Repeat before Forgetting: Spaced Repetition for Efficient and Effective Training of Neural Networks

Hadi Amiri, Timothy A. Miller, Guergana Savova
Boston Children's Hospital Informatics Program, Harvard Medical School
{firstname.lastname}@childrens.harvard.edu

Abstract

We present a novel approach for training artificial neural networks. Our approach is inspired by broad evidence in psychology that shows human learners can learn efficiently and effectively by increasing intervals of time between subsequent reviews of previously learned materials (spaced repetition). We investigate the analogy between training neural models and findings in psychology about human memory model and develop an efficient and effective algorithm to train neural models. The core part of our algorithm is a cognitively-motivated scheduler according to which training instances and their "reviews" are spaced over time. Our algorithm uses only 34-50% of data per epoch, is 2.9-4.8 times faster than standard training, and outperforms competing state-of-the-art baselines.[1]

1 Introduction

Deep neural models are known to be computationally expensive to train even with fast hardware (Sutskever et al., 2014; Wu et al., 2016). For example, it takes three weeks to train a deep neural machine translation system on 100 Graphics Processing Units (GPUs) (Wu et al., 2016). Furthermore, a large amount of data is usually required to train effective neural models (Goodfellow et al., 2016; Hirschberg and Manning, 2015).

Bengio et al. (2009) and Kumar et al. (2010) developed training paradigms which are inspired by the learning principle that humans can learn more effectively when training starts with *easier* concepts and gradually proceeds with more *difficult* concepts. Since these approaches are motivated by a "starting small" strategy they are called *curriculum* or *self-paced* learning.

In this paper, we present a novel training paradigm which is inspired by the broad evidence in psychology that shows human ability to retain information improves with repeated exposure and exponentially decays with delay since last exposure (Cepeda et al., 2006; Averell and Heathcote, 2011). Spaced repetition was presented in psychology (Dempster, 1989) and forms the building block of many educational devices, including flashcards, in which small pieces of information are repeatedly presented to a learner on a schedule determined by a spaced repetition algorithm. Such algorithms show that human learners can learn efficiently and effectively by increasing intervals of time between subsequent reviews of previously learned materials (Dempster, 1989; Novikoff et al., 2012).

We investigate the analogy between training neural models and findings in psychology about human memory model and develop a spaced repetition algorithm (named Repeat before Forgetting, RbF) to efficiently and effectively train neural models. The core part of our algorithm is a scheduler that ensures a given neural network spends more time working on difficult training instances and less time on easier ones. Our scheduler is inspired by factors that affect human memory retention, namely, *difficulty* of learning materials, *delay* since their last review, and *strength* of memory. The scheduler uses these factors to lengthen or shorten review intervals with respect to individual learners and training instances. We evaluate schedulers based on their scheduling accuracy, i.e., accuracy in estimating network memory retention with respect to previously-seen instances, as well as their effect on the efficiency and effectiveness of downstream neural networks.[2]

[1] Our code is available at scholar.harvard.edu/hadi/RbF/

[2] In this paper, we use the terms memory retention, recall, and learning interchangeably.

The contributions of this paper are: (1) we show that memory retention in neural networks is affected by the same (known) factors that affect memory retention in humans, (2) we present a novel training paradigm for neural networks based on spaced repetition, and (3) our approach can be applied without modification to any neural network.

Our best RbF algorithm uses 34-50% of training data per epoch while producing similar results to state-of-the-art systems on three tasks, namely sentiment classification, image categorization, and arithmetic addition.[3] It also runs 2.9-4.8 times faster than standard training, and outperforms competing state-of-the-art baselines.

2 Neural and Brain Memory Models

Research in psychology describes the following memory model for human learning: the probability that a human recalls a previously-seen item (e.g., the Korean translation of a given English word) depends on the *difficulty* of the item, *delay* since last review of the item, and the *strength* of the human memory. The relation between these indicators and memory retention has the following functional form (Reddy et al., 2016; Ebbinghaus, 1913):

$$\Pr(recall) = \exp(-\frac{difficulty \times delay}{strength}). \quad (1)$$

An accurate memory model enables estimating the time by which an item might be forgotten by a learner so that a review can be scheduled for the learner before that time.

We investigate the analogy between the above memory model and memory model of artificial neural networks. Our intuition is that if the probability that a network recalls an item (e.g., correctly predicts its category) depends on the same factors (difficulty of the item, delay since last review of the item, or strength of the network), then we can develop spaced repetition algorithms to efficiently and effectively train neural networks.

2.1 Recall Indicators

We design a set of preliminarily experiments to directly evaluate the effect of the aforementioned factors (recall indicators) on memory retention in neural networks. For this purpose, we use a set of training instances that are partially made available to the network during training. This scheme

[3]We obtained similar results on QA tasks (Weston et al., 2016) but they are excluded due to space limit.

