

Meta-Learning via Feature-Label Memory Network

Dawit Mureja and Hyunsin Park and Chang D. Yoo

Korea Advanced Institute of Science and Technology

School of Electrical Engineering

{dawitmureja, hs.park, cd_yoo}@kaist.ac.kr

Abstract

Deep learning typically requires training a very capable architecture using a large dataset. However, many important learning problems demand an ability to draw valid inferences from a small size dataset, and such problems pose a particular challenge for deep learning. In this regard, various researches on “meta-learning” are being actively conducted. Recent work has suggested a Memory Augmented Neural Network (MANN) for meta-learning. MANN is an implementation of a Neural Turing Machine (NTM) with the ability to rapidly assimilate new data in its memory, and use this data to make accurate predictions. In models such as the MANN, the input data samples and their appropriate labels from previous step are bound together in the same memory locations. This often leads to memory interference when performing a task as these models have to retrieve a feature of an input from a certain memory location and read only the label information bound to that location. In this paper, we tried to address this issue by presenting a more robust MANN. We revisited the idea of meta-learning and proposed a new memory augmented neural network by explicitly splitting the external memory into feature and label memories. The feature memory is used to store the features of input data samples and the label memory stores their labels. Hence, when predicting the label of a given input, the memory augmented network with separate feature and label memory unit uses the feature memory unit as a reference to extract the stored feature of the input, and based on that feature, it retrieves the label information of the input from the label memory unit. In order for the network to function in this framework, a new memory-writing module to encode label information into the label memory in accordance with the meta-learning task structure is designed. Here, we demonstrate that the memory-augmented network outperforms MANN by a large margin in supervised one-shot classification tasks using Omniglot and MNIST datasets.

1 Introduction

Deep learning is heavily dependent on big data. Traditional gradient based neural networks require extensive and iterative training using large datasets. In these models, training occurs through a continuous update of weight parameters in order to optimize the loss function during training. However, when there is only a little data to learn from, deep learning is prone to poor performance because traditional networks will not acquire enough knowledge about the specific task

via weight updates, and hence, they fail to make accurate predictions when tested.

Previous works have approached the task of learning from few samples using different methods such as probabilistic models based on Bayesian learning (Fei-Fei, Fergus, and Perona 2006), generative models using probability density functions (Lake et al. 2011; Rezende et al. 2016), Siamese neural networks (Koch 2015), and meta-learning based memory augmented models (Santoro et al. 2016; Vinyals et al. 2016).

In this work, we revisited the problem of meta-learning using memory augmented neural networks. Meta-learning is a two-tiered learning framework in which an agent learns not only about the specific task, for instance, image classification, but also about how the task structure varies across target domains (Christophe, Ricardo, and Pavel 2004; Santoro et al. 2016). Neural architectures with an external memory such as Neural Turing Machines (NTMs) (Graves, Wayne, and Danihelka 2014) and memory networks (We- ston, Chopra, and Bordes 2014) have shown the ability of meta-learning.

Recent memory augmented neural networks for meta-learning such as MANN (Santoro et al. 2016) use a plain memory matrix as an external memory. In these models, input data samples and their labels are bound together in the same memory locations. In models such as the MANN, the input data samples and their appropriate labels from previous step are bound together in the same memory locations. This often leads to memory interference when performing a task as they have to retrieve a feature of an input from a certain memory location and read only the label information bound to that location.

Our primary contribution in this work is designing a different version of NTM (Graves, Wayne, and Danihelka 2014) by splitting the external memory into feature and label memories to avoid any catastrophic interference. The feature memory is used to store input data features and the label memory is used to encode the label information of the inputs. Therefore, during testing, ideal performance in our model requires using the feature memory as a reference to accurately retrieve the stored feature of an input image and effectively reading the corresponding label information from the label memory. In order to accomplish this, we designed a new memory writing module based on the meta-learning

