Neural Self-Training through Spaced Repetition

Hadi Amiri
Department of Biomedical Informatics, Harvard
10 Shattuck Street, Boston, MA 02115
hadi@hms.harvard.edu

Abstract

Self-training is a semi-supervised learning approach for utilizing unlabeled data to create better learners. The efficacy of self-training algorithms depends on their data sampling techniques. The majority of current sampling techniques are based on predetermined policies which may not effectively explore the data space or improve model generalizability. In this work, we tackle the above challenges by introducing a new data sampling technique based on *spaced repetition* that dynamically samples informative and diverse unlabeled instances with respect to individual learner and instance characteristics. The proposed model is specifically effective in the context of neural models which can suffer from overfitting and high-variance gradients when trained with small amount of labeled data. Our model outperforms current semi-supervised learning approaches developed for neural networks on publicly-available datasets.

1 Introduction

It is often expensive or time-consuming to obtain labeled data for Natural Language Processing tasks. In addition, manually-labeled datasets may not contain enough samples for downstream data analysis or novelty detection (Wang and Hebert, 2016). To tackle these issues, semi-supervised learning (Zhu, 2006; Chapelle et al., 2009) has become an important topic when one has access to small amount of labeled data and large amount of unlabeled data.

Self-training is a type of semi-supervised learning in which a downstream learner (e.g. a classifier) is first trained with labeled data, then the trained model is applied to unlabeled data to generate more labeled instances. A select sample of these instances together with their *pseudo* (predicted) labels are added to the labeled data and the learner is re-trained using the new labeled dataset. This process repeats until there is no more unlabeled data left or no improvement is observed in model performance on validation data (Zhu, 2006; Zhu and Goldberg, 2009).

Conventional self-training methods often rely on prediction confidence of their learners to sample unlabeled data. Typically the most confident unlabeled instances are selected (HEARST, 1991; Yarowsky, 1995; Riloff and Jones, 1999; Zhou et al., 2012). This strategy often causes only those unlabeled instances that match well with the current model being selected during self-training, therefore, the model may fail to best generalize to complete sample space (Zhang and Rudnicky, 2006; Wu et al., 2018). Ideally, a self-training algorithm should explore the space thoroughly for better generalization and higher performance. Recently Wu et al. (2018) developed an effective data sampling technique for "co-training" (Blum and Mitchell, 1998) methods which require two distinct views of data. Although effective, this model can't be readily applied to some text datasets due to the two distinct view requirement.

In the context of neural networks, *pretraining* is an effective semi-supervised approach in which layers of a network are first pretrained by learning to reconstruct their inputs, and then network parameters are optimized by supervised fine-tuning on a target task (Hinton and Salakhutdinov, 2006; Bengio et al., 2007; Erhan et al., 2010). While pretraining has been effective in neural language modeling and document classification (Dai and Le, 2015; Miyato et al., 2016), it has an inherent limitation: the *same* neural model or parts thereof must be used in both pretraining and fine-tuning steps. This poses a major limitation on the design choices as some pretraining tasks may need to exploit several data types (e.g., speech and text), or might require deeper network architectures.

Proceedings of NAACL-HLT 2019, pages 21–31
Minneapolis, Minnesota, June 2 - June 7, 2019.

The above challenges and intuitions inspire our work on developing a novel approach for neural self-training. The core part of our approach is a data sampling policy which is inspired by findings in cognitive psychology about spaced repetition (Dempster, 1989; Cepeda et al., 2006; Averell and Heathcote, 2011); the phenomenon in which a learner (often a human) can learn efficiently and effectively by accurately scheduling reviews of learning materials. In contrast to previous self-training approaches, our spaced repetition-based data sampling policy is not predetermined, explores the entire data space, and dynamically selects unlabeled instances with respect to the "strength" of a downstream learner on a target task, and "easiness" of unlabeled instances. In addition, our model relaxes the "same model" constraint of pretraining-based approaches by naturally decoupling pretraining and fine-tuning models through spaced repetition.

The contributions of this paper are (a): we propose an effective formulation of spaced repetition for self-training methods; to the best of our knowledge, this is the first work that investigates spaced repetition for semi-supervised learning, (b): our approach dynamically samples data, is not limited to predetermined sampling strategies, and naturally decouples pretraining and fine-tuning models, and (c): it outperforms current state-of-the-art baselines on large-scale datasets.

Our best model outperforms standard and current state-of-the-art semi-supervised learning methods by 6.5 and 4.1 points improvement in macro-F1 on sentiment classification task, and 3.6 and 2.2 points on churn classification task. Further analyses show that the performance gain is due to our model's ability in sampling *diverse* and *informative* unlabeled instances (those that are different from training data and can improve model generalizability).