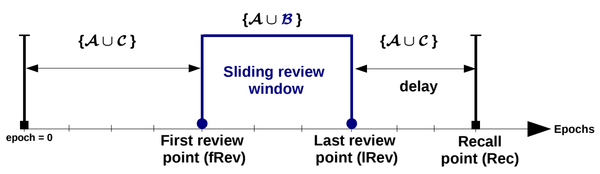


Figure 1: Effect of recall indicators on network retention. Training data is uniformly at random divided into three disjoint sets $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ that respectively contain 80%, 10%, and 10% of the data. Network retention is computed against set $\mathcal{B}$ instances at recall point.

will allow us to intrinsically examine the effect of recall indicators on memory retention in isolation from external effects such as size of training data, number of training epochs, etc.

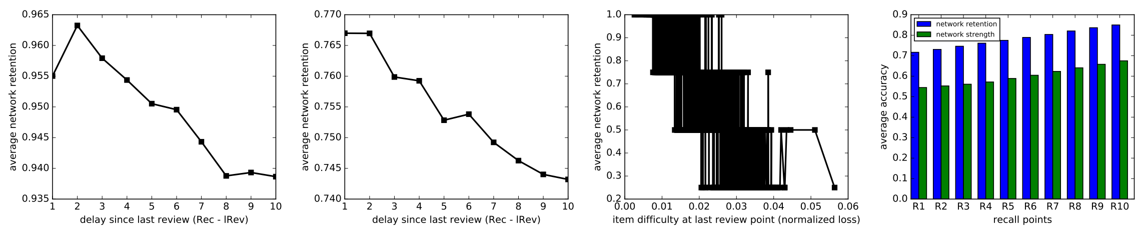
We first define the following concepts to ease understanding the experiments (see Figure 1):

- **First** and **Last review points (fRev and lRev)** of a training instance are the first and last epochs in which the instance is used to train the network respectively,

- **Recall point (Rec)** is the epoch in which *network retention* is computed against some training instances; network retention is the probability that a neural network recalls (i.e. correctly classifies) a previously-seen training instance, and

- **Delay since last review** of a training instance is the difference between the recall point and the last review point of the training instance.

Given training data and a neural network, we uniformly at random divide the data into three disjoint sets: a *base* set $\mathcal{A}$, a *review* set $\mathcal{B}$, and a *replacement* set $\mathcal{C}$ that respectively contain 80%, 10%, and 10% of the data. As depicted in Figure 1, instances of $\mathcal{A}$ are used for training at every epoch, while those in $\mathcal{B}$ and $\mathcal{C}$ are partially used for training. The network initially starts to train with $\{\mathcal{A} \cup \mathcal{C}\}$ instances. Then, starting from the first review point, we inject the review set $\mathcal{B}$ and remove $\mathcal{C}$, training with $\{\mathcal{A} \cup \mathcal{B}\}$ instances at every epoch until the last review point. The network will then continue training with $\{\mathcal{A} \cup \mathcal{C}\}$ instances until the recall point. At this point, network retention is computed against set $\mathcal{B}$ instances, with delay defined as the number of epochs since last review point. The intuition behind using review and replacement sets, $\mathcal{B}$ and $\mathcal{C}$ respectively, is to avoid external effects (e.g.



(a) Delay full (b) Delay partial (c) Item difficulty (d) Network strength

Figure 2: (a) Delay since last review vs. average network retention (accuracy) on set $\mathcal{B}$ instances at recall point. Recall point is fixed and set to the epoch in which networks obtain their best performance based on rote training. (b) The same as (a) except that recall point is set to the epoch in which networks obtain half of their best performance based on rote training. (c) Item Difficulty (normalized loss at last review point) vs. average network retention at recall point on set $\mathcal{B}$ instances. (d) Network strength (network accuracy on validation data at recall point) vs. average network retention at recall point on set $\mathcal{B}$ instances. Length of sliding window is fixed throughout experiments and set to 5 epochs.

size of data or network generalization and learning capability) for our intrinsic evaluation purpose.

To conduct these experiments, we identify different neural models designed for different tasks.[4] For each network, we fix the recall point to either the epoch in which the network is fully trained (i.e., obtains its best performance based on standard or "rote" training in which all instances are used for training at every iteration), or partially trained (i.e., obtains half of its best performance based on rote training). We report average results across these networks for each experiment.

2.1.1 Delay since Last Review

As aforementioned, delay since last review of a training instance is the difference between the recall point (Rec) and the last review point (lRev) of the training instance. We evaluate the effect of delay on network retention (against set $\mathcal{B}$ instances) by keeping the recall point fixed while moving the sliding window in Figure 1. Figures 2(a) and 2(b) show average network retention across networks for the fully and partially trained recall points respectively. The results show an inverse relationship between network retention and delay since last review in neural networks.

2.1.2 Item Difficulty

We define difficulty of training instances by the loss values generated by a network for the instances. Figure 2(c) shows the difficulty of set $\mathcal{B}$ instances at the last review point against average network retention on these instances at recall point. We normalize loss values to unit vectors (to make them comparable across networks) and then average them across networks for both fully and partially trained recall points. As the results show, network retention decreases as item difficulty increases.

[4]See section 4, we use Addition and CIFAR10 datasets and their corresponding neural networks for these experiments.