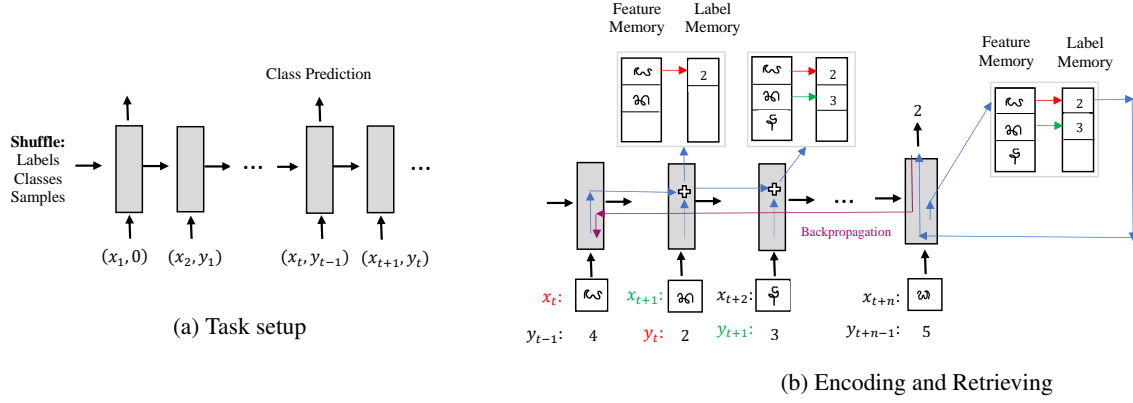


Figure 1: Meta-learning task structure. (a) Omniglot images, x_t , are presented along with labels in a temporally offset manner. At time step t , the network sees an input image x_t with a label y_{t-1} from the previous time step. Labels are also shuffled from episode to episode. This prevents the model from learning sample-class bindings via weight updates, instead it learns to regulate input and output information using its two memories. (b) Here is how the model works. When the network sees an input image for the first time at a certain time step, it stores a particular feature of the input in the feature memory. When the appropriate label is presented at the next time step, the network stores the label information of the input in the label memory. Then, sample-class bindings will be formed between the two memories. When the network is given the same class of image in later time step, the network retrieves the input feature from the feature memory and uses the retrieved information to read the corresponding label memory for prediction.

task structure that monitors the way in which information is written into the label memory.

2 Related Work

Our work is based on a recent work done by a Santoro et al. They approached the problem of one shot learning with the notion of meta learning and suggested a Memory Augmented Neural Network (MANN). MANN is an implementation of NTM (Graves, Wayne, and Danihelka 2014) with an ability to rapidly assimilate new data, and use this data to make accurate predictions after a few samples.

In previous implementation of NTM, memory was addressed both by content and location. However, in their work, they presented a new memory access module. This memory access module is called Least Recently Used Access (LRUA)(Santoro et al. 2016). It is a pure content-based memory writer that writes memories either to the least recently used location or to the most recently used location of the memory. According to this module, new information is written into rarely used locations (preserving recently encoded information) or it is written to the last used location (to update the memory with newer, and possibly relevant, information).

3 Task Methodology

In this work, we used a similar task structure used in recent works (Santoro et al. 2016; Vinyals et al. 2016). As we implemented supervised learning, the model is tasked to infer information from a labelled training data. This involves presenting the label y_t along with input x_t at time step t . However, in our work, the training data was presented in the following manner: $D = \{(x_t, y_{t-1})\}_{t=1}^T$,

where D is the dataset, x_t is the input at time step t and y_{t-1} is the class label from previous time step $t - 1$. Therefore, the model sees the following input sequence: $(x_1, 0), (x_2, y_1), \dots, (x_T, y_{T-1})$ (Figure 1(a)).

Moreover, the label used for a particular class of input images in a certain episode is not necessarily the same as the label used for the same class of input images in another episode. Random shuffling of labels is used from episode to episode in order to prevent the model from slowly learning sample-class bindings in its weights. Instead, it learns to store input information into the feature memory and store the corresponding output information into the label memory, when presented at the next time step, after which sample-class bindings, between the input features in the feature memory and the class labels in the label memory, will be formed for later use (Figure 1(b)).

4 Memory Augmented Model

Neural Turing Machine (NTM) (Graves, Wayne, and Danihelka 2014) is a memory augmented neural network that has two main components: a controller and an external memory. It can be seen as a differentiable version of a Turing machine. The controller is a neural network that provides an internal representation of the input used by read and write heads to interact with the external memory. It can be either feed-forward or recurrent neural network.

In this work, we designed a memory augmented neural network, a different version of NTM, with its memory split into partitions: Feature memory (M^f) and Label memory (M^l). The feature memory is used as a reference memory to retrieve the stored representation of an input data. The label memory is used to read an output information of the input based on the retrieved information from the feature memory.