2 Method

Conventional self-training methods employ the following steps to utilize unlabeled data for semi-supervised learning: (1) train a learner, e.g. a classifier, using labeled data, (2) iteratively select unlabeled instances based on a data sampling technique, and add the sampled instances (together with their predicted *pseudo* labels) to the labeled data, and (3) iteratively update the learner using the new labeled dataset.

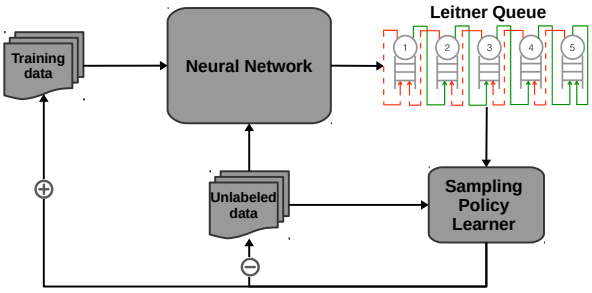


Figure 1: Neural Self-training Framework: at every self-training episode, the network uses current labeled data to iteratively optimize its parameters against a target task, and dynamically explores unlabeled data space through spaced repetition (specifically Leitner queue) to inform a data sampler that selects unlabeled data for the next self-training episode. Dashed/Red and solid/green arrows in Leitner queue indicate instance movements among queues.

The core difference between self-training algorithms is in the second step: data sampling policy. In this paper, we develop a new data sampling technique based on "spaced repetition" which dynamically explores the data space and takes into account instance and learner characteristics (such as easiness of instances or learner strength on target task) to sample unlabeled data for effective self-training.

Figure 1 illustrates our proposed neural self-training framework. We assume the downstream learner is a neural network that, at every self-training episode, (a): takes *current* labeled and unlabeled data as input, (b): uses labeled data to iteratively optimize its parameters with respect to a target task, and (c): dynamically explores unlabeled data space through spaced repetition to inform a data sampler that selects unlabeled data for the next self-training episode.

2.1 Spaced Repetition

Spaced repetition (Dempster, 1989; Cepeda et al., 2006; Averell and Heathcote, 2011) was presented in psychology 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 humans and machines can better learn by scheduling reviews of materials so that more time is spent on difficult concepts and less time on easier ones (Dempster, 1989; Novikoff et al., 2012; Amiri et al., 2017).

In this paper, we focus on a specific spaced repetition framework called Leitner system (Leitner, 1974). Suppose we have n queues $\{q_0, \ldots, q_{n-1}\}$. In general, Leitner system initially places all instances in the first queue, q_0. During training, if an instance from q_i is correctly classified by the learner, it will be "promoted" to q_{i+1} (solid/green arrows in Figure 1), otherwise it will be "demoted" to the previous queue, q_{i-1} (dashed/red arrows in Figure 1). Therefore, as the learner trains through time, higher queues will accumulate instances that are easier for the learner, while lower queues will accumulate harder instances.

To use Leitner system for neural self-training, we assume our learner is a neural network, place all *unlabeled* instances in the first queue of Leitner system (line 2 in Algorithm 1), and gradually populate them to other queues while training the network. Our Leitner system uses iteration-specific network predictions on unlabeled instances and current pseudo labels of these instances to move them between queues (see line 4-5 in Algorithm 1); pseudo labels can be obtained through posterior predictions generated by any trained downstream learner (see Section 2.2). Instances with similar class predictions and pseudo labels will be promoted to their next queues, and those with opposite predictions and labels will be demoted to lower queues. We note that, errors (e.g. inaccurate pseudo labels or network predictions) can inversely affect instance movements among queues. However, our sampling technique (see below) alleviates this issue because such misleading instances, if sampled, can't improve the generalizability of downstream learners. Details of our Leitner system is shown in Table 1.

Algorithm 1. Leitner system

Input:
$\mathbf{L}, \mathbf{U}, \mathbf{V}$: labeled, unlabeled, and validation data
$\mathbf{y}$: pseudo labels for $\mathbf{U}$
k : number of training epochs
n : number of queues
Output:
Q: Leitner queue populated with $\mathbf{U}$

1 $Q = [q_0, q_1, \ldots, q_{n-1}]$
2 $q_0 = [\mathbf{U}]$, $q_i = []$ for $i \in [1, n-1]$
3 **for** $epoch = 1$ **to** k:
4 $model = \texttt{epoch\_train}(\mathbf{L}, \mathbf{V})$
5 $promos, demos = \texttt{eval}(Q, \mathbf{y}, model)$
6 $Q = \texttt{schedule}(Q, promos, demos)$
7 **end for**
8 **return** Q

Table 1: Leitner system for neural self-training. All *unlabeled* instances are initially placed in the first queue and then populated to other queues depending on their easiness and learner (network) performance. `epoch_train`(.) uses training data to train the network for a single epoch and returns a trained model, `eval`(.) applies the current model on unlabeled instances in all queues and, based on given pseudo labels (treated as gold labels), returns lists of correctly and incorrectly classified instances, $promos$ and $demos$ respectively, and `schedule`(.) moves $promos$ and $demos$ instances to their next and previous queues respectively, and returns the updated queue.