2.1.3 Network Strength

We define strength of a network by its performance on validation data. To understand the effect of network strength on its retention, we use the same experimental setup as before except that we keep the delay (difference between recall point and last review point) fixed while gradually increasing the recall point; this will make the networks stronger by training them for more epochs. Then, at every recall point, we record network retention on set $\mathcal{B}$ instances and network accuracy on validation data. Average results across networks for two sets of 10 consecutive recall points (before fully and partially trained recall points) are shown in Figure 2(d). As the results show, network retention increases as memory strength increases.

The above experiments show that memory retention in neural networks is affected by the same factors that affect memory retention in humans: (a) neural networks forget training examples after a certain period of intervening training data (b): the period of recall is shorter for more difficult examples, and (c): recall improves as networks achieve better overall performance. We conclude that delay since last review, item difficulty (loss values of training instances), and memory strength (network performance on validation data) are key indicators that affect network retention and propose to design spaced repetition algorithms that take such indicators into account in training neural networks.

Algorithm 1. Leitner System

Input: $\mathbf{H}$: training data, $\mathbf{V}$: validation data, k : number of iterations, n : number of queues
Output: trained model

```
0  $Q = [q_0, q_1, \ldots, q_{n-1}]$
1  $q_0 = [\mathbf{H}]$, $q_i = []$ for $i$ in $[1, n-1]$
2  For $epoch = 1$ to $k$:
3    $current\_batch = []$
4    For $i = 0$ to $n-1$:
5      If $epoch\%2^i == 0$:
6        $current\_batch = current\_batch + q_i$
7    End For
8    $pmos, dmos, model = \mathbf{train}(current\_batch, \mathbf{V})$
9    $\mathbf{update\_queue}(Q, pmos, dmos)$
10 End for
11 return $model$
```

q_0	epochs = $\{1, 2, 3, 4, 5, \ldots\}$
q_1	epochs = $\{2, 4, 6, 8, 10, \ldots\}$
q_2	epochs = $\{4, 8, 12, 16, 20, \ldots\}$
$\ldots$	

Figure 3: Leitner System. The $\mathbf{train}(.)$ function trains the network for one epoch using instances in the $current\_batch$, and the $\mathbf{update\_queue}(.)$ function promotes the recalled (correctly classified) instances, $pmos$, to the next queue and demotes the forgotten ones, $dmos$, to q_0.

3 Spaced Repetition

We present two spaced repetition-based algorithms: a modified version of the Leitner system developed in (Reddy et al., 2016) and our Repeat before Forgetting (RbF) model respectively.

3.1 Leitner System

Suppose we have n queues $\{q_0, q_1, \ldots, q_{n-1}\}$. The Leitner system initially places all training instances in the first queue, q_0. As Algorithm 1 shows, at each training iteration, the Leitner scheduler chooses some queues to train a downstream neural network. Only instances in the selected queues will be used for training the network. During training, if an instance from q_i is recalled (e.g. correctly classified) by the network, the instance will be "promoted" to q_{i+1}, otherwise it will be "demoted" to the first queue, q_0.[5]

The Leitner scheduler reviews instances of q_i at every 2^i iterations. Therefore, instance in lower queues (difficult/forgotten instances) are reviewed more frequently than those in higher queues (easy/recalled ones). Figure 3 (bottom) provides examples of queues and their processing epochs. Note that the overhead imposed on training by the Leitner system is $O(|current\_batch|)$ at every epoch for moving instances between queues.

[5] Note that in (Reddy et al., 2016) demoted instances are moved to q_{i-1}. We observed significant improvement in Leitner system by moving such instances to q_0 instead of q_{i-1}.

3.2 RbF Model

3.2.1 RbF Memory Models

The challenge in developing memory models is to estimate the time by which a training instance should be reviewed before it is forgotten by the network. Accurate estimation of the review time leads to efficient and effective training. However, a heuristic scheduler such as Leitner system is sub-optimal as its hard review schedules (i.e. only 2^i-iteration delays) may lead to early or late reviews.

We develop flexible schedulers that take recall indicators into account in the scheduling process. Our schedulers lengthen or shorten inter-repetition intervals with respect to individual training instances. In particular, we propose using density kernel functions to estimate the latest epoch in which a given training instance can be recalled. We aim to investigate how much improvement (in terms of efficiency and effectiveness) can be achieved using more flexible schedulers that utilize the recall indicators.