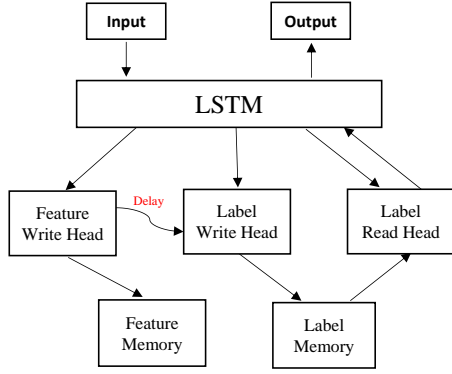


Figure 2: Feature-Label Memory Network (FLMN). It a memory augmented neural network with LSTM controller and two memories (Feature and Label memories). Input features are encoded into the feature memory using the feature memory write head. Labels of the inputs are written into the label memory using the label memory write head. The two write heads are linked recursively in accordance with the task structure. Label read head is used to read label information of a given input from the label memory.

In our model, we used Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber 1997) as a controller due to its better performance compared to other controller models. Figure 2 shows the high-level diagram of our model.

Feature-Label Memory Network (FLMN) has two memories, and hence, has two write heads. Feature memory write head writes into the feature memory (M^f). Label memory write head is a writer to the label memory (M^l). Even though information is encoded in both memories, output information is read only from the label memory using the label read head.

Here is how our model works. Given some input x_t at time step t , the controller produces three interface vectors, k_t , a_t^f and a_t^l . Key vector (k_t) is used to retrieve a particular memory, i , from a row of the feature memory; i.e. $M_t^f(i)$. Add vectors (a_t^f and a_t^l) are used to modify the content of feature memory (M_t^f) and label memory (M_t^l), respectively.

4.1 Reading from the Label Memory

Before the output information of the input image x_t is read from the label memory, the corresponding feature of x_t is retrieved from the feature memory using a key k_t . When retrieving memory, the i^{th} row of the feature memory (M_t^f) is addressed using *cosine similarity* measure,

$$K[k_t, M_t^f(i)] = \frac{k_t \cdot M_t^f(i)}{\|k_t\| \cdot \|M_t^f(i)\|} \quad (1)$$

This measure, K , is then used to produce read-weight vector (w_t^r) whose elements are computed according to the follow-

ing softmax:

$$w_t^r(i) \leftarrow \frac{\exp(K[k_t, M_t^f(i)])}{\sum_j \exp(K[k_t, M_t^f(j)])} \quad (2)$$

The read weights are then used to read from label memory (M^l). The read memory, r_t , is computed as follows,

$$r_t \leftarrow \sum_i w_t^r(i) M_t^l(i) \quad (3)$$

4.2 Writing into the Feature Memory

In order to write into the feature memory, we implemented the LRUA module (Santoro et al. 2016) with slight modifications. According to this module, new information is written either into rarely used locations or to the last used location. The distinction between these two options is accomplished by an interpolation using usage weight vector w_t^u .

The usage weight vector w_t^u at a given time step is computed by decaying the previous usage weights and adding the current write weights of the feature memory w_t^{wf} and read weights as follows,

$$w_t^u \leftarrow \gamma w_{t-1}^u + w_t^{wf} + w_t^r \quad (4)$$

where, γ is a decay parameter.

In order to access the least-used location of the feature memory, least-used weight vector w^{lu} is defined from the usage weight vector w^u ,

$$w^{lu}(i) = \begin{cases} 1 & \text{if } w^u(i) = \min(w^u) \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Write weights for the feature memory (w^{wf}) are then obtained by using a learnable sigmoid gate parameter to compute a convex combination of the previous read weights and previous least-used weights.

$$w_t^{wf} \leftarrow \sigma(\alpha) w_{t-1}^r + (1 - \sigma(\alpha)) w_{t-1}^{lu} \quad (6)$$

where, $\sigma(\alpha) = \frac{1}{1 + \exp(-\alpha)}$ and α is a scalar gate parameter to interpolate between weights.