2.2 Self-Training with Leitner Queues

We formulate the data sampling process as a decision-making problem where, at every self-training episode, the decision is to select a subset of unlabeled instances for self-training using information from Leitner queues. A simple, yet effective, approach to utilize such information is a greedy one in which instances of the queue that most improves the performance of the current model on validation data will be selected. We refer to this queue as *designated* queue:

Algorithm 2 shows details of our self-training approach. At every episode, we use current labeled data to train a task-specific neural network (line 2). Here, we weight the loss function using class size to deal with imbalanced data, and weight pseudo-labeled instances (as a function of episodes) to alleviate the effect of potentially wrong pseudo labels while training the network. We then use the trained network to generate pseudo labels for current unlabeled instances (line 3). These instances are then populated in Leitner queues as described before (line 4). Given the populated Leitner queues, the sample for current self-training episode is then created using instances of the designated queue, the queue that most improves the performance of the current network on validation data (lines 5-8). Instances of the designated queue will be removed from unlabeled data and added to labeled data with their pseudo labels treated as gold labels (lines 9-10).

We note that finding designated queues (lines 5-8 in Algorithm 2) imposes computational complexity on our model. However, in practice, we observe that designated queues are almost always among middle or higher queues in Leitner system, i.e. $q_i, \forall i \in [\lfloor n/2 \rfloor, n-1]$ where n in the number of queues. This can help accelerating the search

Algorithm 2. Neural Self-training

Input:
$\mathbf{L}, \mathbf{U}, \mathbf{V}$: labeled, unlabeled, and validation data
K : number of self-training episodes
Output:
M: classification model

for $episode = 1$ **to** K:
  $M_L = \texttt{train}(\mathbf{L}, \mathbf{V})$
  $\mathbf{y} = \texttt{label}(M_L, \mathbf{U})$
  $Q = \texttt{Leitner\_system}(\mathbf{L}, \mathbf{U}, \mathbf{V}, \mathbf{y})$ \\Alg. 1
  for q **in** Q:
    $M_q = \texttt{train}(\mathbf{L} + [q, \mathbf{y}[q]], \mathbf{V})$
  end for
  $M, q_{desig} = \texttt{get\_best}(M_Q, M_L)$
  $\mathbf{L} = \mathbf{L} + [q_{desig}, \mathbf{y}[q_{desig}]]$
  $\mathbf{U} = \mathbf{U} - q_{desig}$
end for
return M

Table 2: Proposed neural self-training framework. `train`(.) uses current labeled data to train the network and returns a trained model, `label`(.) generates pseudo labels for unlabeled instances using the trained model, `Leitner_system`(.) populates current unlabeled instances in Leitner queue, and `get_best`(.) compares performance of given models on validation data and returns the best model in conjunction with the queue that leads to the best performance, if any. Instances of the designated queue will be removed from unlabeled data and added to labeled data with their pseudo labels treated as gold labels.

process. In addition, learning a data sampling policy from movement patterns of instances among queues may help alleviating/eliminating the need for such an iterative search; see Section 4.4.

Finally, at test time, we apply the resulting self-trained network to test data and use the result for model comparison.

3 Experiments

We compare different self-training approaches in two settings where learners (neural networks) have low or high performance on original labeled data. This consideration helps investigating sensitivity of different self-training algorithms to the initial performance of learners.

3.1 Datasets and Evaluation Metric

As datasets, we use movie reviews from IMDb and short microblog posts from Twitter. These datasets and their corresponding tasks are described below and their statistics are provided in Table 3. In terms of preprocessing, we change all texts to lowercase, and remove stop words, user names, and URLs from texts in these datasets:

	Train	Val.	Test	Unlabeled
IMDb	5×1K	5×1K	5×48K	50K
Churn	5×1K	5×1K	5×3K	100K

Table 3: Statistics of dataset used in experiments.

IMDb: The IMDb dataset was developed by Maas et al. (2011)[1] for sentiment classification where systems should classify the polarity of a given movie review as positive or negative. The dataset contains 50K labeled movie reviews. For the purpose of our experiments, we randomly sample 1K, 1K, and 48K instances from this data (with balanced distribution over classes) and treat them as labeled (training), validation, and test data respectively. We create five such datasets for robustness against different seeding or data partitions. This dataset also provides 50K unlabeled reviews.

Churn: This dataset contains more than 5K tweets about three telecommunication brands and was developed by Amiri and Daumé III (2015)[2] for the task of *churn* prediction[3] where systems should predict if a twitter post indicates user intention about leaving a brand - classifying tweets as *churny* or *non-churny* with respect to brands. We replace all target brand names with the keyword BRAND and other non-target brands with BRAND-OTHER for the purpose of our experiments. Similar to IMDb, we create five datasets for experiments. We also crawl an additional 100K tweets about the target brands and treat them as unlabeled data.