We propose considering density kernels as schedulers that favor (i.e., more confidently delay) less difficult training instances in stronger networks. As a kernel we can use any non-increasing function of the following quantity:

$$x_i = \frac{d_i \times t_i}{s_e}, \tag{2}$$

where d_i indicates the loss of network for a training instance $h_i \in \mathbf{H}$, t_i indicates the number of epochs to next review of h_i, and s_e indicates the performance of network— on validation data— at epoch e. We investigate the Gaussian, Laplace, Linear, Cosine, Quadratic, and Secant kernels as described below respectively:

$$f_{gau}(x, \tau) = \exp(-\tau x^2), \tag{3}$$

$$f_{lap}(x, \tau) = \exp(-\tau x), \tag{4}$$

$$f_{lin}(x, \tau) = \begin{cases} 1 - \tau x & x < \frac{1}{\tau} \\ 0 & \text{otherwise} \end{cases}, \tag{5}$$

$$f_{cos}(x, \tau) = \begin{cases} \frac{1}{2}\cos(\tau \pi x) + 1 & x < \frac{1}{\tau} \\ 0 & \text{otherwise} \end{cases}, \tag{6}$$

$$f_{qua}(x, \tau) = \begin{cases} 1 - \tau x^2 & x^2 < \frac{1}{\tau} \\ 0 & \text{otherwise} \end{cases}, \tag{7}$$

$$f_{sec}(x, \tau) = \frac{2}{\exp(-\tau x^2) + \exp(\tau x^2)}, \tag{8}$$

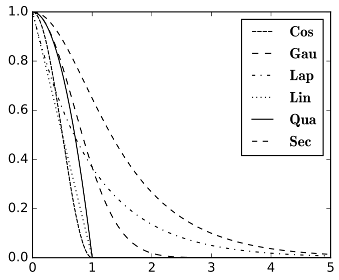


Figure 4: RbF kernel functions with $\tau = 1$.

where τ is a learning parameter. Figure 4 depicts these kernels with $\tau = 1$. As we will discuss in the next section, we use these kernels to optimize delay with respect to item difficulty and network strength for each training instance.

3.2.2 RbF Algorithm

Our Repeat before Forgetting (RbF) model is a spaced repetition algorithm that takes into account the previously validated recall indicators to train neural networks, see Algorithm 2. RbF divides training instances into *current* and *delayed* batches based on their delay values at each iteration. Instances in the current batch are those that RbF is less confident about their recall and therefore are reviewed (used to re-train the network) at current iteration. On the other hand, instances in the delayed batch are those that are likely to be recalled by the network in the future and therefore are not reviewed at current epoch. At each iteration, the RbF scheduler estimates the optimum delay (number of epochs to next review) for each training instance in the current batch. RbF makes such item-specific estimations as follows:

Given the difficulty of a training instance d_i, the memory strength of the neural network at epoch e, s_e, and an RbF memory model f (see section 3.2.1), RbF scheduler estimates the maximum delay $\hat{t}_i$ for the instance such that it can be recalled with a confidence greater than the given threshold $\eta \in (0, 1)$ at time $e + \hat{t}_i$. As described before, d_i and s_e can be represented by the current loss of the network for the instance and the current performance of the network on validation data respectively. Therefore, the maximum delay between the current (epoch e) and next reviews of the instance can be estimated as follows:

$$\hat{t}_i = \arg\min_{t_i} \big(f(x_i, \hat{\tau}) - \eta\big)^2, \qquad (9)$$
$$s.t \quad 1 \le t_i \le k - e$$

Algorithm 2. RbF Training Model

Input: $\mathbf{H}$: training data, $\mathbf{V}$: validation data, k : number of iterations, f : RbF kernel, η: recall confidence
Output: trained model

```
0  $t_i = 1$ for $h_i \in \mathbf{H}$
1  For $epoch = 1$ to $k$:
2    $current\_batch = \{h_i : t_i <= 1\}$
3    $delayed\_batch = \{h_i : t_i > 1\}$
4    $s_{epoch}, model = \mathbf{train}(current\_batch, \mathbf{V})$
5    $\hat{\tau} = \arg\min_{\tau} \big(f(x_i, \tau) - a_i\big)^2 \quad \forall h_i \in \mathbf{V}, a_i \ge \eta$
6    $\hat{t_i} = \arg\min_{t_i} \big(f(x_i, \hat{\tau}) - \eta\big)^2 \forall h_i \in current\_batch$
7    $t_i = t_i - 1 \quad \forall h_i \in delayed\_bach$
8  End for
9  return $model$
```

Figure 5: RbF training model. The $\mathbf{train}(.)$ function at line 5 trains the network for one epoch using instances in the $current\_batch$. Note that at each iteration *epoch*, x_i is computed using Equation (2) and strength of the current model, s_{epoch}.

where $\hat{\tau}$ is the optimum value for the learning parameter obtained from validation data, see Equation (10). In principle, reviewing instances could be delayed for any number of epochs; in practice however, delay is bounded both below and above (e.g., by queues in the Leitner system). Thus, we assume that, at each epoch e, instances could be delayed for at least one iteration and at most $k - e$ iterations where k is the total number of training epochs. We also note that t_i is a lower bound of the maximum delay as s_e is expected to increase and d_i is expected to decrease as the network trains in next iterations.

