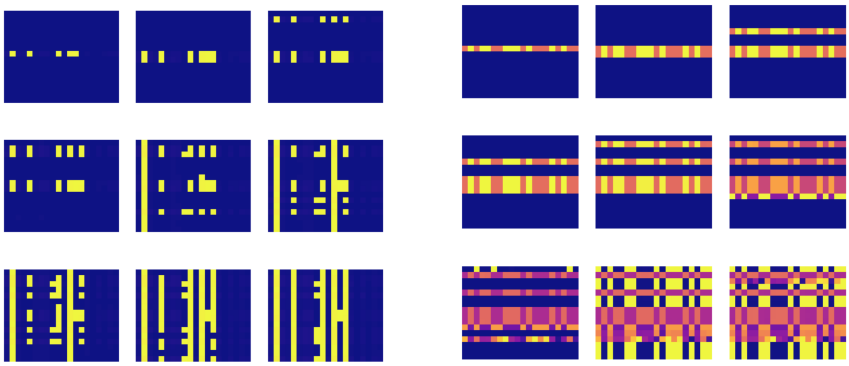
Memory Forcing

Arip Asadulaev, Vladislav Tretyak, Andrey Filchenkov

Abstract

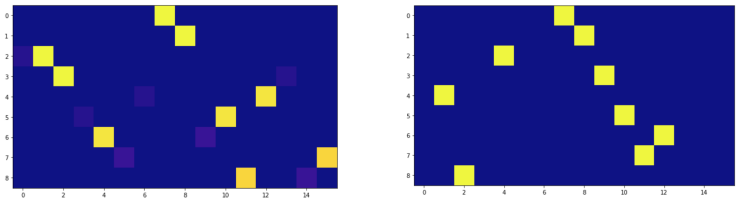
In this paper we proposed Memory Forcing algorithm for Memory Augmented Neural Network(MANN). The main concept of proposed method is training controller of MANN, depending on the state of it's memory matrix and form better interfaces with external memory. We test our model on classical Copy Task. Also we present a new approach of training recurrent neural networks, this algorithm can be presented as next stage of Professor Forcing algorithm[7]. During training, discriminator performs adversarial learning on memory matrix and makes domain adaptation, to keep the dynamics of the MANN the same, during training and sampling phases. We apply Memory Forcing algorithm for classic Copy Task and in sequence generation task using Sequential variant of MNIST dataset.

1 Introduction



(a) incorrect prediction (b) correct prediction

Figure 1: Dynamics of MANN memory matrix for every timestep during Copy Task



(a) incorrect prediction (b) correct prediction

Figure 2: Dynamics of MANN write weights for every timestep during Copy Task

Preprint. Work in progress.

Is there any additional task that improve Neural Networks quality on main task? Yes, it is memory, it appears that Neural Networks become significantly more powerful if they are able to learn to interact with external interfaces, as it is presented in section 2.

Is it possible to recreate such a main task, by solving which, we will get an improvement of NN quality on any additional task? Our hypothesis is, that this task is learning to interface with external memory. We consider direction when memory using as a main task and firstly model always trains to interact with memory. In this settings, other tasks become additionally. Our experiments have shown that already at this stage, network performance is displayed on memory, figures 1 and 2 illustrate a clear visual difference in the dynamics of memory and its interfaces with correct and incorrect prediction.

Approches that solve task based on memory called Memory Based Learning(cite) and Memory-Based Language Processing(cite) which is inspired by work in pre-Chomskyan linguistics, categorization psychology, and statistical pattern recognition. In this paradigms learning is the storage of examples in memory, and processing is similarity-based reasoning with these stored examples, that is, memory in these tasks plays a major role.

We suppose that Memory-based learning necessary in order to set up a universal fitness function for any task. In deep learning(cite), we have a huge amount of teaching methods for different types of data and different tasks, such as reinforcement learning, Supervised Learning, Unsupervised Learning, and so on. Each of these methods has subdomains for more specific types of tasks, and for each of them a special loss function is used. We believe that Memory Forcing is the step towards the development of neural versions of Memory-Based Learning, where memory as a universal loss function allows you to train models on many different tasks.

