Online Restricted Boltzmann Machines

Sequential learning for compact representations and prediction

Saleh Albeaik and Jason Ramirez
EE290s Final Project Report
UC Berkeley
12/12/2018

1 Introduction

Restricted Boltzmann Machines (RBMs) [7] have been used to infer compact latent representations for data. An RBM is a generative stochastic neural network with two binary layers, one comprised of visible units and the other comprised of hidden units. The units are organized as a complete bipartite graph connecting the visible and hidden layers where edge connections correspond to weights. The early successes of RBM learning shed light into the training of deeper neural networks. The authors of [6] proposed an algorithm to learn RBMs sequentially in a manner that supports experience retention. They demonstrated that RBMs learned sequentially perform as well as those learned offline, and hence would constitute a powerful tool in dynamic environments.

The authors of [6] proposed the Online Contrastive Divergence with Generative Replay (OCD_{GR}) algorithm for sequential RBMs. They demonstrated that this approach can be used as a compact substitute for other memory intensive algorithms such as Experience Replay (ER). OCD_{GR} reconstructs memory from compact representations similar to the hippocampus of the human brain [5]. That is, unlike ER, the observations experienced when using OCG_{GR} are not explicitly stored, but instead, are approximate reconstructions that are dynamically generated for recall purposes. This makes RBMs and the operation of OCD_{GR}, supported by their biological plausibility, an interesting area of investigation in the context of online learning [2].

In this project, we aim to investigate the use of RBMs trained using OCD_{GR} as a compact way to represent and track arbitrary distributions of potentially large and complex observation spaces. The report proceeds by presenting RBMs and OCD_{GR} as proposed by authors. The report then shows that in the simplest sequence prediction scenario, the form of the prediction rule of the Multiplicative Weights (MW) algorithm is a special case of an optimal RBM. The report then presents performance comparison between sequentially learned RBMs and MWs on arbitrary sequence prediction scenarios (with and without memory). Finally, the report presents an example of using RBMs and MW for sequence prediction to form complex observation spaces, namely, from the MNIST[1] dataset.

2 Restricted Boltzmann Machines (RBMs)

This section presents a brief summary of the architecture of and learning algorithms for Restricted Boltzmann Machines (RBMs).

2.1 RBM Architecture and Design

Formally, an RBM consists of a hidden layer $\mathbf{h} = [h_1, h_2, \ldots, h_{n_h}] \in \{0,1\}^{n_h}$ of size n_h, and a visible layer $\mathbf{v} = [v_1, v_2, \ldots, v_{n_v}] \in \{0,1\}^{n_v}$ of size n_v. An RBM is parameterized by a set of biases $\mathbf{a} \in \mathbb{R}^{n_v}$ and $\mathbf{b} \in \mathbb{R}^{n_v}$, for the visible and hidden units respectively, and a weighted adjacency matrix $\mathbf{W} \in \mathbb{R}^{n_h \times n_v}$. The visible layer functions as the input layer, while the hidden layer provides the RBM with the ability to learn complex latent representations of input sufficient for generative reconstruction. Given $\mathbf{a}, \mathbf{b}$, and $\mathbf{W}$, the energy function [4] for a state $\{\mathbf{v}, \mathbf{h}\}$ of the RBM is defined as follows:

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{a}^T\mathbf{v} - \mathbf{b}^T\mathbf{h} - \mathbf{h}^T\mathbf{W}\mathbf{v}$$

For the paper cited above, we construct a set of the RBM's free parameters, $\Theta = \{\mathbf{W}, \mathbf{a}, \mathbf{b}\}$, for convenience. The probability of each pair of hidden and visible vectors, $\mathbf{v}, \mathbf{h}$ is defined using the energy function as:

$$p(\mathbf{v}, \mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z}$$

[1] http://yann.lecun.com/exdb/mnist/

where Z is the "partition function", which is more precisely defined as the sum of of the probabilities of all visible and hidden vector pairs:

$$Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}$$

and so the probability of a single visible vector $\mathbf{v}$ is given by:

$$p(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}$$

Given a set of observations, $D = \{v_1, v_2, \dots\}$, the RBM is trained by maximizing the probability of the visible vectors using the expected log-likelihood gradient over D:

$$\mathbb{E}_{\hat{P}}[\frac{\partial(logP(\mathbf{v}))}{\partial\Theta}] = -\mathbb{E}_{\hat{P}}[\frac{\partial\mathcal{F}(\mathbf{v})}{\partial\Theta}] + \mathbb{E}_{P}[\frac{\partial\mathcal{F}(\mathbf{v})}{\partial\Theta}]$$

where $\hat{P}$ and P represent the empirical and model distributions of D, respectively, and $\mathcal{F}(\mathbf{v})$ is the free energy of the visible vector $\mathbf{v}$, defined as:

$$\mathcal{F}(\mathbf{v}) = -\mathbf{a}^T\mathbf{v} - \sum_j \log(1 + \exp(b_j + \mathbf{W_j}\mathbf{v}))$$