Algorithm 2 shows the outline of the proposed RbF model. We estimate the optimum value of τ (line 5 of Algorithm 2) for RbF memory models using validation data. In particular, RbF uses the loss values of validation instances and strength of the network obtained at the previous epoch to estimate network retention for validation instances at the current epoch (therefore $t_i = 1$ for every validation instance). The parameter τ for each memory model is computed as follows:

$$\hat{\tau} = \arg\min_{\tau} \big(f(x_j, \tau) - a_j\big)^2, \forall h_j \in \mathbf{V}, a_j \ge \eta, \qquad (10)$$

where $a_j \in (0, 1)$ is the current accuracy of the model for the validation instance h_j. RbF then predicts the delay for current batch instances and reduces the delay for those in the delayed batch by one epoch. The overhead of RbF is $O(|\mathbf{H}|)$ to compute delays and $O(|\mathbf{V}|)$ to compute $\hat{\tau}$. Note that (9) and (10) have closed form solutions.

Dataset	train, dev, test	Network	Task
IMDb	20K, 5K, 25K	MLP/fastext (Joulin et al., 2017), best epoch=8	sentiment analysis
CIFAR10	45K, 5K, 10K	CNN (Chan et al., 2015) best epoch=64	image classification
Addition	40K, 5K, 10K	LSTM (Sutskever et al., 2014) best epoch=32	arithmetic addition

Table 1: Datasets, models, and tasks.

4 Experiments

Table 1 describes the tasks, datasets, and models that we consider in our experiments. It also reports the training epochs for which the models produce their best performance on validation data (based on rote training). We note that the Addition dataset is randomly generated and contains numbers with at most 4 digits.[6]

We consider three schedulers as baselines: a slightly modified version of the Leitner scheduler (Lit) developed in Reddy et al. (2016) for human learners (see Footnote 5), curriculum learning (CL) in which training instances are scheduled with respect to their *easiness* (Jiang et al., 2015), and the uniform scheduler of rote training (Rote) in which all instances are used for training at every epoch. For Lit, we experimented with different queue lengths, $n = \{3, 5, 7\}$, and set $n = 5$ in the experiments as this value led to the best performance of this scheduler across all datasets.

Curriculum learning starts training with *easy* instances and gradually introduces more complex instances for training. Since easiness information is not readily available in most datasets, previous approaches have used heuristic techniques (Spitkovsky et al., 2010; Basu and Christensen, 2013) or optimization algorithms (Jiang et al., 2015, 2014) to quantify easiness of training instances. These approaches consider an instance as easy if its loss is smaller than a threshold (λ). We adopt this technique as follows: at each iteration e, we divide the entire training data into *easy* and *hard* sets using iteration-specific λ_e and the loss values of instances, obtained from the current partially-trained network. All easy instances in conjunction with $\alpha_e \in [0, 1]$ fraction of easiest hard instances (those with smallest loss values greater than λ_e) are used for training at iteration e. We set each λ_e to the average loss of training instances that are correctly classified by the current partially-trained network. Furthermore, at each iteration e, we set $\alpha_e = e/k$ to gradually introduce complex instances at every new iteration.[7] Note that we treat all instances as easy at $e = 0$.

[6] https://github.com/fchollet/keras/blob/master/examples/addition_rnn.py

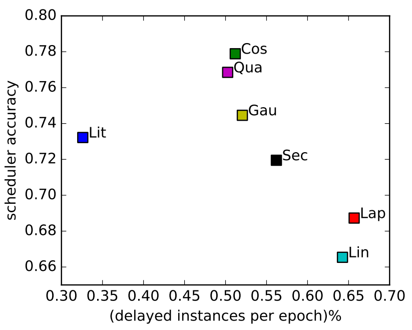


Figure 6: Accuracy of schedulers in predicting network retention. For these experiments recall confidence is set to its default value, $\eta = 0.5$.

Performance values reported in experiments are averaged over 10 runs of systems and the confidence parameter η is always set to 0.5 unless otherwise stated.

4.1 Evaluation of Memory Models

In these experiments, we evaluate memory schedulers with respect to their accuracy in predicting network retention for delayed instances. Since curriculum learning does not estimate delay for training instances, we only consider Leitner and RbF schedulers in these experiments.

For this evaluation, if a scheduler predicts a delay t for a training instance h at epoch e, we evaluate network retention with respect to h at epoch $e + t$. If the network recalls (correctly classifies) the instance at epoch $e + t$, the scheduler has correctly predicted network retention for h, and otherwise, it has made a wrong prediction. We use this binary outcome to evaluate the accuracy of each scheduler. Note that the performance of schedulers on instances that have not been delayed is not a major concern. Although failing to delay an item inversely affects efficiency, it makes the network stronger by providing more instances to train from. Therefore, we consider a good scheduler as the one that accurately delays more items.

Figure 6 depicts the average accuracy of schedulers in predicting networks' retention versus the average fraction of training instances that they delayed per epoch. As the results show, all schedulers

[7] k is the total number of iterations.