On this stage our idea is to try to increase memory significance on training process, we are presenting a simple method that allows to train models with respect to memory dynamics, called Memory Forcing.

Neural Network based methods that using memory mechanisms as an accelerator of learning on sequential data called Memory Augmented Neural Networks. A huge breakthrough was made by Alex Graves in model called Neural Turing Machine (NTM)[3], which learns to interact with an external memory that is differentiable and continuous. Differentiable Neural Computer (DNC)[4] is extension of Neural Turing machine, which outperforms RNNs by a significant margin on several benchmark tasks.

To check memory reflection to controller dynamics we make some experiments in sequence generation domain. Existing methods in this field can be softly scored into two classes, which use methods of adversarial training[10] and which are based strictly on the techniques of language modeling. Both these methods have its own pros and cons. It is quite common practice, when this methods are integrated in order to generate better sequences.

There are several works which include techniques of adversarial training and language modeling[7][13][2]. The most obvious example of combining work of opposing training is method called Professor Forcing proposed by .., which showed impressive results in language modeling, vocal synthesis on raw waveforms, handwriting generation and image generation on Sequential MNIST[8]. This algorithm makes hidden states of Recurrent Neural Network (RNN) more similar whether the network inputs are self-generated and when it is trained in teacher forcing mode.

Some approaches, SeqGAN[13], MaskGAN[2], successfully use Reinforcement Learning (RL) to apply the Generative Adversarial Networks (GAN)[10] framework to sequential data. Where output of models passed to discriminator and its prediction is used as a reward in REINFORCE [11] algorithm.

The main contributions of this article are:

- We have proposed an improved method for training recurrent neural networks.
- Unlike all previously proposed memory architectures, our method is explicitly trained to form the correct memory representation.
- Our method outperforms other recurrent neural network based methods on benchmark task.

Our experiments were performed on classical problems for memory based neural networks.

2 Memory Benefits

2.1 Context Attention Interface

Neural Turing Machine[3] consists of neural network controller and a memory bank, this architecture learns to interact with an external memory that is differentiable and continuous. An external memory allows NTM to solve tasks that were previously unsolvable by conventional machine learning methods.

To interact with a memory matrix selective read and write operations is used. This operations are denoted as "heads." Both "heads" is managed by corresponding write and read weights w_t^w, w_t^r. To interact with memory using this weights at each timestep NTM output a set of vector and scalars:

$$\begin{aligned}
\hat{\mathbf{g}}^f, \hat{\mathbf{g}}^i, \hat{\mathbf{g}}^o, \hat{\mathbf{u}} &= \mathbf{W}^{xh}\left(\mathbf{x}_t, \mathbf{y}_{t-1}\right) + \mathbf{W}^{hh}\mathbf{h}_{t-1} + \mathbf{b}^h \\
\mathbf{g}^f &= \sigma\left(\hat{\mathbf{g}}^f\right) \\
\mathbf{g}^i &= \sigma\left(\hat{\mathbf{g}}^i\right) \\
\mathbf{g}^o &= \sigma\left(\hat{\mathbf{g}}^o\right) \\
\mathbf{u} &= \sigma\left(\hat{\mathbf{u}}^o\right) \\
\mathbf{c}_t &= \mathbf{g}^f \odot \mathbf{c}_{t-1} + \mathbf{g}^i \odot \mathbf{u}, \\
\mathbf{h}_t &= \mathbf{g}^o \odot \tanh\left(\mathbf{c}_t\right) \\
\mathbf{o}_t &= \left(\mathbf{h}_t, \mathbf{r}_t\right)
\end{aligned} \tag{1}$$

Where where $\hat{\mathbf{g}}^f, \hat{\mathbf{g}}^o,$ and $\hat{\mathbf{g}}^i$ are the forget, output, input gates. $\mathbf{c}^t$ is the cell state, $\mathbf{h}_t$ is the hidden state, $\mathbf{b}^h$ are the hidden state biases, $\mathbf{o}_t$ is the concatenated output of the controller. $\mathbf{W}^{xh}$ are input layer weights and $\mathbf{W}^{hh}$ are the weights between hidden states connected through time. And $\mathbf{r}_t$ is read vector, considered from memory:

$$\mathbf{r}_t \leftarrow \sum_i w_t^r(i)\mathbf{M}_t(i) \tag{2}$$

$\mathbf{M}_t(i)$ is a $N * M$ memory matrix at time t, where N is the number of memory locations and M is the size of each location. Writing to memory then occurs in accordance with the computed write weights w_t^w and key vector $\mathbf{k}_t$ produced by controller:

$$\mathbf{M}_t(i) \leftarrow \mathbf{M}_{t-1}(i) + w_t^w(i)\mathbf{k}_t, \forall i \tag{3}$$

Graves et. al [4] proposed an extension of Neural Turing Machine called DNC. DNC comprises a controller with external memory trained in the end-to-end manner.