Therefore, new content is written either to the previously used memory (if $\sigma(\alpha)$ is 1) or the least-used memory (if $\sigma(\alpha)$ is 0). Before writing into the feature memory, the least used location of the memory is cleared. This can be done via element-wise multiplication using the least-used weights from the previous time step:

$$M_t^f(i) \leftarrow M_{t-1}^f(i) \cdot (1 - w_{t-1}^{lu}(i)), \quad \forall i \quad (7)$$

Then writing into memory occurs in accordance with the computed weight vectors using the feature add vector (a_t^f) as follows,

$$M_t^f(i) \leftarrow M_t^f(i) + w_t^{wf}(i) a_t^f, \quad \forall i \quad (8)$$

4.3 Writing into the Label Memory

According to (3), the read memory r_t is retrieved from the label memory $M_t^l(i)$ using the read weights w_t^r with the elements computed using (2) which involves the feature memory $M_t^f(i)$. Hence, the label memory should be written in a similar manner as the feature memory so that when an input image x_t is provided to the network at time step t , the network retrieves the stored feature of the input from M_t^f and based on that feature, it extracts the label of the input image from M_t^l .

In order to accomplish the above scenario, we designed a new memory writing module for the label memory. The new module is based on the task setup in which the model was trained. As mentioned earlier, during training, the model sees the following input sequence: $(x_1, 0), (x_2, y_1), \dots, (x_T, y_{T-1})$. The label y_t at time step $t + 1$ is the appropriate label for the input x_t which was presented along with the label y_{t-1} at time step t . Based on this observation, we designed a **recursive memory writing module**.

According to this module, the label memory write-weight vector w^{wl} at time step t is computed from the previous feature memory write-weight vector w_{t-1}^{wf} in a recursive manner as follows,

$$w_t^{wl}(i) \leftarrow w_{t-1}^{wf}(i) \quad (9)$$

The label memory (M^l) is then written according to the write weights w^{wl} using the label add vector a_t^l .

$$M_t^l(i) \leftarrow M_{t-1}^l(i) + w_t^{wl}(i)a_t^l, \quad \forall i \quad (10)$$

This memory is then read as shown in (3) to give a read memory, r , which will be used by the controller as an input to a softmax classifier, and as an additional input for the next controller state.

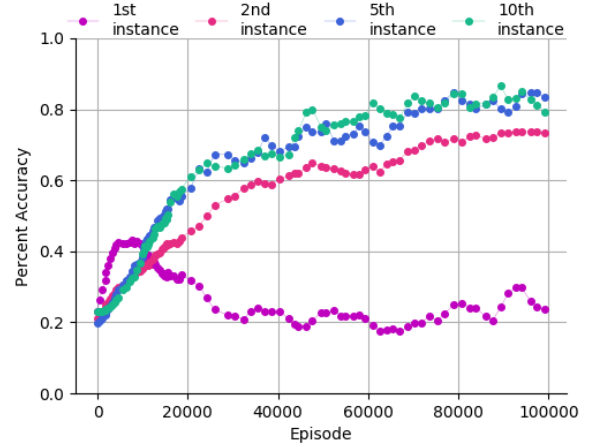
Based on this module, the label y_t at time step $t + 1$ will be written into the label memory in the same manner as the input x_t (from the previous time step t) was written into the feature memory. This enhances the model to accurately retrieve input information from the feature memory and use this feature to effectively read the corresponding output information from the label memory without any interference.

5 Experimental Results

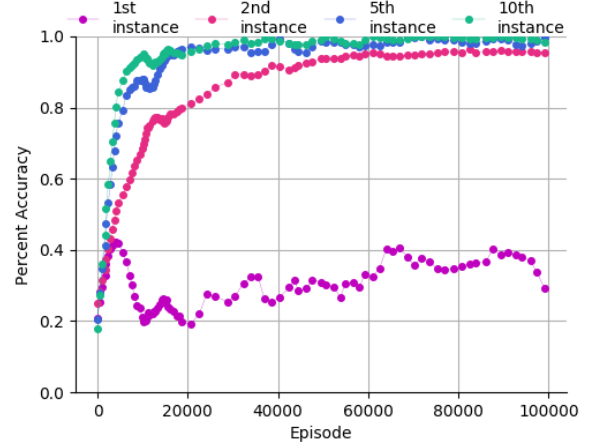
We tested our model in one-shot image classification tasks using Omniglot and miniMNIST datasets. The omniglot dataset consists of 1623 characters from 50 different alphabets. The number of samples per each class (character) is 20. The dataset is also called MNIST transpose due to the fact that it contains large number of classes with relatively few data samples per class. This makes the dataset ideal for one-shot learning.