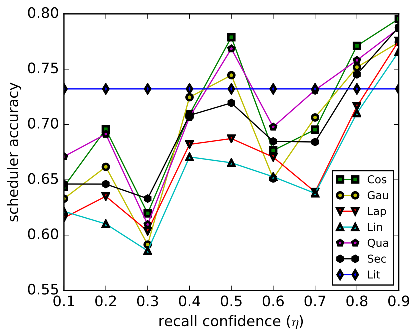


Figure 7: Effect of recall confidence η on the accuracy of different schedulers in predicting network retention (best seen in color.)

delay substantial amount of instances per epoch. In particular, Cos and Qua outperform Lit in both predicting network retention and delaying items, delaying around 50% of training instances per epoch. This is while Gau and Sec show comparable accuracy to Lit but delay more instances. On the other hand, Lap, which has been found effective in Psychology, and Lin are less accurate in predicting network retention. This is because of the tradeoff between delaying more instances and creating stronger networks. Since these schedulers are more flexible in delaying greater amount of instances, they might not provide networks with enough data to fully train.

Figure 7 shows the performance of RbF schedulers with respect to the recall confidence parameter η, see Equation (9). As the results show, schedulers have poor performance with smaller values of η. This is because smaller values of η make schedulers very flexible in delaying instances. However, the performance of schedulers are not dramatically low even with very small ηs. Our further analyses on the delay patterns show that although a smaller η leads to more delayed instances, the delays are significantly shorter. Therefore, most delayed instances will be "reviewed" shortly in next epochs. These bulk reviews make the network stronger and help it to recall most delayed instance in future iterations.

On the other hand, greater ηs lead to more accurate schedulers at the cost of using more training data. In fact, we found that larger ηs do not delay most training instances in the first few iterations. However, once the network obtains a reasonably high performance, schedulers start delaying instances for longer durations. We will further study this effect in the next section.

Model	Accuracy	TIPE	X Faster	Gain
CL	0.868	0.71	0.93	1.40
Lit	0.859	0.67	2.87	1.85
Gau	0.874	0.48	3.02	2.16
Lap	0.857	**0.34**	4.66	**3.15**
Lin	0.864	0.36	**4.78**	3.07
Cos	0.868	0.49	2.90	2.10
Qua	0.871	0.50	2.95	2.08
Sec	0.866	0.44	3.09	2.33
Cos $\eta = 0.9$	**0.880**	0.76	2.36	1.42
Rote	0.887	1.00	1.00	1.00

Table 2: Comparison of schedulers in terms of average network accuracy, average fraction of instances used for training per epoch (TIPE), and the extent to which a model runs faster than Rote training (X Times Faster). Gain column indicates the $\frac{Accuracy}{TIPE}$ improvement over rote training.

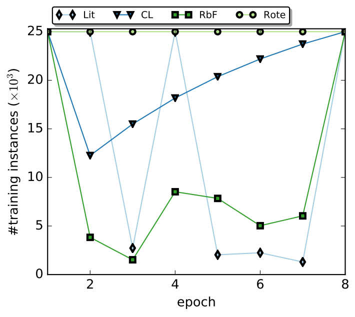
4.2 Efficiency and Effectiveness

We compare RbF against Leitner and curriculum learning in terms of efficiency of training and effectiveness of trained models. We define effectiveness as the accuracy of a trained network on balanced test data, and efficiency as (a): fraction of instances used for training per epoch, and (b): required time for training the networks. For RbF schedulers, we set η to 0.5 and consider the best performing kernel Cosine with $\eta = 0.9$ based on results in Figure 7.

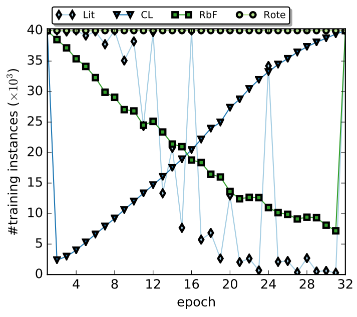
The results in Table 2 show that all training paradigms have comparable effectiveness (Accuracy) to that of rote training (Rote). Our RbF schedulers use less data per epoch (34-50% of data) and run considerably faster than Rote (2.90-4.78 times faster for $\eta = 0.5$). The results also show that Lit is slightly less accurate but runs 2.87 time faster than Rote; note that, as a scheduler, Lit is less accurate than RbF models, see Figures 6 and 7.

In addition, CL leads to comparable performance to RbF but is considerably slower than other schedulers. This is because this scheduler has to identify easier instances and sort the harder ones to sample training data at each iteration. Overall, the performance of Lit, CL, Cos $\eta = .5$ and Cos $\eta = .9$ are only 2.76, 1.90, 1.88, and 0.67 absolute values lower than that of Rote respectively. Considering the achieved efficiency, these differences are negligible (see the overall gain in Table 2).

