




Making Learners (More) Monotone

Tom Julian Viering¹ , Alexander Mey¹ , and Marco Loog^{1,2} 

¹ Delft University of Technology, Delft, The Netherlands

{t.j.viering,a.mey,m.loog}@tudelft.nl

² University of Copenhagen, Copenhagen, Denmark

Abstract. Learning performance can show non-monotonic behavior. That is, more data does not necessarily lead to better models, even on average. We propose three algorithms that take a supervised learning model and make it perform more monotone. We prove consistency and monotonicity with high probability, and evaluate the algorithms on scenarios where non-monotone behaviour occurs. Our proposed algorithm MT_{HT} makes less than 1% non-monotone decisions on MNIST while staying competitive in terms of error rate compared to several baselines. Our code is available at <https://github.com/tomviering/monotone>.

Keywords: Learning curve · Model selection · Learning theory

1 Introduction

It is a widely held belief that more training data usually results in better generalizing machine learning models—cf. [11, 17] for instance. Several learning problems have illustrated, however, that more training data can lead to worse generalization performance [3, 9, 12]. For the peaking phenomenon [3], this occurs exactly at the transition from the underparametrized to the overparametrized regime. This double-descent behavior has found regained interest in the context of deep neural networks [1, 18], since these models are typically overparametrized. Recently, also several new examples have been found, where in quite simple settings more data results in worse generalization performance [10, 19].

It can be difficult to explain to a user that machine learning models can actually perform worse when more, possibly expensive to collect data has been used for training. Besides, it seems generally desirable to have algorithms that guarantee increased performance with more data. How to get such a guarantee? That is the question we investigate in this work and for which we use learning curves. Such curves plot the expected performance of a learning algorithm versus the amount of training data.¹ In other words, we wonder how we can make learning curves monotonic.

The core approach to make learners monotone is that, when more data is gathered and a new model is trained, this newly trained model is compared to

¹ Not to be confused with training curves, where the loss versus epochs (optimization iterations) is plotted.

the currently adopted model that was trained on less data. Only if the new model performs better should it be used. We introduce several wrapper algorithms for supervised classification techniques that use the holdout set or cross-validation to make this comparison. Our proposed algorithm MT_{HT} uses a hypothesis test to switch if the new model improves significantly upon the old model. Using guarantees from the hypothesis test we can prove that the resulting learning curve is monotone with high probability. We empirically study the effect of the parameters of the algorithms and benchmark them on several datasets including MNIST [8] to check to what degree the learning curves become monotone.

This work is organized as follows. The notion of monotonicity of learning curves is reviewed in Sect. 2. We introduce our approaches and algorithms in Sect. 3, and prove consistency and monotonicity with high probability in Sect. 4. Section 5 provides the empirical evaluation. We discuss the main findings of our results in Sect. 6 and end with the most important conclusions.

2 The Setting and the Definition of Monotonicity

We consider the setting where we have a learner that now and then receives data and that is evaluated over time. The question is then, how to make sure that the performance of this learner over time is monotone—or with other words, how can we guarantee that this learner over time improves its performance?

We analyze this question in a (frequentist) classification framework. We assume there exists an (unknown) distribution P over $\mathcal{X} \times \mathcal{Y}$, where \mathcal{X} is the input space (features) and \mathcal{Y} is the output space (classification labels). To simplify the setup we operate in rounds indicated by i , where $i \in \{1, \dots, n\}$. In each round, we receive a batch of samples S^i that is sampled i.i.d. from P . The learner L can use this data in combination with data from previous rounds to come up with a hypothesis h_i in round i . The hypothesis comes from a hypothesis space \mathcal{H} . We consider learners L that, as subroutine, use a supervised learner $A: \mathcal{S} \rightarrow \mathcal{H}$, where \mathcal{S} is the space of all possible training sets.

We measure performance by the error rate. The true error rate on P equals

$$\epsilon(h_i) = \int_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} l_{0-1}(h_i(x), y) dP(x, y) \quad (1)$$

where l_{0-1} is the zero-one loss. We indicate the empirical error rate of h on a sample S as $\hat{\epsilon}(h, S)$. We call n rounds a run. The true error of the returned h_i by the learner L in round i is indicated by ϵ_i , all the ϵ_i 's of a run form a learning curve. By averaging multiple runs one obtains the expected learning curve, $\bar{\epsilon}_i$.