We evaluate models in terms of macro-F1 score, i.e. the mean of F1-scores across classes.

3.2 Downstream Learner and Settings

As downstream neural networks (referred to as base classifiers), we consider current state-of-the-art deep averaging networks (DANs) (Shen et al., 2018; Iyyer et al., 2015; Joulin et al., 2017; Arora et al., 2017) for IMDb, and a basic CNN model for Churn dataset with parameters set from the work presented in (Gridach et al., 2017) except for pre-trained embeddings. In terms of DANs, we use `FastText` (Joulin et al., 2017) for its high per-

[1] http://ai.stanford.edu/~amaas/data/sentiment/

[2] https://scholar.harvard.edu/hadi/chData

[3] Churn is a term relevant to customer retention in marketing discourse; examples of churny tweets are "my days with BRAND are numbered," "debating if I should stay with BRAND," and "leaving BRAND in two days."

formance and simplicity. `FastText` is a feedforward neural network that consists of an embedding layer that maps vocabulary indices to embeddings, an averaging layer that averages word embeddings of inputs, and several hidden layers (we use two layers of size 256) followed by a prediction layer with `sigmoid` activation.

We use 300-dimensional word embeddings provided by Google's `word2vec` toolkit (Mikolov et al., 2013). In Algorithm 1, we set the number of training epochs to $k = 32$, and stop training when F1 performance on validation data stops improving with patience of three continuous iterations, i.e. after three continuous epochs with no improvement, training will be stopped. In addition, we set the number of training episodes to $K = 20$ and stop training when this number of episodes is reached or there is no unlabeled data left for sampling; the latter case is often the reason for stopping in our self-training method. In addition, we experiment with different number of Leitner queues chosen from $n = \{3, 5, 7, 9, 11\}$.

3.3 Baselines

We consider the following baselines:

- **Standard self-training**: This approach iteratively trains a network on current labeled data and applies it to current unlabeled data; it uses a prediction confidence threshold to sample unlabeled instances (Zhu, 2006). We set the best confidence threshold from {.80,.85,.90,.95} using validation data.
- **Autoencoder self-training** (Dai and Le, 2015): This approach first pretrains a network using unlabeled data (through a layer-wise training approach to optimally reconstruct the inputs), and then fine-tunes it using labeled data with respect to the target task.
- **Adversarial self-training** (Miyato et al., 2016): This model utilizes pretraining as described above, but also applies adversarial perturbations to word embeddings for more effective learning (perturbation is applied to embeddings instead of word inputs because words or their one-hot vectors do not admit infinitesimal perturbation; the network is trained to be robust to the worst perturbation).
- **Knowledge Transfer self-training** (Noroozi et al., 2018): This model uses a clustering approach (e.g. k-means) to create clusters of unlabeled instances that have similar representations, where representations are derived from standard pretraining as described above. The model then pretrains a network by learning to *classify* unlabeled instances to their corresponding clusters. The resulting pretrained network is then fine-tuned with respect to the target task using labeled data (with slight modification at prediction layer which makes the network suitable for target task). We set the best number of clusters from $\{10, 20, \ldots, 100\}$ based on model performance on validation data.

	IMDb	Churn
Base Classifier	73.02	65.77
SST (Standard ST)	74.43	65.77
PST (Pretraining ST)	76.36	67.27
AST (Adversarial ST)	76.09	67.70
KST (Knowledge Transfer ST)	77.11	67.06
LST (Leitner ST)	**78.27***	**69.90***

Table 4: Macro-F1 performance of models across datasets; Note that Standard ST (SST) samples only 1.4K and 0 instances from IMDb and Churn datasets respectively; sampling more data decreases SST's performance down to 66.94 and 57.04 perhaps due to ineffective exploring of data space. Our model achieves its best performance on IMDb and Churn datasets with $n = 5$ and $n = 7$ Leitner queues respectively.

3.4 Model Performance

Table 4 reports Macro-F1 performance of different models; we report average performance across five random test sets for each task (see Section 3.1 and Table 3). The performance of base classifiers in supervised settings, where the networks are only trained on original labeled datasets, is reasonably high on IMDb (73.02) and low on Churn (65.77). Standard ST (SST) improves performance on IMDb but not on Churn dataset. SST achieves its best performance (on validation data) in the first few episodes when, on average, 1.4K and 0 instances are sampled for IMDb and Churn datasets respectively. Beyond that, the performance considerably decreases down to 66.94 (IMDb) and 57.04 (Churn) respectively. This is perhaps due to imbalanced class size in Churn dataset, failure of SST to explore the data space, or classification mistakes that reinforce each other. Several previous works also observed no improvement with SST (Gollapalli et al., 2013; Zhu and Goldberg, 2009; Zhang and Rudnicky, 2006); but some successful applications have been reported (Wu et al.,

2018; Zhou et al., 2012; Riloff and Jones, 1999; Yarowsky, 1995; HEARST, 1991).