More intuitively, the free energy of a visible vector is the energy that a single state configuration of $\mathbf{v}$ would need to have to have the same probability of the sum of the state configurations that contain $\mathbf{v}$. Therefore,

$$e^{-\mathcal{F}(\mathbf{v})} = \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}$$

2.2 Contrastive Divergence (CD)

The Contrastive Divergence (CD) algorithm [3] is used to approximate the expected log-likelihood gradient over D, which would otherwise be intractable. Moreover, CD is also easily adaptable to online settings. Using CD, the second term of the expected log-likelihood gradient is replaced by a single sample and the first term is approximated using a MCMC (Monte Carlo Markov Chain). That is, for a specified number of steps, starting from $\mathbf{v^0} \in D$, one then samples $\mathbf{h^0}$ using $P(\mathbf{h^0}|\mathbf{v^0})$, then samples $\mathbf{v^1}$ using $P(\mathbf{v^1}|\mathbf{h^0})$, and so on until $\mathbf{h}^{n_{CD}}$ is sampled (i.e., Gibbs sampling), where n_{CD} is the number of sampling steps. The conditional probabilities are defined as follows:

$$P(\mathbf{h} = 1|\mathbf{v}, \Theta) = \sigma(\mathbf{b} + \mathbf{W}\mathbf{v}) \text{ and } P(\mathbf{h} = 1|\mathbf{v}, \Theta) = \sigma(\mathbf{a} + \mathbf{W^T}\mathbf{h})$$

where σ is the sigmoid function.

The free parameters, Θ can then be updated via the following equations:

$$\begin{aligned}
\Delta W_{ji} &\propto v_i^0 h_j^0 - v_i^{n_{CD}} h_j^{n_{CD}} \quad \forall i \in \{1, \dots, n_v\}, \forall j \in \{1, \dots, n_h\} \\
\Delta a_i &\propto v_i^0 - v_i^{n_{CD}} \quad \forall i \in \{1, \dots, n_v\} \\
\Delta b_j &\propto h_j^0 - h_j^{n_{CD}} \quad \forall j \in \{1, \dots, n_h\}
\end{aligned}$$

2.3 Sequential learning algorithm (OCD_{GR})

The Online Contrastive Divergence with Generative Replay is a biologically-inspired algorithm that parallels the memory recall capabilities of the human brain. It sequentially learns an RBM and incorporates it to generate accurate representative samples of the 'already-experienced' data distribution in an online fashion via Gibbs sampling [1]. The primary benefit of the use of this algorithm in relation to ER is it's reduction in space complexity since the dataset does not need to be stored in memory.

We use the notation described in [6] for the following description of OCD_{GR}. The algorithm performs online batch update to its parameters. The batch size before each update (between time $t-1$ and t of the algorithm) is adjustable using the $\mathcal{B}_t$ hyper-parameter. $\hat{\mathcal{B}}_t$ is the set of data samples generated (equivalent to ER recall for

experience retention) by the RBM using the free parameters Θ_{t-1}. OCD_{GR}, unlike ER, has the Markov property that $P(\Theta_t)$ depends only on Θ_{t-1} and B_t.

The OCD_{GR} introduced two additional (to classical offline CD) meta-parameters. Those are: the number of Gibbs sampling steps, n_{Gs}, performed to generate new data samples (recall) using previous parameters of the RBM's; and $n_{\hat{\mathcal{B}}}$, the number of samples generated by the RBM in each update iteration of the algorithm. The other parameters of the algorithm are common to all RBMs: the number of visible units, n_v, the number of hidden units, n_v, the number of Contrastive Divergence steps, n_{CD}, the number of training epochs, n_E, the learning rate, α, the size of the mini-batch obtained online from the data set, n_B, the momentum ρ, and weight decay ξ.

A copy of the pseudo-code of the algorithm provided by the authors in [6] is included in Figure 1. The primary components of the algorithm are the dynamic generation of historical data using the RBM and Gibbs sampling (lines 14-22) and the CD update with generative replay (lines 26-41), where CD is computed using the newly generated and recently received data (i.e. $\hat{\mathcal{B}}_t$ and $\mathcal{B}_t$).