In graph experiments, after almost two million training samples, LSTM reached an average of only 37 percent, while DNC reached up to 98.8 percent accuracy after one million training samples. During synthetic question answering trials using the bAbI question-answering data set, DNC has also outperformed LSTM by a significant margin. Coupled with RL, DNC are able to read and write sequence structures that are relevant for the reward. As well as in question answering task, in automatic molecule generation, the capacity and strength of the model play key roles and serve as the main argument for the selection of the DNC architecture for the generation of molecular structures.

In that work[9], authors used NTM for one shot classification. Authors demonstrate the ability of a memory-augmented neural network make accurate predictions after only a few samples.

2.2 Discrete Interfaces

Reinforcement Learning–Neural Turing Machine (RL-NTM)[14] model uses discrete interfaces to write and read from the memory. Due to discrete Interfaces are not differentiable, authors used Reinforcement Learning methods to learn to interact with memory by dint of discrete interfaces. In this work, REINFORCE algorithm is used to learn where to access the discrete interfaces and to use Backpropagation to write into memory.

Both Backpropagation and REINFORCE maximize the expected log probability of outputs, where the expectation is taken over all possible sequences of actions, weighted with probability of taking these actions. This model achieves significant results on the "repeat copy" task and also on reversing a sequence task.

2.3 Bayes Interfaces

In generation domain, inspired by Kanerva's sparse distributed memory authors propose memory augumented network[12]. This Variational Autoencoder(VAE)[6] based model can adapt to new samples and generate it, after few training iterations. Authors derive a Bayesian memory update rule that optimally trades-off save new content and provides effective compression and storage of complex old data.

To train this model, authors optimize a variational lower-bound of the conditional likelihood J,

$$\begin{aligned} \mathcal{J} = \int p(X, M) \ln p_\theta(X|M) \mathrm{d}M \mathrm{d}X = \\ \int p(X) p(M|X) \sum_{t=1}^{T} \ln p_\theta \left(x_t|M\right) \mathrm{d}M \mathrm{d}X \end{aligned} \tag{4}$$

Where M is a memory and X in samples from dataset D And conditional likelihood can be derived in a fashion similar to standard VAEs:

$$\begin{aligned} \mathcal{L} = \mathbb{E}_{q_\phi(M|X)p(X)} \sum_{t=1}^{T} \{ \mathbb{E}_{q_\phi(y_t, z_t|x_t, M)} \left[\ln p_\theta \left(x_t|z_t\right)\right] \\ - D_{KL} \left(q_\phi \left(y_t|x_t\right) \| p_\theta \left(y_t\right)\right) \\ - D_{KL} \left(q_\phi \left(z_t|x_t, y_t, M\right) \| p_\theta \left(z_t|y_t, M\right)\right) \} \end{aligned} \tag{5}$$

3 Method

In current article, we show a class of methods that uses memory as an additional function of loss, for more skillful memory shaping and therefore improving network performance.

Technically, for sequence generation task, our method is a combination of MANN and Professor Forcing algorithm. Professor Forcing algorithm makes dynamics of it's hidden states indistinguishable whether the network is trained with teacher forcing mode or at free-running generative mode when its inputs are self-generated. Additional to hidden states, our method tries to make indistinguishable memory states of MANN during teacher forcing and free-running. The proposed architecture consists of a single generative architecture, which is based on context attention Interface with memory and 2 discriminators from GANs framework. First discriminator distinguishes the hidden state of the recurrent network during Teacher Forcing and Free Running. Second discriminator distinguishes memory state during Teacher Forcing and Free Running.