5.1 Experiment Setup

In this work, we implemented both our model and MANN (Santoro et al. 2016) and compared their performance in supervised one-shot classification tasks. However, the experimental settings we used for implementing MANN are



(a) Experiment I. Training accuracy for MANN

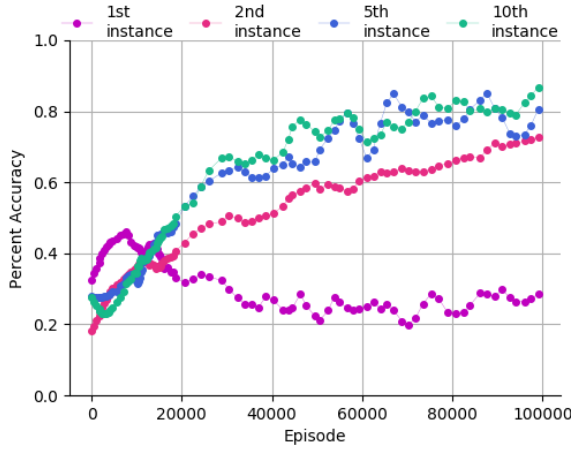


(b) Experiment I. Training accuracy for FLMN

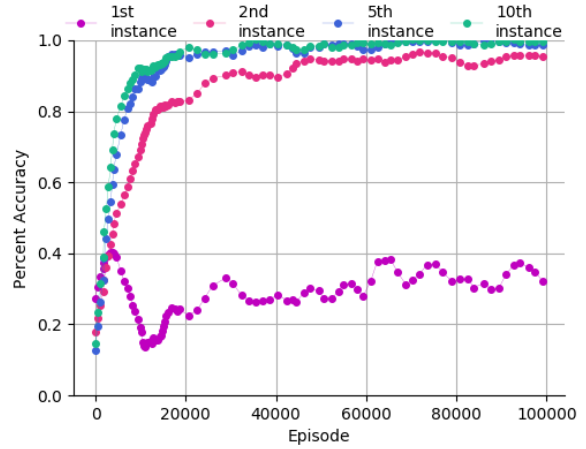
Figure 3: Omniglot classification. No data augmentation was performed. In (a) and (b), each episode contains 5 classes and 10 samples per each class. As expected, the 1st instance accuracy is quite low in both models. This is because the models have to do a random guess for the first presentation of the class. However, as we can see from (b), FLMN 1st instance accuracy is more than a blind guess, especially after 20,000 episodes, which indicates that the model is making an educated guess for new classes based on the previous classes it has already seen. For the 2nd and other instances, both models use their memory to achieve better accuracy. The 2nd instance accuracy of FLMN has reached 80% with in the first 20,000 episodes while the 2nd instance accuracy of MANN reached only to 40% accuracy.

slightly different from the implementation of MANN in the original paper (Santoro et al. 2016).

In the paper, the number of reads from the memory used was four. Data augmentation was performed by randomly translating and rotating character images. New classes were also created through 90°, 180° and 270° rotations of existing data. A minibatch size of 16 was used.



(a) Experiment II. Training accuracy for MANN



(b) Experiment II. Training accuracy for FLMN

Figure 4: Omniglot classification. Data augmentation was performed via rotating and translating random character images in an episode. Each episode contains 5 classes and 10 samples per each class. As we can see from (a) and (b), our model has outperformed MANN by displaying better training accuracies for each instances.

In our case, one read from memory was used. In order to make a fair comparison, we tried to balance the memory of the two models. we used an $N \times M$ memory matrix for MANN, where N is the number of memory locations, and M is the size at each location. For our model, we split the memory into two and we used $N/2 \times M$ memory matrix for each memory. Using these settings, we performed three types of experiments.

5.2 Experiment: Type I

In the first experiment, the original omniglot dataset was used without performing any data modification. Out of the 1623 available classes, 1209 classes were used for training and the rest 414 classes were used for testing the models. Note that these two sets are disjoint. Therefore, after training, both models were tested with never-seen omniglot classes. For computational simplicity, image sizes were down scaled to 20×20 . One-hot vector representations were used for class labels and training was done using 100,000 episodes. Several experiments were performed for different number of classes (and different number of samples per each class) in an episode. Figure 3 shows the training accuracy of the models for 5 classes and 10 samples (per each class) in an episode.