The goal for the learner L is twofold. The error rates of the returned models ϵ_i 's should (1) be as small as possible, and (2) be monotonically decreasing. These goals can be at odds with another. For example, always returning a fixed model ensures monotonicity but incurs large error rates. To measure (1), we summarize performance of a learning curve using the Area Under the Learning Curve (AULC) [6, 13, 16]. The AULC averages all ϵ_i 's of a run. Low AULC indicates that a learner manages to quickly reduce the error rate.

Monotone in round i means that $\epsilon_{i+1} \leq \epsilon_i$. We may care about monotonicity of the expected learning curve *or* individual learning curves. In practice, however, we typically get one chance to gather data and submit models. In that case, we rather want to make sure that then any additional data also leads to better performance. Therefore, we are mainly concerned with monotonicity of *individual* learning curves. We quantify monotonicity of a run by the fraction of non-monotone transitions in an individual curve.

3 Approaches and Algorithms

We introduce three algorithms (learners L) that wrap around supervised learners with the aim of making them monotone. First, we provide some intuition how to achieve this: ideally, during the generation of the learning curve, we would check whether $\epsilon(h_{i+1}) \leq \epsilon(h_i)$. A fix to make a learner monotone would be to output h_i instead of h_{i+1} if the error rate of h_{i+1} is larger. Since learners do not have access to $\epsilon(h_i)$, we have to estimate it using the incoming data. The first two algorithms, $\text{MT}_{\text{SIMPLE}}$ and MT_{HT} , use the holdout method to this end; newly arriving data is partitioned into training and validation sets. The third algorithm, MT_{CV} , makes use of cross validation.

$\text{MT}_{\text{SIMPLE}}$: Monotone Simple. The pseudo-code for $\text{MT}_{\text{SIMPLE}}$ is given by Algorithm 1 in combination with the function `UpdateSimple`. Batches S^i are split into training (S_t^i) and validation (S_v^i). The training set S_t is enlarged each round with S_t^i and a new model h_i is trained. S_v^i is used to estimate the performance of h_i and h_{best} . We store the previously best performing model, h_{best} , and compare its performance to that of h_i . If the new model h_i is better, it is returned and h_{best} is updated, otherwise h_{best} is returned.

Because h_i and h_{best} are both compared on S_v^i the comparison is more accurate because the comparison is paired. After the comparison S_v^i can safely be added to the training set (line 7 of Algorithm 1).

We call this algorithm $\text{MT}_{\text{SIMPLE}}$ because the model selection is a bit naive: for small validation sets, the variance in the performance measure could be quite large, leading to many non-monotone decisions. In the limit of infinitely large S_v^i , however, this algorithm should always be monotone (and very data hungry).

MT_{HT} : Monotone Hypothesis Test. The second algorithm, MT_{HT} , aims to resolve the issues of $\text{MT}_{\text{SIMPLE}}$ with small validation set sizes. In addition, for this algorithm, we prove that individual learning curves are monotone with high probability. The same pseudo-code is used as for $\text{MT}_{\text{SIMPLE}}$ (Algorithm 1), but with a different update function `UpdateHT`. Now a hypothesis test HT determines if the newly trained model is significantly better than the previous model. The hypothesis test makes sure that the newly trained model is not better due to chance (such as an unlucky sample). The hypothesis test is conservative, and only switches to a new model if we are reasonably sure it is significantly better, to avoid non-monotone decisions. Japkowicz and Shah [7] provide an accessible introduction to understand the frequentist hypothesis testing.

Algorithm 1. M_{SIMPLE} and M_{HT}

input: supervised learner A , rounds n , batches S^i
 $u \in \{\text{updateSimple}, \text{updateHT}\}$
 if $u = \text{updateHT}$: confidence level α , hypothesis test HT

```

1  $S_t = \{\}$ 
2 for  $i = 1, \dots, n$  do
3   Split  $S^i$  in  $S_t^i$  and  $S_v^i$ 
4   Append to  $S_t : S_t = [S_t; S_t^i]$ 
5    $h_i \leftarrow A(S_t)$ 
6    $Update_i \leftarrow u(S_v^i, h_i, h_{\text{best}}, \alpha, HT)$  // see below
7   Append to  $S_t : S_t = [S_t; S_v^i]$ 
8   if  $Update_i$  or  $i = 1$  then
9     |  $h_{\text{best}} \leftarrow h_i$ 
10  end
11  Return  $h_{\text{best}}$  in round  $i$ 
12 end