3 Multiplicative Weights

This section gives a brief summary of Multiplicative Weights (MW) algorithm as introduced in class for completeness (lectures 5, 13, and 14). Randomized MW algorithm can be used for sequential prediction and can achieve a sub-linear regret on arbitrary sequences. The algorithm keeps track of predictors as follows for binary prediction:

$$w_{t,k} = e^{-\eta L_{t-1,k}}, k \in \{0,1\}$$

and

$$Prob(x_t = 1) = \frac{w_{t,1}}{w_{t,1} + w_{t,0}}$$

where $L_{t-1,k}$ is hamming loss given the set of observations (x_1, x_2, ..., x_{t-1}) for each of the two constant binary predictors (i.e., constant ones, or constant zeros). Optimal sublinear regret bound is achieved for learning rate $\eta = \frac{1}{\sqrt{T}}$ where T is sequence length. MW algorithm can be used for prediction given a set of past observations (predictors with memory) by extending the set of predictors (experts) and tracking each given a specific context (ex: a predictor given last three observations were the sequence (0, 1, 1)). In the case where predictors of variable memory length (all predictors of at most 3 memory cells) are considered together, predictors with larger memory should be penalized in proportion to their expected overfit; hence the penalty $2^D ln(2)$ where D is memory length.

4 Sequence prediction

In sequential learning for sequence prediction, the goal is to predict next value in the sequence (x_t) well while estimating the parameters of a prediction algorithm online given past observations ($x_1, x_2, \ldots, x_{t-1}$). In randomized binary prediction, this takes the form of sampling from $Prob(x_t = 1|\phi_t)$ where ϕ_t is estimated from X=(x_1, x_2, ..., x_{t-1}). Similarly, the prediction can be performed given past N observations (N-long memory predictor) using $Prob(x_t = 1|(x_{t-N}, \ldots, x_{t-1}), \phi_t)$.

In this section, we use sequentially learned RBM as a compact way to represent and track an approximate distribution of the input sequence ($X_{1\ldots T}$), which is then used for prediction. The section proceeds by showing (for the basic case of zero-memory prediction) that a special case of RBM have an equivalent prediction rule to that of MW algorithm. The section then presents sequence prediction results for both RBM and MW (with and without memory).

4.1 RBMs for sequence prediction

RBMs model the probability function $Prob(v, h|\phi_t)$ where $v \in \{0,1\}^{n_v}$ is the visible random variable (input), $h \in \{0,1\}^{n_h}$ is a hidden random variable (latent representation), and ϕ_t is the set of learned parameters by time step t. In a no-memory case single bit prediction case ($n_v = 1$), to predict the next value in the sequence (i.e., $\hat{x}_t$), prediction is sampled from:

$$Prob(v = 1|\phi_t = \{a_t, b_t, W_t\}) = \sum_h Prob(v = 1, h|\phi_t = \{a_t, b_t, W_t\}) = \frac{\sum_h e^{a_t + h^T b_t + h^T W_t}}{\sum_{h,v} e^{v a_t + h^T b_t + h^T W_t v}}$$

%% *Initialization of the various parameters*
Set $n_h, n_v, n_{Gs}, n_{CD}, n_E, n_B, n_{\hat{B}}, \alpha, \rho, \xi$
Initialize RBM parameters Θ_0(i.e., $\mathbf{W}_0, \mathbf{a}_0, \mathbf{b}_0$)$\sim \mathcal{N}(0, \sigma)$
Set $\Delta\Theta_0^{n_E} = 0$
Set $t = 1, \mathbf{B}_t = \varnothing$
%% *A continuous loop to handle sequential incoming data*
while *system is running* **do**
    Observe a new data point $\mathbf{d}$
    Add $\mathbf{d}$ to $\mathbf{B}_t$
    if $\mathbf{B}_t$ *contains* n_B *observed data points* **then**
        Set $\hat{\mathbf{B}}_t = \varnothing$
        %% *Generate new data points with the RBM*
        if $t > 1$ **then**
            for $i = 1 : n_{\hat{B}}$ **do**
                %% *Run Gibbs sampling*
                Initialize $\mathbf{h} \sim \mathcal{U}(0, 1)$
                for $k = 1 : n_{Gs}$ **do**
                    Infer $P(\mathbf{v} = 1|\mathbf{h}, \Theta_{t-1})$
                    Infer $P(\mathbf{h} = 1|\mathbf{v}, \Theta_{t-1})$
                end
                Add $\mathbf{v}$ to $\hat{\mathbf{B}}_t$
            end
        end
        %% *Update parameters*
        Set $\Theta_t^0 = \Theta_{t-1}$ and $\Delta\Theta_t^0 = \Delta\Theta_{t-1}^{n_E}$
        for $e(epoch) = 1 : n_E$ **do**
            %% *Create a training batch from* $\mathbf{B}_t$ *and* $\hat{\mathbf{B}}_t$
            Set $\mathbf{V} = \mathbf{B}_t \cup \hat{\mathbf{B}}_t$
            Infer $P(\mathbf{H} = 1|\mathbf{V}, \Theta_t^{e-1})$
            %% *Collect positive statistics* Ψ^+
            Compute Ψ^+ from $\mathbf{V}$ and $\mathbf{H}$
            for $k = 1 : n_{n_{CD}}$ **do**
                Infer $P(\mathbf{V} = 1|\mathbf{H}, \Theta_t^{e-1})$
                Infer $P(\mathbf{H} = 1|\mathbf{V}, \Theta_t^{e-1})$
            end
            %% *Collect negative statistics* Ψ^-
            Compute Ψ^- from $\mathbf{V}$ and $\mathbf{H}$
            %% *Perform parameters update*
            $\Delta\Theta_t^e = \rho\Delta\Theta_t^{e-1} + \alpha[(\Psi^+ - \Psi^-)/(n_B + n_{\hat{B}}) - \xi\mathbf{\Theta}_t^{e-1}]$
            $\Theta_t^e = \Theta_t^{e-1} + \Delta\Theta_t^e$
        end
        Set $\Theta_t = \Theta_t^{n_E}$
        %% *Clean the memory*
        Delete $\hat{\mathbf{B}}_t, \mathbf{B}_t$ from memory
        %% *Advance to the next time step*
        Set $t = t + 1, \mathbf{B}_t = \varnothing$
    end
end