Parameters θ_g of RNN generator are trained to maximize the likelihood of the data and to fool the discriminator. We considered two variants of latter. Negative log-likelihood objective (a) is usual teacher-forced training criterion for RNNs:

$$NLL\left(\boldsymbol{\theta}_g\right) = E_{(\boldsymbol{x}, \boldsymbol{y}) \sim \mathrm{data}} \left[-\log P_{\boldsymbol{\theta}_g}(\boldsymbol{y}|\boldsymbol{x})\right] \tag{6}$$

The discriminators D parameters θ_d are trained to maximize likelihood of correctly classifying a behavior sequence:

$$\begin{aligned} C_d\left(\boldsymbol{\theta}_d|\boldsymbol{\theta}_g\right) = E_{(\boldsymbol{x}, \boldsymbol{y}) \sim \mathrm{data}} \left[-\log D\left(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}_g\right), \boldsymbol{\theta}_d\right) \\ + E_{\boldsymbol{y} \sim P_{\boldsymbol{\theta}_g}(\boldsymbol{y}|\boldsymbol{x})} \left[-\log\left(1 - D\left(B\left(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}_g\right), \boldsymbol{\theta}_d\right)\right)\right] \end{aligned} \tag{7}$$

We ask free-running teacher-forced behavior to be indistinguishable from teacher-forced behavior:

$$C\left(\boldsymbol{\theta}_g|\boldsymbol{\theta}_d\right) = E_{\boldsymbol{x} \sim \mathrm{data}, \boldsymbol{y} \sim P_{\boldsymbol{\theta}_g}(\boldsymbol{y}|\boldsymbol{x})} \left[-\log D\left(B\left(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta}_g\right), \boldsymbol{\theta}_d\right)\right] \tag{8}$$

We use equations (7) for training Hidden States and Memory Discriminators, and equations (8) for training generator in both domains. Totally we update our generator parameter using this function:

$$NLL + C_h + C_m \tag{9}$$

Where C_h and C_m equation (8) calculated for Hidden states and memory matrix.

Another type of Memory Forcing algorithm consist in training discriminator to classify the memory dynamics with correct and incorrect network predictions. Based on this, it trains the controller to make more accurate predictions using adversarial training. In this case, the loss function takes the form:

$$NLL + C_p \tag{10}$$

Here C_p is equation (8) if D is positive/negative memory classifier. Figure 2 shows the memory samples and the sequence of memory read addresses with incorrect and correct predictions.

4 Experiments

4.1 Sequence Generation

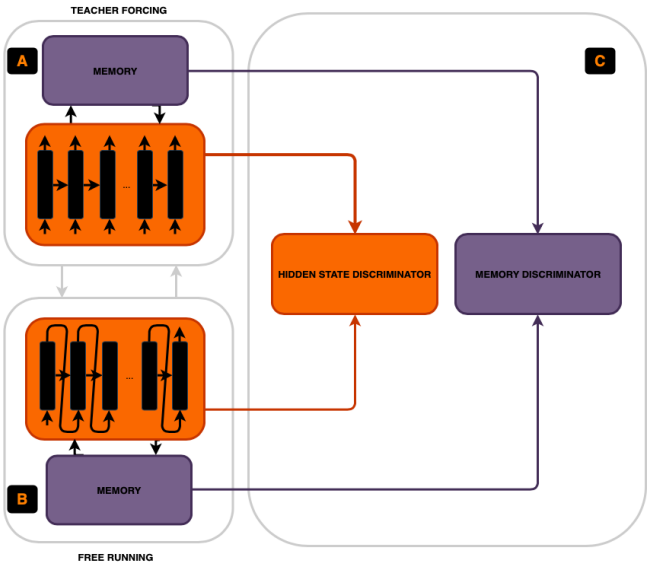
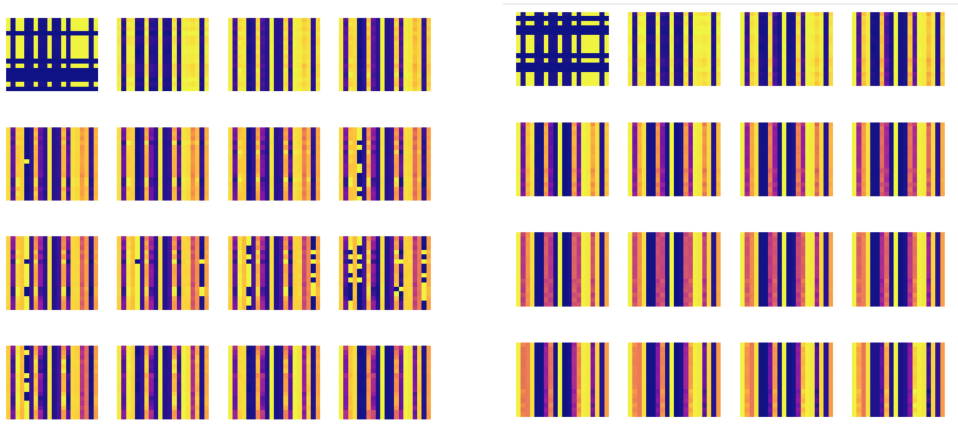