```

Function UpdateSimple

input: $S_v^i, h_i, h_{\text{best}}$

```

1  $P_{\text{current}} \leftarrow \hat{\epsilon}(h_i, S_v^i)$ 
2  $P_{\text{best}} \leftarrow \hat{\epsilon}(h_{\text{best}}, S_v^i)$ 
3 return ( $P_{\text{current}} \leq P_{\text{best}}$ )

```

Function UpdateHT

input: $S_v^i, h_i, h_{\text{best}}$, confidence level α , hypothesis test HT

```

1  $p = HT(S_v^i, h_i, h_{\text{best}})$  // p-value
2 return ( $p \leq \alpha$ )

```

The choice of hypothesis test depends on the performance measure. For the error rate the McNemar test can be used [7, 14]. The hypothesis test should use paired data, since we evaluate two models on one sample, and it should be one-tailed. One-tailed, since we only want to know whether h_i is better than h_{best} (a two tailed test would switch to h_i if its performance is significantly different). The test compares two hypotheses: $H_0 : \epsilon(h_i) = \epsilon(h_{\text{best}})$ and $H_1 : \epsilon(h_i) < \epsilon(h_{\text{best}})$.

Several versions of the McNemar test can be used [4, 7, 14]. We use the McNemar exact conditional test which we briefly review. Let b be the random variable indicating the number of samples classified correctly by h_{best} and incorrectly by h_i of the sample S_v^i , and let N_d be the number of samples where they disagree. The test conditions on N_d . Assuming H_0 is true, $P(b = x | H_0, N_d) = \binom{N_d}{x} (\frac{1}{2})^{N_d}$. Given x b 's, the p -value for our one tailed test is $p = \sum_{i=0}^x P(b = i | H_0, N_d)$.

The one tailed p -value is the probability of observing a more extreme sample given hypothesis H_0 considering the tail direction of H_1 . The smaller the p -value, the more evidence we have for H_1 . If the p -value is smaller than α , we accept H_1 , and thus we update the model h_{best} . The smaller α , the more conservative the hypothesis test, and thus the smaller the chance that a wrong decision is made due to unlucky sampling. For the McNemar exact conditional test [4] the False Positive Rate (FPR, or the probability to make a Type I error) is bounded by α : $P(p \leq \alpha | H_0) \leq \alpha$. We need this to prove monotonicity with high probability.

MT_{CV}: Monotone Cross Validation. In practice, often K -fold cross validation (CV) is used to estimate model performance instead of the holdout. This is what MT_{CV} does, and is similar to MT_{SIMPLE}. As described in Algorithm 2, for each incoming sample an index I maintains to which fold it belongs. These indices are used to generate the folds for the K -fold cross validation.

During CV, K models are trained and evaluated on the validation sets. We now have to memorize K previously best models, one for each fold. We average the performance of the newly trained models over the K -folds, and compare that to the average of the best previous K models. This averaging over folds is essential, as this reduces the variance of the model selection step as compared to selecting the best model overall (like MT_{SIMPLE} does).

In our framework we return a single model in each iteration. We return the model with the optimal training set size that performed best during CV. This can further improve performance.

Algorithm 2. M_{CV}

```

input:  $K$  folds, learner  $A$ , rounds  $n$ , batches  $S^i$ 
1  $b \leftarrow 1$  // keeps track of best round
2  $S = \{\}, I = \{\}$ 
3 for  $i = 1, \dots, n$  do
4   Generate stratified CV indices for  $S^i$  and put in  $I^i$ . Each index  $i$ 
   indicates to which validation fold the corresponding sample belongs.
5   Append to  $S$ :  $S \leftarrow [S; S^i]$ 
6   Append to  $I$ :  $I \leftarrow [I; I^i]$ 
7   for  $k = 1, \dots, K$  do
8      $h_i^k \leftarrow A(S[I \neq k])$  // training set of  $k$ th fold
9      $P_i^k \leftarrow \hat{\epsilon}(h_i^k, S[I = k])$  // validation set of  $k$ th fold
10     $P_b^k \leftarrow \hat{\epsilon}(h_b^k, S[I = k])$  // update performance of prev. models
11  end
12   $Update_i \leftarrow (mean(P_i^k) \leq mean(P_b^k))$  // mean w.r.t.  $k$ 
13  if  $Update_i$  or  $i = 1$  then
14     $b \leftarrow i$ 
15  end
16   $k \leftarrow \arg \min_k P_b^k$  // break ties
17  Return  $h_b^k$  in round  $i$ 
18 end

```

4 Theoretical Analysis

We derive the probability of a monotone learning curve for MT_{SIMPLE} and MT_{HT}, and we prove our algorithms are consistent if the model updates enough.