Figure 1: Online Contrastive Divergence with Generative Replay [6]

where the parameters $\phi_t = \{a_t, b_t, W_t\}$ are learned by feeding observations X=(x_1, x_2, …, x_{t-1}) sequentially into the OCD_{GR} algorithm.

In the case, of prediction given the N last observations (i.e., the set ob_t = (x_{t-N}, …, x_{t-1})), $\hat{x}_t$ is sampled

from:

$$Prob(v_{n_v} = 1|(v_1, \ldots, v_{n_v-1}) = ob_t, \phi_t) = \frac{\sum_h Prob(v_{n_v} = 1, (v_1, \ldots, v_{n_v-1}) = ob_t, h|\phi_t)}{\sum_{h,v_{n_v}} Prob(v_{n_v}, (v_1, \ldots, v_{n_v-1}) = ob_t, h|\phi_t)}$$

where the parameter set $\phi_t = \{a_t, b_t, W_t\}$ is learned sequentially using OCD_{GR} by feeding the vector $v = (x_{t-N-1}, \ldots, x_{t-1})$. In this case, the algorithm learns an approximate distribution of sequences of N+1 of consecutive observations.

4.2 RBMs as an alternative for MW

This section focuses on the basic case of sequential prediction with no memory where $n_v = 1$ and we assume as well that $n_h = 1$. MW prediction rule is equivalent to a trivial RBM where the the visible and hidden layers are not connected (i.e., $W = 0$). In this case, the RBM model can be written as follows:

$$\begin{aligned} Prob(v) &= \frac{\sum_{h\in\{0,1\}} e^{av+bh+wvh}}{\sum_{h\in\{0,1\},\bar{v}\in\{0,1\}} e^{a\bar{v}+bh+w\bar{v}h}} = \frac{\sum_{h\in\{0,1\}} e^{av+bh}}{\sum_{h\in\{0,1\},\bar{v}\in\{0,1\}} e^{a\bar{v}+bh}} \\ &= \frac{e^{av} + e^{av+b}}{1 + e^b + e^a + e^{a+b}} = \frac{e^{av}}{1 + e^a} \end{aligned}$$

The parameters of this RBM can be estimated by optimizing its log likelihood (where v_i is the set of observations):

$$log(\ell(\phi; v)) = \sum_{i=1}^{t} [av_i - log(1 + e^a)]$$

By taking the derivative and solving for the stationary points (where $\bar{v}^{(1\ldots t)}$ is sample mean for data received until time t), we get:

$$\frac{\partial}{\partial a} (log(\ell(\phi; v))) = \sum_{i=1}^{t} \left(v_i - \frac{1}{1 + e^a} e^a \right) = 0 \implies a = -log(\frac{1}{\bar{v}^{(1\ldots t)}} - 1)$$

By substituting this back into the prediction rule, we get:

$$Prob(v = 1) = \frac{e^{log(\bar{v}^{(1\ldots t)})}}{e^{log(\bar{v}^{(1\ldots t)})} + e^{log(1-\bar{v}^{(1\ldots t)})}}$$

which is equivalent to the MW prediction rule where learning rate $\eta = \frac{1}{t}$.

4.3 Results

This section presents sequence prediction results using RBM and using MW. The first section shows results for zero-memory length prediction, and the second shows results for N-long-memory prediction. Similar to MW, the results demonstrate the ability of RBM to learn an approximation for arbitrary sequence distributions online with empirically sublinear regret comparable to that of MW.