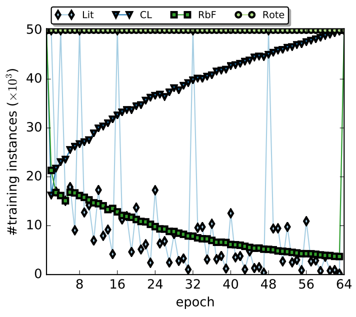
Figure 8 reports detailed efficiency and effectiveness results across datasets and networks. For clear illustration, we report accuracy at iterations 2^i $\forall i$ in which Lit is trained on the entire data, and consider Cos $\eta = .5$ as RbF scheduler. In terms of efficiency (first row of Figure 8), CL starts with (small set of)



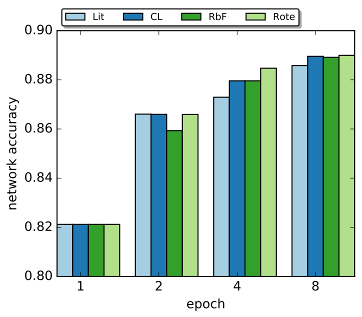
(a) Efficiency IMDb



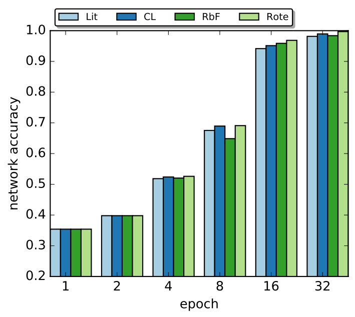
(b) Efficiency Addition



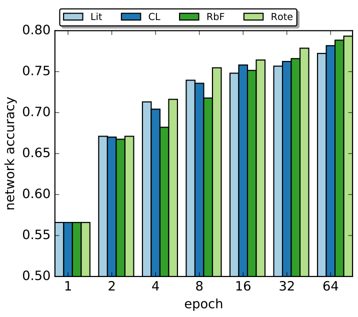
(c) Efficiency CIFAR10



(d) Effectiveness IMDb



(e) Effectiveness Addition



(f) Effectiveness CIFAR10

Figure 8: Efficiency and Effectiveness of schedulers across three datasets and networks. RbF uses Cos $\eta = .5$ as kernel. CL starts with (small set of) easier instances and gradually incorporate slightly harder instances at each iteration. Lit and RbF start big and gradually delay reviewing easy instances.

easier instances and gradually increases the amount of training data by adding slightly harder instances into its training set. On the other hand, Lit and RbF start big and gradually delay reviewing (easy) instances that the networks have learned. The difference between these two training paradigms is apparent in Figures 8(a)-8(c).

The results also show that the efficiency of a training paradigm depends on the initial effectiveness of the downstream neural network. For CL to be efficient, the neural network need to initially have *low* performance (accuracy) so that the scheduler works on smaller set of easy instances. For example, in case of Addition, Figures 8(b) and 8(e), the initial network accuracy is only 35%, therefore most instances are expected to be initially treated as hard instances and don't be used for training. On the other hand, CL shows a considerably lower efficiency for networks with slightly high initial accuracy, e.g. in case of IMDb or CIFAR10 where the initial network accuracy is above 56%, see Figures 8(a) and 8(d), and 8(c) and 8(f) respectively.

In contrast to CL, Lit and RbF are more efficient when the network has a relatively higher initial performance. A higher initial performance helps the schedulers to more confidently delay "reviewing" most instances and therefore train with a much smaller set of instances. For example, since the initial network accuracy in IMDb or CIFAR10 is above 56%, Lit and RbF are considerably more efficient from the beginning of the training process. However, in case of low initial performance, Lit and RbF tend to avoid delaying instances at lower iterations which leads to poor efficiency at the beginning. This is the case for the Addition dataset in which instances are gradually delayed by these two schedulers even at epoch 8 when the performance of the network reaches above 65%, see Figures 8(e) and 8(b). However, Lit gains its true efficiency after iteration 12, see Figure 8(b), while RbF still gradually improves the efficiency. This might be because of the lower bound delays that RbF estimates, see Equation (9).

Furthermore, the effectiveness results in Figure 8 (bottom) show that all schedulers produce comparable accuracy to the Rote scheduler throughout the training process, not just at specific iterations. This indicates that these training paradigms can much faster achieve the same generalizability as standard training, see Figures 8(b) and 8(e).

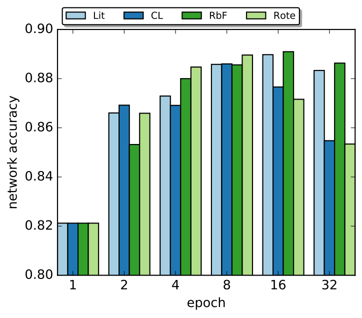


Figure 9: Robustness against overtraining.

4.3 Robustness against Overtraining

We investigate the effect of spaced repetition on overtraining. The optimal number of training epochs required to train `fastText` on the IMDb dataset is 8 epochs (see Table 1). In this experiment, we run `fastText` on IMDb for greater number of iterations to investigate the robustness of different schedulers against overtraining. The results in Figure 9 show that Lit and RbF (Cos $\eta = 0.5$) are more robust against overtraining. In fact, the performance of Lit and RbF further improve at epoch 16 while CL and Rote overfit at epoch 16 (note that CL and Rote also require considerably more amount of time to reach to higher iterations). We attribute the robustness of Lit and RbF to the scheduling mechanism which helps the networks to avoid retraining with easy instances. On the other hand, overtraining affects Lit and RbF at higher training iterations, compare performance of each scheduler at epochs 8 and 32. This might be because these training paradigms overfit the network by paying too much training attention to very hard instances which might introduce noise to the model.