Theorem 1. *Assume we use the McNemar exact conditional test (see Sect. 3) with $\alpha \in (0, \frac{1}{2}]$, then the individual learning curve generated by Algorithm MT_{HT} with n rounds is monotone with probability at least $(1 - \alpha)^n$.*

Proof. First we argue that the probability of making a non-monotone decision in round i is at most α . If $H_1 : \epsilon(h_i) < \epsilon(h_{\text{best}})$ or $H_0 : \epsilon(h_i) = \epsilon(h_{\text{best}})$ is true, we are monotone in round i , so we only need to consider a new alternative hypothesis $H_2 : \epsilon(h_i) > \epsilon(h_{\text{best}})$. Under H_0 we have [4]: $P(p \leq \alpha | H_0) \leq \alpha$. Conditioned on H_2 , b is binomial with larger mean than in the case of H_0 , thus we observe larger p -values if $\alpha \in (0, \frac{1}{2}]$, thus $P(p \leq \alpha | H_2) \leq P(p \leq \alpha | H_0) \leq \alpha$. Therefore the probability of being non-monotone in round i is at most α . This holds for any model h_i, h_{best} and anything that happened before round i . Since S_v^i are independent samples, being non-monotone in each round can be seen as independent events, resulting in $(1 - \alpha)^n$. \square

If the probability of being non-monotone in all rounds is at most β , we can set $\alpha = 1 - \beta^{\frac{1}{n}}$ to fulfill this condition. Note that this analysis also holds for $\text{MT}_{\text{SIMPLE}}$, since running MT_{HT} with $\alpha = \frac{1}{2}$ results in the same algorithm as $\text{MT}_{\text{SIMPLE}}$ for the McNemar exact conditional test.

We now argue that all proposed algorithms are consistent under some conditions. First, let us revisit the definition of consistency [17].

Definition 1 (Consistency [17]). *Let L be a learner that returns a hypothesis $L(S) \in \mathcal{H}$ when evaluated on S . For all $\epsilon_{\text{excess}} \in (0, 1)$, for all distributions D over $X \times Y$, for all $\delta \in (0, 1)$, if there exists a $n(\epsilon_{\text{excess}}, D, \delta)$, such that for all $m \geq n(\epsilon_{\text{excess}}, D, \delta)$, if L uses a sample S of size m , and the following holds with probability (over the choice of S) at least $1 - \delta$,*

$$\epsilon(L(S)) \leq \min_{h \in \mathcal{H}} \epsilon(h) + \epsilon_{\text{excess}}, \tag{2}$$

then L is said to be consistent.

Before we can state the main result, we have to introduce a bit of notation. U_i indicates the event that the algorithm updates h_{best} (or in case of M_{CV} it updates the variable b). H_i^{i+z} to indicates the event that $\neg U_i \cap \neg U_{i+1} \cap \dots \cap \neg U_{i+z}$, or in words, that in round i to $i + z$ there has been no update. To fulfill consistency, we need that when the number of rounds grows to infinity, the probability of updating is large enough. Then consistency of A makes sure that h_{best} has sufficiently low error. For this analysis it is assumed that the number of rounds of the algorithms is not fixed.

Theorem 2. *$\text{MT}_{\text{SIMPLE}}$, MT_{HT} and MT_{CV} are consistent, if A is consistent and if for all i there exists a $z_i \in \mathbb{N} \setminus 0$ and $C_i > 0$ such that for all $k \in \mathbb{N} \setminus 0$ it holds that $P(H_i^{i+kz_i}) \leq (1 - C_i)^k$.*

Proof. Let A be consistent with $n_A(\epsilon_{\text{excess}}, D, \delta)$ samples. Let us analyze round i where i is big enough such that² $|S_t| > n_A(\epsilon_{\text{excess}}, D, \frac{\delta}{2})$. Assume that

$$\epsilon(h_{\text{best}}) > \min_{h \in \mathcal{H}} \epsilon(h) + \epsilon_{\text{excess}}, \tag{3}$$

² In case of MT_{CV} , take $|S_t|$ to be the smallest training fold size in round i .