4.3.1 Zero-Memory Prediction

This section shows results for sequence prediction using RBM and using MW both predicting using no memory. Figure 2 shows results for stochastic sequences (Bernoulli), and Figure 3 shows results for deterministic sequences (constant, and alternating). RBM trained sequentially achieved comparable performance to MW on most sequences. MW algorithm outperformed RBM over the Bernoulli(0.7) and Bernoulli(0.2) sequences only, which is expected since RBM strictly models sequence distribution (probabilities converges to 0.7 and 0.2) as opposed to MW, which is designed to optimize expected reward (probabilities converges to 1 and 0).

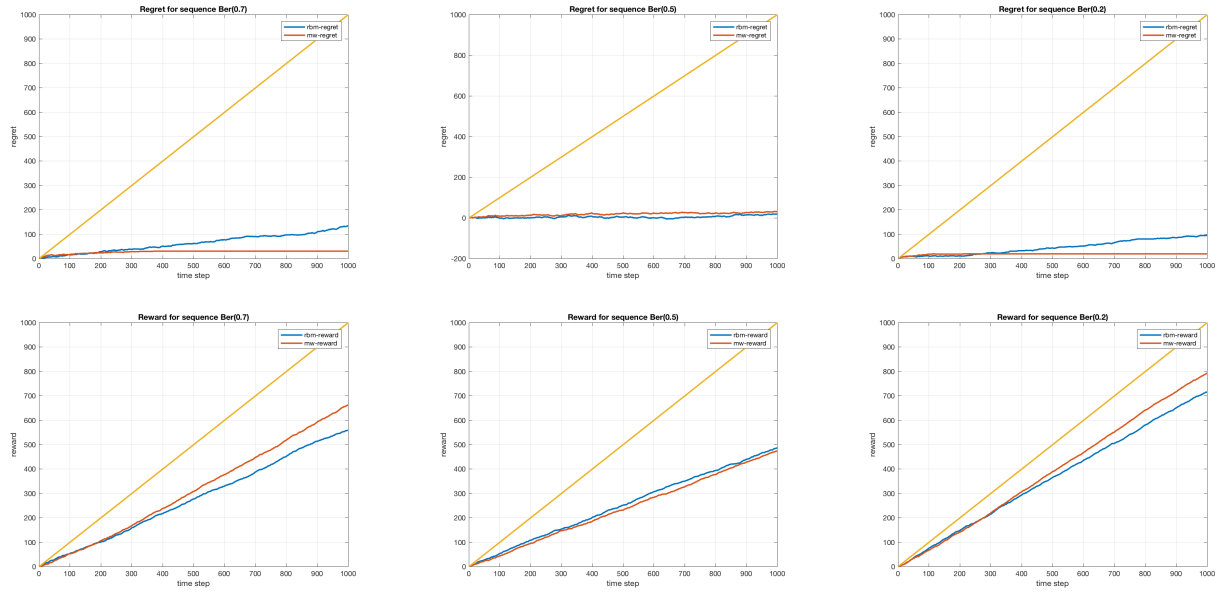

Figure 2: Regret and cumulative reward plots for no-memory randomized RBM and MW for the case of stochastic sequences, and stochastic time-varying sequences.

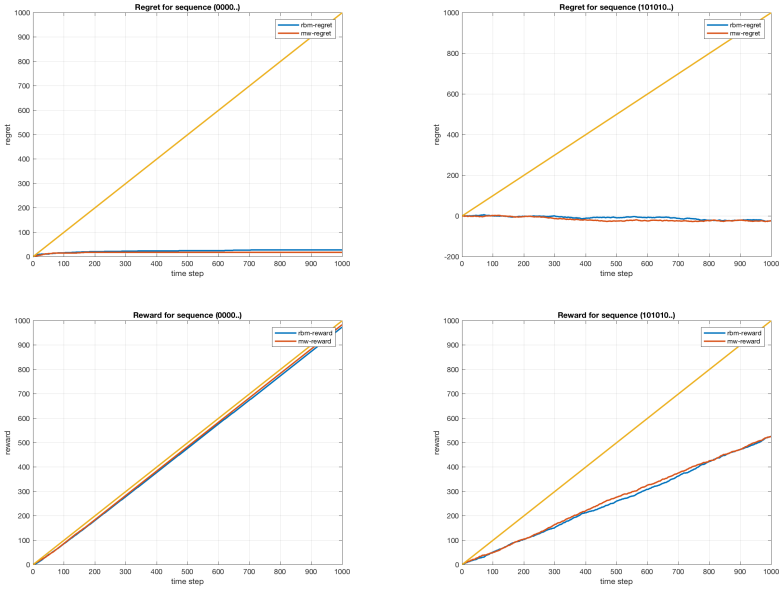

Figure 3: Regret and cumulative reward plots for no-memory randomized RBM and MW for the case of constant sequence and alternating sequence.