As we can see from Figure 3, our model has outperformed MANN in making accurate predictions. The 2nd instance accuracy of our model has reached nearly 80% accuracy within the first 20,000 episodes of training, while the 10th instance accuracy of MANN could only reach 60% accuracy.

5.3 Experiment: Type II

In our second experiment, we performed data augmentation without creating new classes. The dataset was augmented by rotating and translating random character images of an episode. The angle for rotation was chosen randomly from a uniform distribution $[-\frac{\pi}{16}, \frac{\pi}{16}]$ with a size of an episode. This

was accompanied by a translation in the x and y dimensions with values uniformly sampled between -10 and 10 pixels. Images were then downscaled to 20×20 .

In a similar manner as the previous experiment, 1209 classes (plus augmentations) were used for training and 414 classes (plus augmentations) were used for testing. Figure 4 shows the training accuracy of MANN and our model for 100,000 episodes.

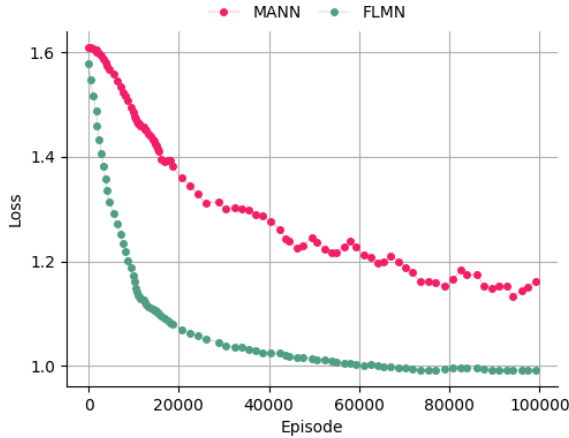
Not only has our model performed better in making accurate predictions but also has learned faster than MANN. This can be shown by plotting the *loss graph* of training for the two experiments (Figure 5).

In both types of experiments, the training process has stopped at the mark of the 100,000 episode. Without any further training, the models were tested with never-seen omniglot classes from the testing set. The testing results are summarized in the Table 1. We borrowed the test result of MANN from Santoro et al. for a reference.

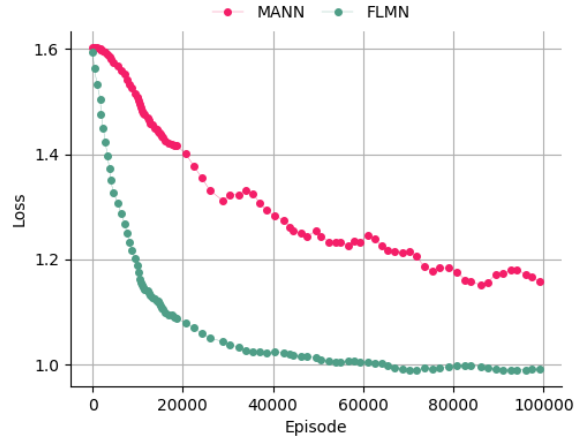
As we can see from the table, our model has demonstrated higher classification accuracy in both experiments compared to MANN. FLMN has reached an accuracy of 85.6% (Experiment I) and 86.5% (Experiment II) on just second presentation of an input sample from a class with in an episode reaching up to 94.1% and 94.4% accuracy by the 10th instance, respectively. On the other hand, MANN achieved an accuracy of 66.7% (Experiment I) and 65.5% (Experiment II) in the 2nd instance reaching up to 78.1% and 77.2% accuracy by the 10th instance, respectively.

5.4 Zero-shot learning

In this experiment, the models were tasked to perform MNIST classification after being trained with omniglot dataset. We used 1209 classes of omniglot dataset for training. For testing, we prepared a miniMNIST dataset. miniMNIST contains only 20 image samples per each class which are randomly selected from the original MNIST



(a) Experiment I. Loss graph for MANN and FLMN



(b) Experiment II. Loss graph for MANN and FLMN

Figure 5: Loss graph. For the same learning rate, the loss graph of FLMN has fallen sharply compared to the loss graph of MANN in both (a) and (b). This indicates that FLMN has learned the task of one-shot classification faster than MANN and explains why it has demonstrated higher training accuracy within the first few episodes.