Figure 3: Memory Forcing for sequence generation

Table 1: MNIST NLL, DNC models was trained only on 10 epochs.

METHOD	MNIST NLL
DBN 2HL	84.55
NADE	88.33
EONADE-5 2HL	84.68
DLGM 8 LEAPFROG STEPS	85.51
DARN 1HL	84.13
DRAW	80.97
PIXEL RNN	79.2
PROFESSOR FORCING	79.58
DNC TEACHER FORCING	46.55
DNC PROFESSOR FORCING	46.30
DNC MEMORY FORCING	46.30

We randomly choose sample and make prediction using Teacher Forcing and Free Running. As seen on example 5, the dynamics for Teacher Forcing is more complicated than at Free Running phase, based on this we try to train our model with respect to memory dynamics. We trained RNN discriminators, that can process variable length inputs. We evaluated Memory Forcing on the task of



(a) Teacher Forcing (b) Free Running

Figure 4: Dynamics of MANN memory matrix for every 50 timestep during single sample generation

sequentially generating pixels of digits from MNIST dataset. We used standard binarized MNIST dataset[8]. We used 1 layer GRU[1]as a controller, with hidden state size of 64. Memory size was set to 20*16 with 4 read heads. For generator we used Adam [5] optimizer with learning rate 1e-4, eps 1e-9, and betas [0.9, 0.98].

(a) Teacher Forcing (b) Professor Forcing (c) Memory Forcing

Figure 5: DNC results with different training techniques after 10 epoch learning on SeqMNIST dataset

Hidden state Discriminators was 6 layer network, with first hidden layer GRU layer and 3 Dense layers, size of layer was set to [Size of controller hidden states, 128, 256, 8, 4, output size]. Memory Discriminators was also 6 layer network, with the same layer sizes as Hidden-state Discriminator, except that input size was 20 * 16 where 16 is the vector size at each location and 20 is the number of this locations.

Only for 10 epoch we train our generator with NLL loss and simultaneously with additional Memory and Hidden-States adversarial training. After 10 epochs, our model achieves the best reported likelihood on this task, despite the fact that our model is not very optimized. Results presented in Table 1 and Figure 5.

4.2 Copy Task

In this case we demonstrate ability of memory to reflect performance of the MANN. The learning algorithm is different form generation task. Here, before we start MANN training, we create buffer for memory samples at correct prediction and incorrect predictions. To determine whether the prediction is correct, we used the Euclidean distance between the output of the network and the goal. When we get a distance value equal to 0.0, we put the formed memory for this prediction into a buffer with