5 Related Work

Ebbinghaus (1913, 2013), and recently Murre and Dros (2015), studied the hypothesis of the exponential nature of forgetting, i.e. how information is lost over time when there is no attempt to retain it. Previous research identified three critical indicators that affect the probability of recall: repeated exposure to learning materials, elapsed time since their last review (Ebbinghaus, 1913; Wixted, 1990; Dempster, 1989), and more recently item difficulty (Reddy et al., 2016). We based our investigation on these findings and validated that these indicators indeed affect memory retention in neural networks. We then developed training paradigms that utilize the above indicators to train networks.

Bengio et al. (2009) and Kumar et al. (2010) also developed cognitively-motivated training paradigms which are inspired by the principle that learning can be more effective when training starts with easier concepts and gradually proceeds with more difficult ones. Our idea is motivated by the spaced repetition principle which indicates learning improves with repeated exposure and decays with delay since last exposure (Ebbinghaus, 1913; Dempster, 1989). Based on this principle, we developed schedulers that space the reviews of training instances over time for efficient and effective training of neural networks.

6 Conclusion and Future Work

We developed a cognitively-motivated training paradigm (scheduler) that space instances over time for efficient and effective training of neural networks. Our scheduler only uses a small fraction of training data per epoch but still effectively train neural networks. It achieves this by estimating the time (number of epochs) by which training could be delayed for each instance. Our work was inspired by three recall indicators that affect memory retention in humans, namely difficulty of learning materials, delay since their last review, and memory strength of the learner, which we validated in the context of neural networks.

There are several avenues for future work including the extent to which our RbF model and its kernels could be combined with curriculum learning or Leitner system to either predict easiness of novel training instances to inform curriculum learning or incorporate Leitner's queueing mechanism to the RbF model. Other directions include extending RbF to dynamically learn the recall confidence parameter with respect to network behavior, or developing more flexible delay functions with theoretical analysis on their lower and upper bounds.

Acknowledgments

We thank Mitra Mohtarami for her constructive feedback during the development of this paper and anonymous reviewers for their thoughtful comments. This work was supported by National Institutes of Health (NIH) grant R01GM114355 from the National Institute of General Medical Sciences (NIGMS). The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

References

Lee Averell and Andrew Heathcote. 2011. The form of the forgetting curve and the fate of memories. *Journal of Mathematical Psychology*, 55(1):25–35.

Sumit Basu and Janara Christensen. 2013. Teaching classification boundaries to humans. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, pages 109–115. AAAI Press.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM.

Nicholas J Cepeda, Harold Pashler, Edward Vul, John T Wixted, and Doug Rohrer. 2006. Distributed practice in verbal recall tasks: A review and quantitative synthesis. *Psychological bulletin*, 132(3):354.

Tsung-Han Chan, Kui Jia, Shenghua Gao, Jiwen Lu, Zinan Zeng, and Yi Ma. 2015. Pcanet: A simple deep learning baseline for image classification? *IEEE Transactions on Image Processing*, 24(12):5017–5032.

Frank N Dempster. 1989. Spacing effects and their implications for theory and practice. *Educational Psychology Review*, 1(4):309–330.

Hermann Ebbinghaus. 1913. *Memory: A contribution to experimental psychology*. 3. University Microfilms.

Hermann Ebbinghaus. 2013. Memory: A contribution to experimental psychology. *Annals of neurosciences*, 20(4):155.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Julia Hirschberg and Christopher D Manning. 2015. Advances in natural language processing. *Science*, 349(6245):261–266.

Lu Jiang, Deyu Meng, Shoou-I Yu, Zhenzhong Lan, Shiguang Shan, and Alexander Hauptmann. 2014. Self-paced learning with diversity. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2078–2086. Curran Associates, Inc.

Lu Jiang, Deyu Meng, Qian Zhao, Shiguang Shan, and Alexander G. Hauptmann. 2015. Self-paced curriculum learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 2694–2700.

Armand Joulin, Edouard Grave, and Piotr Bojanowski Tomas Mikolov. 2017. Bag of tricks for efficient text classification. *European Chapter of the Association for Computational Linguistics, EACL 2017*, page 427.

M Pawan Kumar, Benjamin Packer, and Daphne Koller. 2010. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, pages 1189–1197.

Jaap MJ Murre and Joeri Dros. 2015. Replication and analysis of ebbinghaus' forgetting curve. *PloS one*, 10(7):e0120644.

Timothy P Novikoff, Jon M Kleinberg, and Steven H Strogatz. 2012. Education of a model student. *Proceedings of the National Academy of Sciences*, 109(6):1868–1873.

Siddharth Reddy, Igor Labutov, Siddhartha Banerjee, and Thorsten Joachims. 2016. Unbounded human learning: Optimal scheduling for spaced repetition. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1815–1824. ACM.

Valentin I Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. 2010. From baby steps to leapfrog: How less is more in unsupervised dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 751–759. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. 2016. Towards ai-complete question answering: A set of prerequisite toy tasks. *5th International Conference on Learning Representations.*

John T Wixted. 1990. Analyzing the empirical course of forgetting. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 16(5):927.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144.*