Table 1: Test-set classification accuracies of MANN and FLMN for Experiment I and Experiment II

| Model | INSTANCE (% CORRECT) | | | | | |
|-----------------------------|----------------------|-----------------|-----------------|-----------------|-----------------|------------------|
| | 1 st | 2 nd | 3 rd | 4 th | 5 th | 10 th |
| MANN (Santoro et al. 2016) | 36.4 | 82.8 | 91.0 | 92.6 | 94.9 | 98.1 |
| MANN (Experiment I) | 21.6 | 66.7 | 74.0 | 76.0 | 76.3 | 78.1 |
| FLMN (Experiment I) | 31.1 | 85.6 | 88.7 | 89.5 | 91.0 | 94.1 |
| MANN (Experiment II) | 22.2 | 65.5 | 72.0 | 74.5 | 76.4 | 77.2 |
| FLMN (Experiment II) | 33.9 | 86.5 | 89.1 | 89.7 | 91.1 | 94.4 |

dataset. The images were downsampled to 20×20 . After 100,000 episodes of training, the models were tested with never-seen MNIST classes. Testing results are summarized in the following table.

Table 2: Test-set classification accuracies of MANN and FLMN for zero-shot learning

| Model | INSTANCE (% CORRECT) | | |
|-------------|----------------------|-----------------|------------------|
| | 1 st | 2 nd | 10 th |
| MANN | 14.6 | 37.3 | 52.0 |
| FLMN | 28.5 | 67.6 | 80.5 |

As we can refer from Table 2, FLMN was able to achieve 80.5% accuracy on the 10th instance in classifying never-seen-before images from miniMNIST dataset after being trained with omniglot dataset.

6 Conclusion

In this paper, we implemented meta-learning framework and proposed Feature-Label Memory Network (FLMN). The novelty of our model is that it stores input data samples and their matching labels into separate memories preventing any memory interference. We also introduced a new memory writing method associated with the task structure

of meta-learning. We have shown that our model has outperformed MANN in supervised one-shot classification tasks using Omnigot and miniMNIST datasets. Future work includes testing our model with more complex datasets and experimenting the performance of our model in other tasks.

References

- Christophe, G.-C.; Ricardo, V.; and Pavel, B. 2004. Introduction to the special issue on meta-learning. *Introduction to the special issue on meta-learning. Machine learning*, 54(3):187193.
- Fei-Fei, L.; Fergus, R.; and Perona, P. 2006. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence* 28(4):594–611.
- Graves, A.; Wayne, G.; and Danihelka, I. 2014. Neural Turing machines. *arXiv preprint arXiv:1410.5401*.
- Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural Computation* 9(8):1735–1780.
- Koch, G. 2015. Siamese neural networks for one-shot image recognition. *PhD thesis, University of Toronto*.
- Lake, B. M.; Salakhutdinov, R.; Gross, J.; and Tenenbaum, J. B. 2011. One shot learning of simple visual concepts. *In Proceedings of the 33rd Annual Conference of the Cognitive Science Society* 72:2.

- Rezende, D. J.; Mohamed, S.; Danihelka, I.; Gregor, K.; and Wierstra, D. 2016. One-shot generalization in deep generative models. *In Proceedings of the 33rd International Conference on Machine Learning, JMLR:W&CP* 48.
- Salakhutdinov, R.; Tenenbaum, J.; and Torralba, A. 2012. One-shot learning with a hierarchical nonparametric bayesian model. *Proceedings of ICML Workshop on Unsupervised and Transfer Learning, PMLR* 27:195–206.
- Santoro, A.; Bartunov, S.; Botvinick, M.; Wierstra, D.; and Lillicrap, T. 2016. Meta-learning with memory-augmented neural networks. *In Proceedings of The 33rd International Conference on Machine Learning*, 1842–1850.
- Vinyals, O.; Blundell, C.; Lillicrap, T.; Kavukcuoglu, K.; and Wierstra, D. 2016. Matching networks for one shot learning. *In Advances in Neural Information Processing Systems* 3630–3638.
- Weston, J.; Chopra, S.; and Bordes, A. 2014. Memory networks. *arXiv preprint arXiv:1410.3916*.
- Woodward, M., and Finn, C. 2017. Active one-shot learning. *arXiv preprint arXiv:1702.06559*.