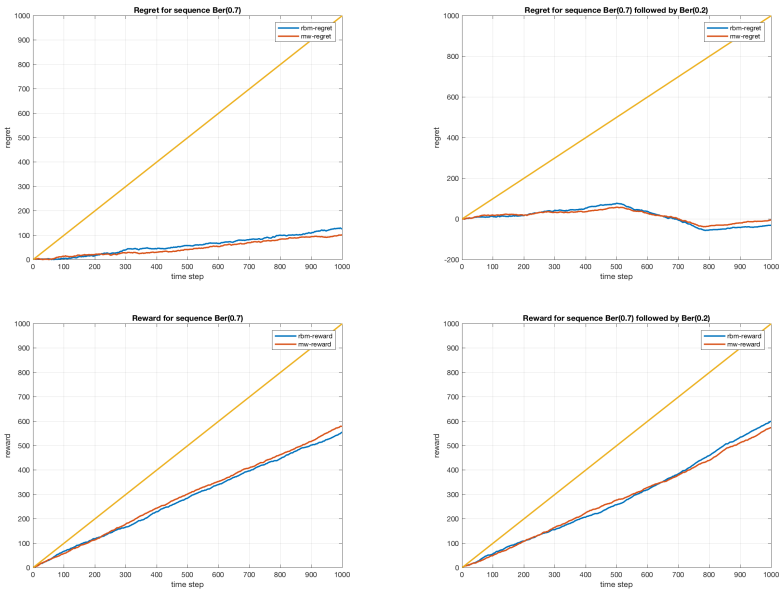

Figure 4: Regret and cumulative reward plots for 5-memory-cells randomized RBM and MW for the case of stochastic sequences, and stochastic time-varying sequences.

4.3.2 N-long-Memory Prediction

This section presents results for sequential prediction using algorithms that utilize N-long-memory (use last N observations as a context). We show results for both RBM and MW where N = 5 for both algorithms. Figure 4 show results for stochastic sequences. Here, we notice that RBM performs as well as MW on the stochastic sequence Ber(0.7), unlike the case presented for no-memory predictors. We speculate that this might be influenced by the fact that MW is susceptible to over-fitting despite being penalized for model complexity. On the other hand, RBM models probability, in which case, additional memory cells are expected to be ignored by independence property. Figure 5 shows results for randomized prediction over alternating deterministic sequences. The gap in performance here is again due to the fact that RBM models distribution of input. This gap disappears (discounting expected error during learning phase) under deterministic prediction scenarios shown in Figure 6.

4.4 Discussion

In this section, we used online RBM to approximate and track distribution of input sequences. We showed that for a trivial case RBM reduces to a prediction rule that is equivalent to that of MW algorithm. We then presented prediction results for both RBM and MW over the same set of sequences. RBM achieved performance comparable overall to that of MW over a variety of sequences both deterministic and stochastic. We noted that despite the overall performance, modeling the distribution of the input sequence is not sufficient to achieve optimal randomized prediction performance as observed in Figure 2. An optimal predictor in this sense should be able to model distribution over expected reward as does the MW algorithm, which converges to a state of certainty over the set of prediction choices. A future work can investigate the potential for using RBM to approximate this distribution over expected reward.

5 Sequence prediction from complex observation spaces

In this section, we fuse RBM and MW to perform sequence prediction from a complex observation space. The RBM, trained (unsupervised) using online OCD_{GR} compresses the observation space into an approximate compact representation (expected to be imperfect and noisy). The MW algorithm uses memory (from last observation) of this compact representation as a context to predict the next label. The true label is then provided to MW for training. Here, we show results for alternating (010101..) sequence prediction using the MNIST dataset as shown in Figure 7. Prediction results for this sequence is shown in Figure 8.