The result also show that pretraining and adversarial-based training, PST and AST in Table 4 respectively, improve the performance of base classifiers by 3.34 and 3.37 points in macro-F1 on IMDb, and by 1.5 and 1.93 points on Churn dataset. In addition, since PST and AST show comparable performance, we conjecture that when original labeled data has a small size, adversarial-based self-training do not considerably improve pretraining. But, considerable improvement can be achieved with larger amount of labeled data, see (Miyato et al., 2016) for detailed comparison on pretraining and adversarial-based training. The results also show that knowledge transfer (KST) outperforms PST and AST on IMDb - indicating that good initial labels derived through clustering information could help semi-supervised learning, even with small amount of seed labeled data.

Table 4 also shows the result of our model, Leitner ST (LST). The best performance of LST is obtained using $n = 5$ and $n = 7$ queues for IMDb and Churn datasets respectively. Considering these queue lengths, our model outperforms base classifiers by 5.25 and 4.13 points in Macro-F1 on IMDb and Churn datasets respectively; similar to PST and AST, our model results in a greater gain when the learner has higher initial performance. It also improves the best self-training baseline, KST for IMDb and AST for Churn, by 1.16 and 2.2 points in macro-F1 on IMDb and Churn datasets respectively where both differences are significant (average ρ-values based on t-test are .004 and .015 respectively).

4 Model Introspection

We investigate several questions about our model to shed light on its improved performance. One partial explanation is that by differentiating instances and augmenting the informative ones, we are creating a more powerful model that better explores the space of unlabeled data. In this section, we elaborate on the behavior of our model by conducting finer-grained analysis at queue-level and investigating the following questions in the context of challenges of semi-supervised learning. Due to space limit, we mainly report results on IMDb and discuss corresponding behaviors on Churn dataset in the text.

4.1 Queue-level Performance

We analyze queue level performance to understand how instances of different queues contribute in creating better models during the self-training process. For this experiment, we train networks using our Leitner self-training framework as normal (where, at every iteration, only instances of the designated queue are added to training data), and report the average macro-F1 performance of the network–on validation data–if it is trained with instances of each queue. Concretely, we report average macro-F1 performance of models learned at line 6 of Algorithm 2 (see M_qs in Table 2).

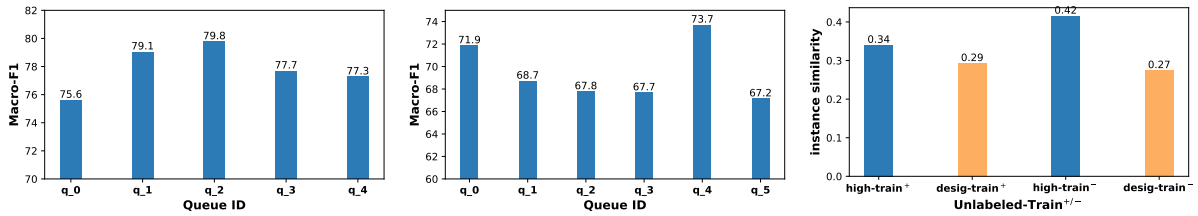
Figures 2(a) and 2(b) show the results on IMDb and Churn datasets for $n = 5$ and $n = 7$ queues respectively. Note that the last queue for Churn dataset, q_6, has never been reached by any instance. This is perhaps because of the difficulty of this task[4] and low initial performance of the network on Churn dataset. q_2 on IMDb and q_4 on Churn dataset result in the best average performance across training episodes, both queues are close to the middle. In addition, the result show that the highest queues (q_4 for IMDb and q_5 for Churn) are often not the best queues. This result can justify the lower performance of Standard ST (SST) as instances in these queues are the easiest (and perhaps most confident ones) for the network; we further analyze these queues in Section 4.2.[5]

4.2 What's the Issue with Highest Queues?

As we discussed before, instances in the highest queues, although easy to learn for the classifier, are not informative and do not contribute to training an improved model; therefore, highest queues are often not selected by our model. To understand the reason, we try to quantify how well instances of these queues *match* with training data. For this purpose, we compute `cosine` similarity between representations of training instances (see below) and those in the highest and designated queues

[4]Churn prediction is a target-dependent task, largely affected by negation and function words, e.g. compare "*switching from*" and "*switching to*," and language complexity, e.g. the tweets "*hate that I may end up leaving BRAND cause they have the best service*" is a positive yet churny tweet.

[5]Note that the performance on lower queues (e.g. q_1 for IMDb and q_0 for Churn) are higher than expected. This is because, at the end of each iteration, instances of designated (best-performing) queues–but not lower queues–are added to training data; instances of designated queues help creating better and more robust models which still perform well even if instances of lower queues are added.