otherwise the proof is trivial. For any round $j \geq i$, since A produces hypothesis h_j with $|S_i| > n_A(\epsilon_{\text{excess}}, D, \frac{\delta}{2})$ samples,

$$\epsilon(h_j) \leq \min_{h \in \mathcal{H}} \epsilon(h) + \epsilon_{\text{excess}} \quad (4)$$

holds with probability of at least $1 - \frac{\delta}{2}$. Now L should update. The probability that in the next kz_i rounds we don't update is, by assumption, bounded by $(1 - C_i)^k$. Since $C_i > 0$, we can choose k big enough so that $(1 - C_i)^k \leq \frac{\delta}{2}$. Thus the probability of not updating after kz_i more rounds is at most $\frac{\delta}{2}$, and we have a probability of $\frac{\delta}{2}$ that the model after updating is not good enough. Applying the union bound we find the probability of failure is at most δ . \square

A few remarks about the assumption. It tells us, that an update is more and more likely if we have more consecutive rounds where there has been no update. It holds if each z_i rounds the update probability is nonzero. A weaker but also sufficient assumption is $\forall_i : \lim_{z \rightarrow \infty} P(H_i^{i+z}) \rightarrow 0$.

For $\text{MT}_{\text{SIMPLE}}$ and MT_{CV} the assumption is always satisfied, because these algorithms look directly at the mean error rate—and due to fluctuations in the sampling there is always a non-zero probability that $\hat{\epsilon}(h_i) \leq \hat{\epsilon}(h_{\text{best}})$. However, for MT_{HT} this may not always be satisfied. Especially if the validation batches N_v are small, the hypothesis test may not be able to detect small differences in error—the test then has zero power. If N_v stays small, even in future rounds the power may stay zero, in which case the learner is not consistent.

5 Experiments

We evaluate $\text{MT}_{\text{SIMPLE}}$ and MT_{HT} on artificial datasets to understand the influence of their parameters. Afterward we perform a benchmark where we also include MT_{CV} and a baseline that uses validation data to tune the regularization strength. This last experiment is also performed on the MNIST dataset to get an impression of the practicality of the proposed algorithms. First we describe the experimental setup in more detail.

Experimental Setup. The peaking dataset [3] and dipping dataset [9] are artificial datasets that cause non-monotone behaviour. We use stratified sampling to obtain batches S^i for the peaking and dipping dataset, for MNIST we use random sampling. For simplicity all batches have the same size. N indicates batch size, and N_v and N_t indicate the sizes of the validation and training sets.

As model we use least squares classification [5, 15]. This is ordinary linear least squares regression on the classification labels $\{-1, +1\}$ with intercept. For MNIST one-versus-all is used to train a multi-class model. In case there are less samples for training than dimensions, the required inverse of the covariance matrix is ill-defined and we resort to the Moore-Penrose Pseudo-Inverse.

Monotonicity is calculated by the fraction of non-monotone iterations per run. AULC is also calculated per run. We do 100 runs with different batches

and average to reduce variation from the randomness in the batches. Each run uses a newly sampled test set consisting of 10000 samples. The test set is used to estimate the true error rate and is not accessible by any of the algorithms.

We evaluate M_{SIMPLE} , M_{HT} and M_{CV} and several baselines. The standard learner just trains on all received data. A second baseline, λ_S , splits the data in train and validation like M_{SIMPLE} and uses the validation data to select the optimal L_2 regularization parameter λ for the least square classifier. Regularization is implemented by adding λI to the estimate of the covariance matrix.

In the first experiment we investigate the influence of N_v and α for MT_{SIMPLE} and MT_{HT} on the decisions. A complicating factor is that if N_v changes, not only decisions change, but also training set sizes because S_v is appended to the training set (see line 7 of Algorithm 1). This makes interpretation of the results difficult because decisions are then made in a different context. Therefore, for the first set of experiments, we do not add S_v to the training sets, also not for the standard learner. For this set of experiment We use $N_t = 4$, $n = 150$, $d = 200$ for the peaking dataset, and we vary α and N_v .

For the benchmark, we set $N_t = 10$, $N_v = 40$, $n = 150$ for peaking and dipping, and we set $N_t = 5$, $N_v = 20$, $n = 40$ for MNIST. We fix $\alpha = 0.05$ and use $d = 500$ for the peaking dataset. For MNIST, as preprocessing step we extract 500 random Fourier-features as also done by Belkin et al. [1]. For MT_{CV} we use $K = 5$ folds. For λ_S we try $\lambda \in \{10^{-5}, 10^{-4.5}, \dots, 10^{4.5}, 10^5\}$ for peaking and dipping, and we try $\lambda \in \{10^{-3}, 10^{-2}, \dots, 10^3\}$ for MNIST.