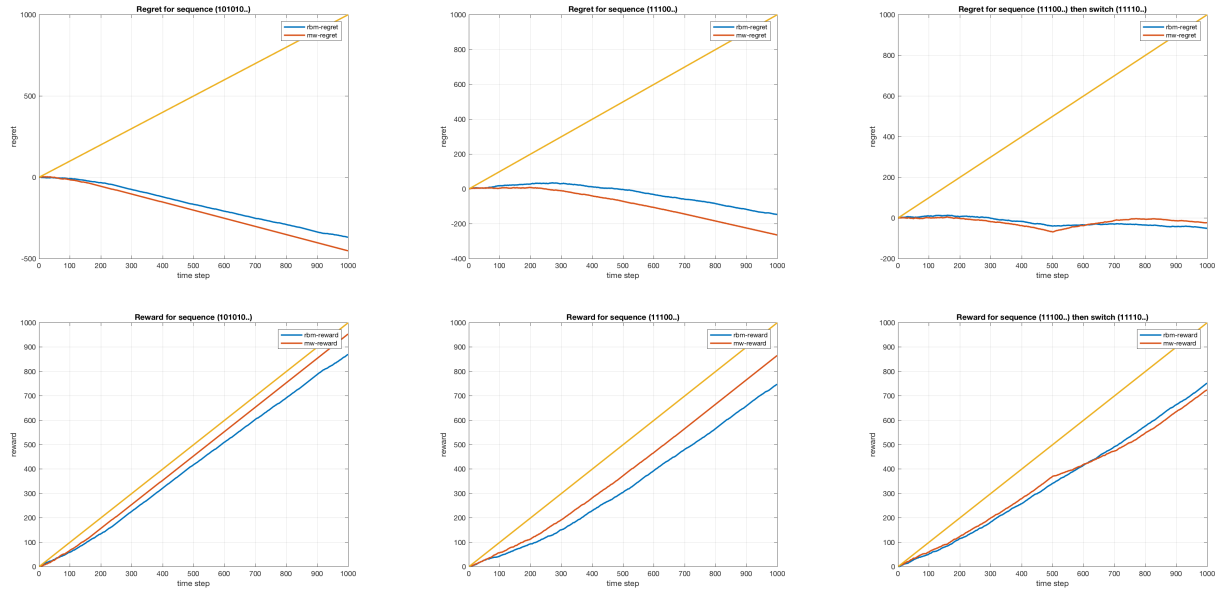

Figure 5: Regret and cumulative reward plots for 5-memory-cells randomized RBM and MW for the case of deterministic alternating sequences.

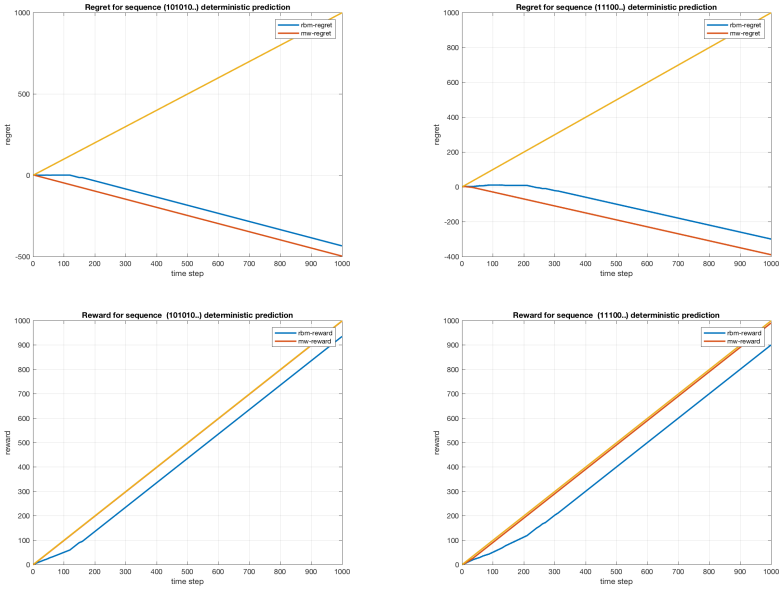

Figure 6: Regret and cumulative reward plots for 5-memory-cells deterministic RBM and MW prediction for the case of deterministic alternating sequences.