(a) Queue level performance (IMDb) (b) Queue level performance (Churn) (c) Highest vs. designated queue (IMDb)

Figure 2: (a) and (b): Average macro-F1 performance computed over individual queues using validation dataset across training episodes (average performance of M_qs at line 6 of Algorithm 2). (a): Performance on IMDb with optimal queue length of $n = 5$, and (b): performance on Churn with optimal queue length of $n = 7$: note that none of unlabeled instances has made it to the last queue. (c): Comparison of highest and designated queues in terms of instance similarity to training data; `high_train` indicates similarity between (representations of) instances in the highest queue and training instances, and `desig_train` shows the corresponding values for instances in the designated queue. $+$ and $-$ signs indicate positive and negative pseudo/gold labels for unlabeled/training instances.

during self-training as follows:

$$\frac{1}{K}\sum_{e=1}^{K} \texttt{cosine}(\mathbf{T}^e, \mathbf{Q}^e)$$

where $\mathbf{T}^e \in \mathbb{R}^{m^e \times d}$ and $\mathbf{Q}^e \in \mathbb{R}^{p^e \times d}$ indicate representations of training instances and those of a given target queue respectively (where d indicates the dimension of representations, and m^e and p^e indicate number of instances in training data and target queue at episode e respectively), and `cosine(.,.)` computes L2-normalized dot product of its input matrices. To obtain the above representations for instances, we compute the output of the last hidden layer (the layer below prediction layer) of the *trained* network at each episode. These outputs can be considered as feature representations for inputs. For finer-grained comparison, we compute similarities with respect to positive and negative classes.

As the results in Figure 2(c) show, instances in the highest queue match well with current training data (and hence the current model), and, therefore, are less informative. On the other hand, instances in the designated queues show considerably smaller similarity with training instances in both positive and negative classes, and, therefore, do not match well with training data. These instances are more informative, and help the network to better explore the space of unlabeled data and optimize for the target task.

4.3 Does Diversity Matter?

We analyze different queues to measure the extent of diversity that each queue introduces to training data during our normal self-training process

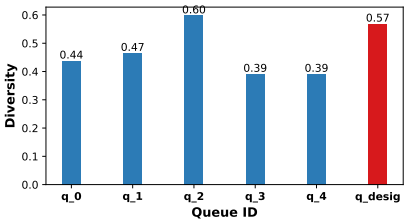


Figure 3: The amount of diversity that instances of each queue introduce if added to training data (on IMDb).

where, at every iteration, only instances of the designated queue are added to training data. Specifically, we compute the extent of diversity that each given queue introduces as follows:

$$\frac{1}{K}\sum_{e=1}^{K} 1 - \texttt{cosine}(\mathbf{T}^e, \texttt{concat}(\mathbf{T}^e, \mathbf{Q}^e))$$

where, as before, $\mathbf{T}^e$ and $\mathbf{Q}^e$ indicate the representations of training and queue instances at episode e respectively, and `concat(.,.)` is a function that creates a new dataset by vertically concatenating $\mathbf{T}^e$ and $\mathbf{Q}^e$.

Figure 3 shows the results. On IMDb, q_2 and designated queues show greater diversity to training data compared to other queues. We note that q_0 carries a greater diversity than q_3 and q_4, but, as we observed in Figure 2, instances of q_0 do not improve performance of the model, perhaps due to their difficulty or wrong pseudo labels. We observe similar behavior in case of Churn dataset where q_4 introduces the highest diversity. From this analysis, we conclude that Leitner self-training enables sampling diverse sets of instances that contributes to training an improved model.

	Leitner ST				
	$n = 3$	$n = 5$	$n = 7$	$n = 9$	$n = 11$
IMDb	76.83	76.77	78.03	75.34	**80.71**
Churn	65.74	64.87	67.06	**68.56**	65.80

Table 5: Macro-F1 performance of diverse queues across datasets. Compare these results with those obtained by designated queues in Table 4.

4.3.1 Diverse Queue

Given the above results on diversity, we investigate whether greater diversity can further improve the performance of our model. For this analysis, we create a considerably more "diverse" queue at every self-training episode and treat it as the designated queue. We create the diverse queue by sampling instances with *high* prediction confidence from *all* queues. In particular, at every episode, we rank instances of each queue based on their prediction confidence and create a diverse queue by combining top $r\%$ instances of each queue, where r indicates the rate of adding new instances and set to $r = 10\%$. We note that a smaller rate is better for adding instances because it allows the model to gradually consume unlabeled instances with high prediction confidence.