Results. We perform a preliminary investigation of the algorithms M_{SIMPLE} and M_{HT} and the influence of the parameters N_v and α . We show several learning curves in Fig. 1a and d. For small N_v and α we observe MT_{HT} gets stuck: it does not switch models anymore, indicating that consistency could be violated.

In Fig. 1b and e we give a more complete picture of all tried hyperparameters in terms of the AULC. In Fig. 1c and f we plot the fraction of non-monotone decisions during a run (note that the legends for the subfigures are different). Observe that the axes are scaled differently (some are logarithmic). In some cases zero non-monotone decisions were observed, resulting in a missing value due to $\log(0)$. This occurs for example if MT_{HT} always sticks to the same model, then no non-monotone decisions are made. The results of the benchmark are shown in Fig. 2. The AULC and fraction of monotone decisions are given in Table 1.

6 Discussion

First Experiment: Tuning α and N_v . As predicted MT_{SIMPLE} typically performs worse than MT_{HT} in terms of AULC and monotonicity unless N_v is very large. The variance in the estimate of the error rates on S_v^i is so large that in most cases the algorithm doesn't switch to the correct model. However, MT_{SIMPLE} seems to be consistently better than the standard learner in terms of monotonicity and AULC, while MT_{HT} can perform worse if badly tuned.

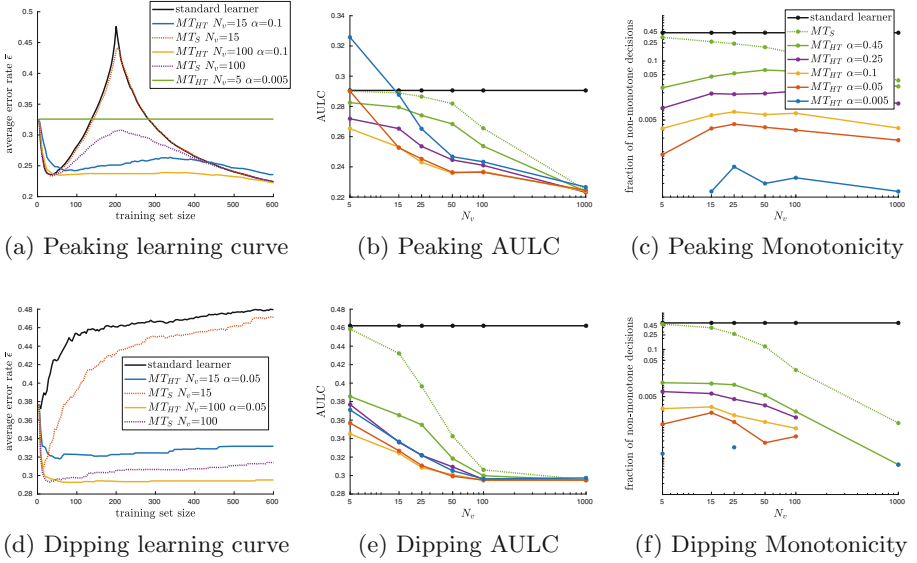


Fig. 1. Influence of N_v and α for MT_{SIMPLE} and MT_{HT} on the Peaking and Dipping dataset. Note that some axes are logarithmic and b, c, e, f have the same legend.

Larger N_v leads typically to improved AULC for both. $\alpha \in [0.05, 0.1]$ seems to work best in terms of AULC for most values of N_v . If α is too small, MT_{HT} can get stuck, if α is too large, it switches models too often and non-monotone behaviour occurs. If $\alpha \rightarrow \frac{1}{2}$, MT_{HT} becomes increasingly similar to MT_{SIMPLE} as predicted by the theory.

The fraction of non-monotone decisions of MT_{HT} is much lower than α . This is in agreement with Theorem 1, but could indicate in addition that the hypothesis test is rather pessimistic. The standard learner and MT_{SIMPLE} often make non-monotone decisions. In some cases almost 50% of the decisions are not-monotone.

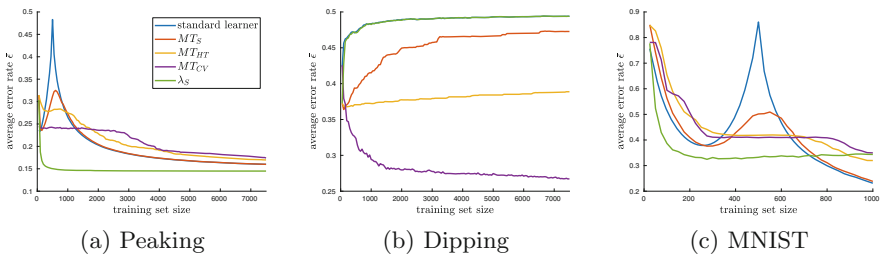


Fig. 2. Expected learning curves on the benchmark datasets.

