated with a randomly chosen, episode-specific label, and fed as input to the network controller.
* Class labels are randomly chosen for each class from episode-to-episode. For one-hot label experiments, labels are of size *N*, where *N* is the maximum number of unique classes that can appear in a given episode.
* Task: - either 5, 10, or 15 unique classes are chosen per episode. Episode lengths are ten times the number of unique classes, unless explicitly mentioned.
* Training was done for 100,000 episodes. The data are pulled from a disjoint test set, and weight updates are ceased. This is deemed the “test phase”.
* For curriculum training, the maximum number of unique classes per episode increments by 1 every 10,000 training episodes. The episode length increases to 10 times this new maximum.
* Parameters: - optimization – Rmsprop was used with a learning rate of and max learning rate of , decay of 0.95 and momentum 0.9.
* Free parameter grid search – a grid search was performed over number of parameters, with the values used shown in parentheses: memory slots (128), memory size (40), controller size (200 hidden units for a LSTM), learning rate (), and number of reads from memory (4). Other free parameters were left constant: usage decay of the write weights (0.99) minibatch size (16).
* Comparisons and controls evaluation metrics – Human comparison – the participants perform the exact same experiment as the network: they observe sequences of images and time-offset labels, and are challenged to predict the class identify for the current input image by inputting a single digit on a keypad.
* They see the class labels in integers from 1 to 5, rather than one-hot vector or strings.
* Participants are made aware of the goals of the task prior to starting, and they perform a single, non-scored trial run prior to their scored trials. Nine participants each performed two scored trials.
* kNN – when no data is available, the kNN classifier randomly return a single class as its prediction.
* For the first data point, the probability that the prediction is correct is where *N* is number of unique classes in a given episode.
* After that, it predicts a class from classes that it has observed. Therefore, all instances of samples that are not members of the first observed class cannot be correctly classified until at least one instance is passed to the classifier.
* Since statistics are averaged across classes, first instance accuracy becomes = , which is 4% and 0.4% for 5 and 15 classes per episode, respectively.