Table 5 shows the effect of diverse queues on the performance of our model on both IMDb and Churn datasets. The results show that diverse queues improve the performance of our Leitner self-training model from 78.27 (reported in Table 4) to 80.71 on IMDb, i.e. 2.44 points improvement in macro-F1. However, the corresponding performance on Churn dataset decreases from 69.90 to 68.56, i.e. 1.34 points decrease in macro-F1. The inverse effect of diverse queues in case of Churn dataset is because diverse queues suffer from the issue of considerable class imbalance more than designated queues. This is because highly confident instances which accumulate in higher queues are often negative instances in case of Churn prediction. Although we tackle this issue by weighting the loss function during training, diverse positive instances which are different from their training counterparts are still needed for performance improvement.

4.4 Do We Need Better Sampling Policies?

We investigate the challenges associated with our data sampling policy by conducting finer-grained analysis on instance movement patterns among queues. To illustrate, assume that we have a Leitner queue of size $n = 3$ and the following movement patterns for four individual instances that ultimately home in on q_0 (recall that correct prediction promotes an instance to a higher queue, while wrong prediction demotes it to a lower queue):

$q_0 \rightarrow q_0 \rightarrow q_0 \rightarrow q_0 \rightarrow q_0$: always in q_0
$q_0 \rightarrow q_1 \rightarrow q_0 \rightarrow q_0 \rightarrow q_0$: mainly in q_0
$q_0 \rightarrow q_1 \rightarrow q_0 \rightarrow q_1 \rightarrow q_0$: partially in q_0
$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_1 \rightarrow q_0$: partially in q_0 & q_1.

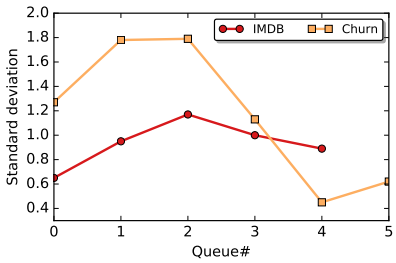


Figure 4: Deviation in instance movements for each queue (in terms of average standard deviation over all training episodes). At every episode, we keep track of instance movements among queues and measure movement variation among instances that ultimately home in on the same queue.

Although all these instances ultimately home in on the same queue, they may have different contributions to the training of a model because there is a considerable difference in the ability of the downstream network in learning their labels. Therefore, if there is a large deviation among movement patterns of instances of the same queue, better data sampling policies could be developed, perhaps through finer-grained queue-level sampling.

For this analyses, we keep track of instance movements among queues and measure standard deviation among movement patterns of instances of the same queue at every self-training episode, and report the average of these deviations.

Figure 4 shows the results. On both datasets, there is considerably greater deviation in movements for middle queues than lower/higher queues. This is meaningful because Leitner system (and other spaced repetition schedulers) are expected to keep easy and hard instances at higher and lower queues respectively. Since such instances mainly stay at lower or higher queues, we observe smaller deviation in their movements. On the other hand, the corresponding values for middle queues indicate that movements in these queues are spread out over a larger range of

queues. From these results, we conjecture that a data sampling policy that conducts finer-grained analysis at queue-level (e.g. by taking into account queue movement patterns) could create better data samples. Verifying this hypothesis will be the subject for future work.

5 Related Work

Semi-supervised learning (Zhu, 2006; Chapelle et al., 2009) is a type of machine learning where one has access to a small amount of labeled data and a large amount of unlabeled data. Self-training is a type of semi-supervised learning to boost the performance of downstream learners (e.g. classifiers) through data sampling from unlabeled data. Most data sampling policies rely on prediction confidence of the downstream learner for sampling unlabeled data (Zhu and Goldberg, 2009). Self-training has been successfully applied to various tasks and domains including word sense disambiguation (HEARST, 1991; Yarowsky, 1995), information extraction (Riloff and Jones, 1999), and object recognition (Zhou et al., 2012).

In addition, co-training (Blum and Mitchell, 1998; Zhang and Rudnicky, 2006; Wu et al., 2018) is another type of semi-supervised learning. It assumes that each instance can be described using two *distinct* feature sets that provide different and complementary information about the instance. Ideally, the two views should be conditionally independent, i.e., the two feature sets of each instance are conditionally independent given the class, and each view should be sufficient, i.e., the class of an instance can be accurately predicted from each view alone. Co-training first learns separate downstream learners for each view using a small set of labeled data. The most confident predictions of each learner on the unlabeled data are then used to iteratively construct additional labeled training data. Recently Wu et al. (2018) developed an effective model based on reinforcement learning (specifically, a joint formulation of a Q-learning agent and two co-training classifiers) to learn data sampling policies and utilize unlabeled data space in the context of co-training methods.

Effective semi-supervised learning algorithms based on pretraining techniques (Hinton and Salakhutdinov, 2006; Bengio et al., 2007; Erhan et al., 2010) have been developed for text classification, deep belief networks (Hinton and Salakhutdinov, 2006), and stacked autoencoders (Vincent et al., 2010; Bengio et al., 2007). In particular, Dai and Le (2015) developed an autoencoder for the later supervised learning process. Miyato et al. (2016) applied perturbations to word embeddings and used pretraining technique and adversarial training for effective semi-supervised learning. These models although effective have not been well studied in the context of semi-supervised learning where models may have low initial performance or limited amount of labeled data. In addition, pretraining is limited by the same architecture requirement in both pretraining and fine-tuning steps.