Table 1. Results of the benchmark. SL is the Standard Learner. AULC is the Area Under the Learning Curve of the error rate. Fraction indicates the average fraction of non-monotone decisions during a single run. Standard deviation shown in (braces). Best monotonicity result is underlined.

	Peaking		Dipping		MNIST	
	AULC	Fraction	AULC	Fraction	AULC	Fraction
SL	0.198 (0.003)	0.31 (0.02)	0.49 (0.01)	0.50 (0.03)	0.44 (0.01)	0.27 (0.04)
MT_S	0.195 (0.005)	0.23 (0.03)	0.45 (0.06)	0.37 (0.15)	0.42 (0.02)	0.11 (0.04)
MT_{HT}	0.208 (0.009)	<u>0.00</u> (0.00)	0.38 (0.08)	<u>0.00</u> (0.00)	0.45 (0.02)	<u>0.00</u> (0.00)
MT_{CV}	0.208 (0.005)	0.34 (0.03)	0.28 (0.02)	0.19 (0.08)	0.45 (0.01)	0.30 (0.06)
λ_S	0.147 (0.003)	0.43 (0.03)	0.49 (0.01)	0.50 (0.03)	0.36 (0.02)	0.46 (0.05)

Second Experiment: Benchmark on Peaking, Dipping, MNIST. Interestingly, for peaking and MNIST datasets any non-monotonicity (double descent [1]) in the *expected* learning curve almost completely disappears for λ_S , which tunes the regularization parameter using validation data (Fig. 2). We wonder if regularization can also help reducing the severity of double descent in other settings. For the dipping dataset, regularization doesn't help, showing that it cannot prevent non-monotone behaviour. Furthermore, the fraction of non-monotone decisions *per run* is largest for this learner (Table 1).

For the dipping dataset M_{CV} has a large advantage in terms of AULC. We hypothesize that this is largely due to tie breaking and small training set sizes due to the 5-folds. Surprisingly on the peaking dataset it seems to learn quite slowly. The expected learning curves of MT_{HT} look better than that of MT_{SIMPLE} , however, in terms of AULC the difference is quite small.

The fraction of non-monotone decisions for MT_{HT} per run is very small as guaranteed. However, it is interesting to note that this does not always translate to monotonicity in the expected learning curve. For example, for peaking and dipping the expected curve doesn't seem entirely monotone. But MT_{CV} , which makes many non-monotone decisions per run, still seems to have a monotone expected learning curve. While monotonicity of each individual learning curves guarantees monotonicity in the expected curve, this result indicates monotonicity of each individual curve may not be necessary. This raises the question: under what conditions do we have monotonicity of the expected learning curve?

General Remarks. The fraction of non-monotone decisions of MT_{HT} being so much smaller than α could indicate the hypothesis test is too pessimistic. Fagerland et al. [4] note that the asymptotic McNemar test can have more power, which could further improve the AULC. For this test the guarantee $P(p \leq \alpha | H_0) \leq \alpha$ can be violated, but in light of the monotonicity results obtained, practically this may not be an issue.

MT_{HT} is inconsistent at times, but this does not have to be problematic. If one knows the desired error rate, a minimum N_v can be determined that ensures the hypothesis test will not get stuck before reaching that error rate. Another possibility is to make the size N_v dependent on i : if N_v is monotonically increasing this directly leads to consistency of MT_{HT} . It would be ideal if somehow N_v could be automatically tuned to trade off sample size requirements, consistency and monotonicity. Since for CV N_v automatically grows and thus also directly implies consistency, a combination of MT_{HT} and MT_{CV} is another option.

Devroye et al. [2] conjectured that it is impossible to construct a consistent learner that is monotone in terms of the expected learning curve. Since we look at individual curves, our work does not disprove this conjecture, but some of the authors on this paper believe that the conjecture can be disproved. One step to make is to get to an essentially better understanding of the relation between individual learning curves and the expected one.

Currently, our definition judges any decision that increases the error rate, by however small amount, as non-monotone. It would be desirable to have a broader definition of non-monotonicity that allows for small and negligible increases of the error rate. Using a hypothesis test satisfying such a less strict condition could allow us to use less data for validation.