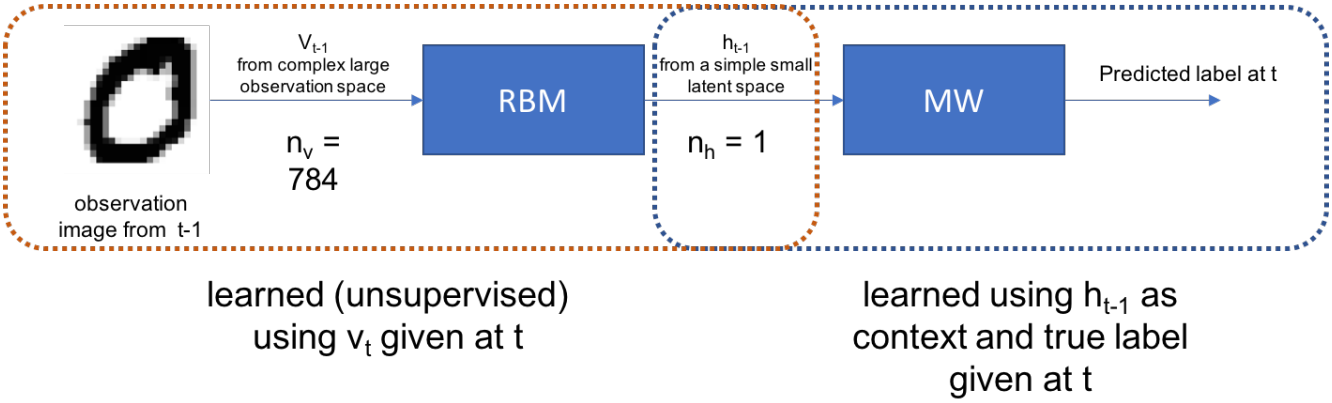

Figure 7: This figure shows the process for sequence prediction from complex observation spaces. A new image and true label is given at each time step from an alternating (010101..) sequence. The fused RBM-MW learns a compact representation, and then uses it as a context to learn the sequence for prediction purposes.

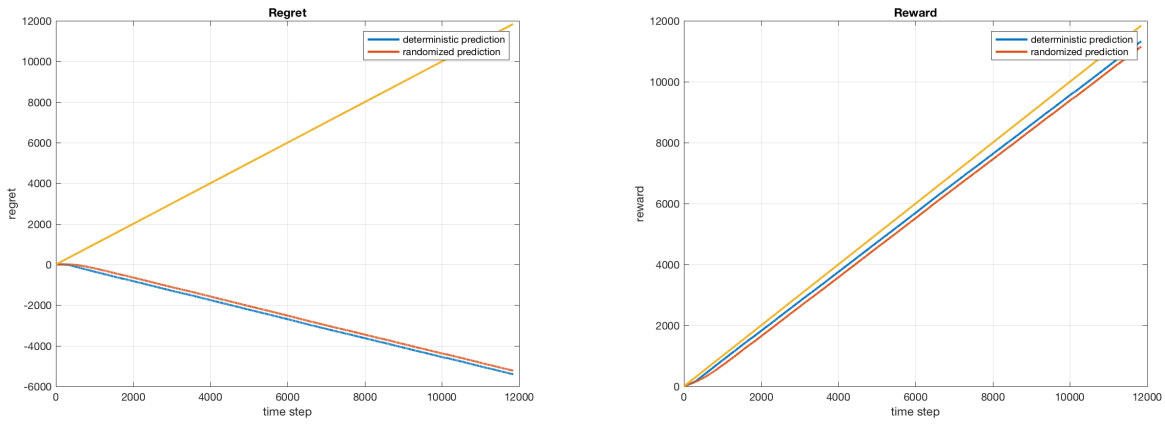

Figure 8: Regret and reward for randomized and deterministic sequence prediction using RBM-MW cascade for the deterministic sequence 01010.... RBM compresses last seen image into a 1 bit representation, MW then uses a memory of length 1 bit for prediction.

6 Conclusion

In our project, we demonstrated a comparable performance between RBMs (used as a distribution estimator and predictor in the context of OCD_{GR} algorithm) and MW algorithm as a benchmark for sequence prediction. We then demonstrated the use of RBMs to learn complex latent representations of online sequences sufficient for sequential prediction tasks using MW. An RBM learned online using OCD_{GR} was able to generate a compact representations (1 bit representation from 784 representation) of the MNIST dataset (0/1 digits received in an alternating sequence). The MW utilized the 1 bit representation as context to achieve near-optimal regret. These results, in addition to the promised generative capabilities of RBM render OCD_{GR} as a powerful tool for sequential tasks. In the future, it would be interesting to construct bounds on the expected regret that OCD_{GR} could achieve when integrated with an online sequence prediction algorithm such as MW. This would establish a theoretical justification for its use in online sequence prediction. Additionally, the algorithm should be empirically and theoretically tested for sensitivity to hyperparameters. This should pave the way for more experiments that utilize larger number of hidden units to enable the RBM to learn more complex representations of the input data for more complex sequential tasks.

References

[1] Y. Bengio. Learning deep architectures for ai. *Found Trends Mach Learn 2(1):1127, DOI 10.1561/2200000006*, 2009.

[2] Summerfield C Botvinick M Hassabis D, Kumaran D. Neuroscience-inspired artificial intelligence. *Neuron 95, 245 258.*, 2017.

[3] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation, 14(8):17111800*, 2002.

[4] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciencess, 79:25542558*, 1982.

[5] McNaughton BL McClelland JL. Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review 102:419-457*, 1995.

[6] Decebal Constantin Mocanu, Maria Torres Vega, Eric Eaton, Peter Stone, and Antonio Liotta. Online contrastive divergence with generative replay: Experience replay without storing data. *arXiv preprint arXiv:1610.05555*, 2016.

[7] Paul Smolenky. Information processing in dynamical systems: Foundations of harmony theory. *Parallel Distributed Processing, volume 1, chapter 6, pages 194281. MIT Press, Cambridge*, 1986.