In this work, we extend previous work in self-training by developing a new and effective data sampling policy based on spaced repetition (Dempster, 1989; Cepeda et al., 2006; Averell and Heathcote, 2011) which addresses some of the above challenges. In particular, our model's data sampling policy is not predetermined, it explores the entire data space and dynamically selects unlabeled instances with respect to the strength of a learner on a target task and easiness of unlabeled instances, and it relaxes the same model constraint of pretraining-based approaches by decoupling pretraining and fine-tuning steps.

6 Conclusion and Future Work

We propose a novel method based on spaced repetition to self-train neural networks using small amount of labeled and large amount of unlabeled data. Our model can select high-quality unlabeled data samples for self-training and outperforms current state-of-the-art semi-supervised baselines on two text classification problems. We analyze our model from various perspectives to explain its improvement gain with respect to challenges of semi-supervised learning. There are several venues for future work including (a): finer-grained data sampling at queue level, (b): extending our model to other machine learning algorithms that employ iterative training, such as boosting approaches, and (c): applying this model to areas where neural networks have not been investigated, e.g. due to limited availability of labeled data.

Acknowledgments

I sincerely thank Mitra Mohtarami and anonymous reviewers for their insightful comments and constructive feedback.

References

Hadi Amiri and Hal Daumé III. 2015. Target-dependent churn classification in microblogs. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 2361–2367. AAAI Press.

Hadi Amiri, Timothy Miller, and Guergana Savova. 2017. Repeat before forgetting: Spaced repetition for efficient and effective training of neural networks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2401–2410.

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2017. A simple but tough-to-beat baseline for sentence embeddings. In *International conference on learning representations (ICLR)*.

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.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. 2007. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160.

Avrim Blum and Tom Mitchell. 1998. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100. 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.

Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. 2009. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542.

Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. In *Advances in neural information processing systems*, pages 3079–3087.

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

Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. 2010. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660.

Sujatha Das Gollapalli, Cornelia Caragea, Prasenjit Mitra, and C Lee Giles. 2013. Researcher homepage classification using unlabeled data. In *Proceedings of the 22nd international conference on World Wide Web*, pages 471–482. ACM.

Mourad Gridach, Hatem Haddad, and Hala Mulki. 2017. Churn identification in microblogs using convolutional neural networks with structured logical knowledge. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 21–30.

M HEARST. 1991. Noun homograph disambiguation using local context in large corpora. In *Proc. 7th Annual Conference of the Centre for the New OED and Text Research: Using Corpora*, pages 1–22.

Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. 2015. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of ACL-IJCNLP*.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431. Association for Computational Linguistics.

Sebastian Leitner. 1974. *So lernt man lernen*. Herder.

Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pages 142–150. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Takeru Miyato, Andrew M Dai, and Ian Goodfellow. 2016. Adversarial training methods for semi-supervised text classification. *International conference on learning representations (ICLR)*.

Mehdi Noroozi, Ananth Vinjimoor, Paolo Favaro, and Hamed Pirsiavash. 2018. Boosting self-supervised learning via knowledge transfer. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 9359–9367.

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.

Ellen Riloff and Rosie Jones. 1999. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proceedings of the sixteenth national conference on Artificial intelligence*, pages 474–479. American Association for Artificial Intelligence.

Dinghan Shen, Guoyin Wang, Wenlin Wang, Martin Renqiang Min, Qinliang Su, Yizhe Zhang, Chunyuan Li, Ricardo Henao, and Lawrence Carin. 2018. Baseline needs more love: On simple word-embedding-based models and associated pooling mechanisms. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, volume 1.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408.

Yu-Xiong Wang and Martial Hebert. 2016. Learning from small sample sets by combining unsupervised meta-training with cnns. In *Advances in Neural Information Processing Systems*, pages 244–252.

Jiawei Wu, Lei Li, and William Yang Wang. 2018. Reinforced co-training. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, volume 1, pages 1252–1262.

David Yarowsky. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pages 189–196. Association for Computational Linguistics.

Rong Zhang and Alexander I Rudnicky. 2006. A new data selection approach for semi-supervised acoustic modeling. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 1, pages I–I. IEEE.

Yan Zhou, Murat Kantarcioglu, and Bhavani Thuraisingham. 2012. Self-training with selection-by-rejection. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 795–803. IEEE.

Xiaojin Zhu. 2006. Semi-supervised learning literature survey. *Technical Report 1530, Computer Science, University of Wisconsin-Madison*, 2(3):4.

Xiaojin Zhu and Andrew B Goldberg. 2009. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130.