Finally, the user of the learning system should be notified that non-monotonicity has occurred. Then the cause can be investigated and mitigated by regularization, model selection, etc. However, in automated systems our algorithm can prevent any known and unknown causes of non-monotonicity (as long as data is i.i.d.), and thus can be used as a failsafe that requires no human intervention.

7 Conclusion

We have introduced three algorithms to make learners more monotone. We proved under which conditions the algorithms are consistent and we have shown for MT_{HT} that the learning curve is monotone with high probability. If one cares only about monotonicity of the expected learning curve, $\text{MT}_{\text{SIMPLE}}$ with very large N_v or MT_{CV} may prove sufficient as shown by our experiments. If N_v is small, or one desires that individual learning curves are monotone with high probability (as practically most relevant), MT_{HT} is the right choice. Our algorithms are a first step towards developing learners that, given more data, improve their performance in expectation.

Acknowledgments. We would like to thank the reviewers for their useful feedback for preparing the camera ready version of this paper.

References

1. Belkin, M., Hsu, D., Ma, S., Mandal, S.: Reconciling modern machine-learning practice and the classical bias-variance trade-off. *Proc. Nat. Acad. Sci.* **116**(32), 15849–15854 (2019)
2. Devroye, L., Györfi, L., Lugosi, G.: *A Probabilistic Theory of Pattern Recognition. Stochastic Modelling and Applied Probability.* Springer, Heidelberg (1996). <https://doi.org/10.1007/978-1-4612-0711-5>
3. Duin, R.: Small sample size generalization. In: *Proceedings of the Scandinavian Conference on Image Analysis*, vol. 2, pp. 957–964 (1995)
4. Fagerland, M.W., Lydersen, S., Laake, P.: The McNemar test for binary matched-pairs data: mid-p and asymptotic are better than exact conditional. *BMC Med. Res. Methodol.* **13**, 91 (2013). <https://doi.org/10.1186/1471-2288-13-91>
5. Hastie, T., Tibshirani, R., Friedman, J.: *The Elements of Statistical Learning.* SSS. Springer, New York (2009). <https://doi.org/10.1007/978-0-387-84858-7>
6. Huijser, M., van Gemert, J.C.: Active decision boundary annotation with deep generative models. In: *ICCV*, pp. 5286–5295 (2017)
7. Japkowicz, N., Shah, M.: *Evaluating Learning Algorithms: A Classification Perspective.* Cambridge University Press, Cambridge (2011)
8. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. *Proc. IEEE* **86**(11), 2278–2324 (1998)
9. Loog, M., Duin, R.: The dipping phenomenon. In: *S+SSPR*, Hiroshima, Japan, pp. 310–317 (2012)
10. Loog, M., Viering, T., Mey, A.: Minimizers of the empirical risk and risk monotonicity. In: *NeuRIPS*, vol. 32, pp. 7476–7485 (2019)
11. Mohri, M., Rostamizadeh, A., Talwalkar, A.: *Foundations of Machine Learning.* MIT Press, Cambridge (2012)
12. Oppor, M., Kinzel, W., Kleinz, J., Nehl, R.: On the ability of the optimal perceptron to generalise. *J. Phys. A: Math. General* **23**(11), L581 (1990)
13. O’Neill, J., Jane Delany, S., MacNamee, B.: Model-free and model-based active learning for regression. In: Angelov, P., Gegov, A., Jayne, C., Shen, Q. (eds.) *Advances in Computational Intelligence Systems.* AISC, vol. 513, pp. 375–386. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-46562-3_24
14. Raschka, S.: Model evaluation, model selection, and algorithm selection in machine learning (2018). arXiv preprint [arXiv:1811.12808](https://arxiv.org/abs/1811.12808)
15. Rifkin, R., Yeo, G., Poggio, T.: Regularized least-squares classification. *Nato Sci. Ser. Sub Ser. III Comput. Syst. Sci.* **190**, 131–154 (2003)
16. Settles, B., Craven, M.: An analysis of active learning strategies for sequence labeling tasks. In: *EMNLP*, pp. 1070–1079 (2008)
17. Shalev-Shwartz, S., Ben-David, S.: *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, Cambridge (2014)
18. Spigler, S., Geiger, M., D’Ascoli, S., Sagun, L., Biroli, G., Wyart, M.: A jamming transition from under- to over-parametrization affects loss landscape and generalization (2018). arXiv preprint [arXiv:1810.09665](https://arxiv.org/abs/1810.09665)
19. Viering, T., Mey, A., Loog, M.: Open problem: monotonicity of learning. In: *Conference on Learning Theory, COLT*, pp. 3198–3201 (2019)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