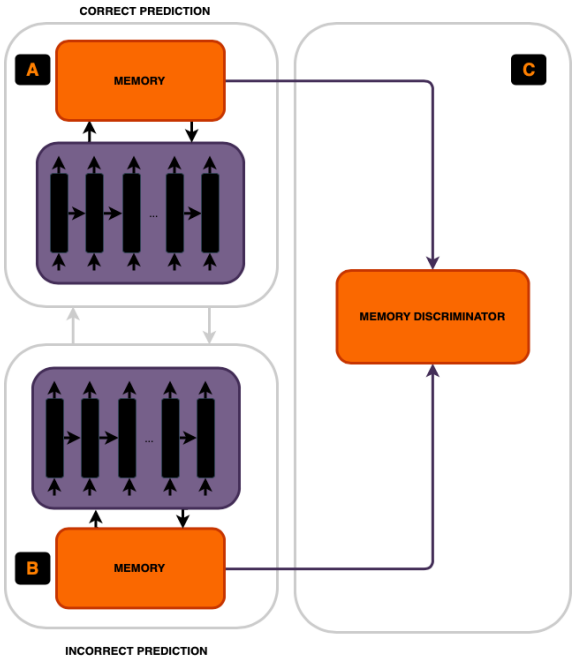


Figure 6: Memory Forcing for Copy Task

correct predictions and vice versa, if the distance is greater than zero. When the buffer size is larger than the batch size for training the memory discriminator, we start adversarial training. Figure 4 shows the Memory Forcing scheme for this task.

Figures 1 and 2 illustrate a clear visual difference in the dynamics of memory and its interfaces with correct and incorrect prediction with 9*6 input size. Since at incorrect predictions sample of memory and write weights have 10^{-3} size. To make plot we multiply them by 10^2. In this task model has the same hyper parameters settings with Sequential MNIST generator.

Every target sequence for this task was 12 * 6, external memory was size of 20*16

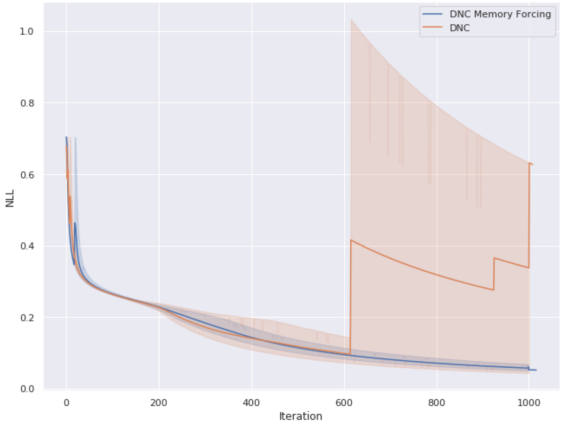


Figure 7: NLL curve for DNC with Memory Forcing regularization on Copy Task

Table 2: Number of correct predictions during 100 000 learning timesteps. Number of correct prediction was counted as mean for 3 parallel experiments for each model.

METHOD	CORRECT PREDICTIONS
DNC	45343
DNC MEMORY FORCING	50374

5 Conclusion and Future Work

In this work we proposed a sample-efficient method for improving training of MANNs, our Method is called Memory Forcing. We tested our method on Copy Task and benchmark Sequential MNIST generation task and compared it with other methods tested on this task.

We plan to develop in future suitable memory mechanisms which even more reflect on the behavior of the model.

The next step in use of memory is reinforcement learning tasks with rare rewards, where a well-formed memory will serve as a source of reward. Also we are planning to apply our technique for MANN with other types of memory interface, such as Kanerva Machine.

References

[1] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[2] William Fedus, Ian J. Goodfellow, and Andrew M. Dai. Maskgan: Better text generation via filling in the. *CoRR*, abs/1801.07736, 2018.

[3] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014. cite arxiv:1410.5401.

[4] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, October 2016.

[5] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[6] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[7] Alex Lamb, Anirudh Goyal, Ying Zhang, Saizheng Zhang, Aaron Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. `https://arxiv.org/pdf/1610.09038.pdf`, 2016.

[8] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[9] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy P. Lillicrap. One-shot learning with memory-augmented neural networks. *CoRR*, abs/1605.06065, 2016.

[10] Salimans Tim, Goodfellow Ian J., Zaremba Wojciech, Cheung Vicki, Radford Alec, and Xi Chen. Improved techniques for training gans. `http://arxiv.org/abs/1606.03498`, 2016.

[11] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[12] Yan Wu, Greg Wayne, Alex Graves, and Timothy P. Lillicrap. The kanerva machine: A generative distributed memory. *CoRR*, abs/1804.01756, 2018.

[13] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. *CoRR*, abs/1609.05473, 2016.

[14] Wojciech Zaremba and Ilya Sutskever. Reinforcement learning neural turing machines. *CoRR*, abs/1505.00521, 2015.