arXiv:2010.03053v1 [cs.LG] 6 Oct 2020

Sequential Changepoint Detection in Neural Networks with Checkpoints

Michalis K. Titsias
DeepMind
mtitsias@google.com

Jakub Sygnowski
DeepMind
sygi@google.com

Yutian Chen
DeepMind
yutianc@google.com

Abstract

We introduce a framework for online changepoint detection and simultaneous model learning which is applicable to highly parametrized models, such as deep neural networks. It is based on detecting changepoints across time by sequentially performing generalized likelihood ratio tests that require only evaluations of simple prediction score functions. This procedure makes use of checkpoints, consisting of early versions of the actual model parameters, that allow to detect distributional changes by performing predictions on future data. We define an algorithm that bounds the Type I error in the sequential testing procedure. We demonstrate the efficiency of our method in challenging continual learning applications with unknown task changepoints, and show improved performance compared to online Bayesian changepoint detection.

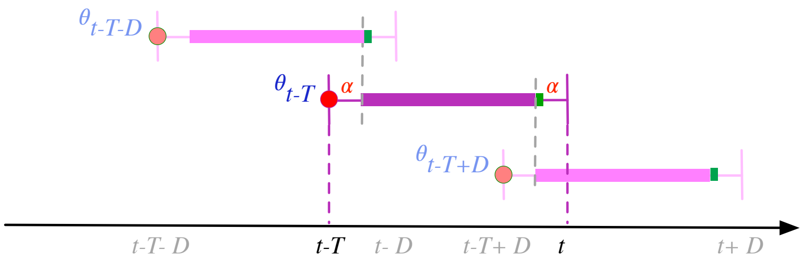
1 Introduction

Online changepoint detection is concerned with the problem of sequential detection of distributional changes in data streams, as soon as such changes occur. It can have numerous applications ranging from statistical process control, e.g. financial times series and medical conditioning monitoring (Hawkins et al., 2003; Bansal and Zhou, 2002; Aminikhanghahi and Cook, 2017; Truong et al., 2018), to problems in machine learning which can involve training very complex and highly parametrized models from a sequence of learning tasks (Ring, 1994; Robins, 1995; Schmidhuber, 2013; Kirkpatrick et al., 2017). In this latter application, referred to as continual learning, it is often desirable to train online from a stream of observations a complex neural network and simultaneously detect changepoints that quantify when a task change occurs.

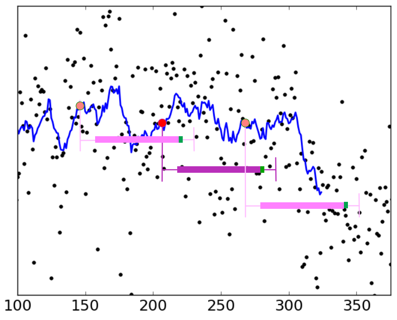
However, current algorithms for simultaneous online learning and changepoint detection are not well suited for models such as neural networks that can have millions of adjustable parameters. For instance, while state of the art Bayesian online changepoint detection algorithms have been developed (Fearnhead, 2006; Fearnhead and Liu, 2007; Adams and MacKay, 2007; Caron et al., 2012; Yildirim et al., 2013), such techniques can be computationally too expensive to use along with neural networks. This is because they are based on Bayesian inference procedures that require selecting suitable priors for all model parameters and they rely on applying accurate online Bayesian inference which is generally intractable, unless the model has a simple conjugate form. For instance, the popular techniques in Fearnhead and Liu (2007); Adams and MacKay (2007) are tested in simple Bayesian conjugate models where exact integration over the parameters is feasible. Clearly, such Bayesian computations are intractable or very expensive for highly non-linear models such as neural networks, which can contain millions of parameters.

In this article, we wish to develop a framework for joint sequential changepoint detection and online model fitting, that could be easily applied to arbitrary systems and it will be particularly suited for highly parametrized models such as neural networks. The key idea we introduce is to sequentially perform statistical hypothesis testing by evaluating predictive scores under cached model checkpoints. Such checkpoints are periodically-updated copies of the model parameters and they are used to detect distributional changes by performing predictions on future/unseen data (relative to the checkpoint), i.e. on data observed after a checkpoint and up to present time. An illustration of the approach is given by Fig. 1, while detailed description of the method is given in Section 3 and Algorithm 1. In statistical testing for change detection we use generalized likelihood ratio tests (Csorgo and Horváth, 1997; Jandhyala et al., 2002) where we bound the Type I error (false positive detection error) during the sequential testing process. The overall algorithm is easy to use and it requires by the user to specify two main hypeparameters: the desired bound on the Type I error and the testing window size between a checkpoint and the present time.

We demonstrate the efficiency of our method on time series as well as on continual learning of neural networks from a sequence of supervised tasks with unknown task changepoints. For these challenging continual learning problems we also consider a strong baseline for comparison, by using a variant of the Bayesian online changepoint detec-



(a)



(b)

Figure 1: (**a**) Visualization of Algorithm 1. Model checkpoints are cached parameters stored (and deleted) periodically every $D = T - 2\alpha$ time steps: $\theta_{t-T-D}, \theta_{t-T}, \theta_{t-T+D}$. Every D steps (e.g. $t - D$, t, $t + D$) the checkpoint that lags T iterations performs the statistical test. E.g. given current time t the active checkpoint is θ_{t-T} (see highlighted magenta segment) which tests for a changepoint in the region $(t - T + \alpha, t - \alpha]$ where α is the minimum sample size in each segment; see Section 3.2. (**b**) An example in a time series dataset (black dots) where the model is a moving average parameter shown by the blue line and the checkpoints are cached values of the moving average indicated by the coloured circles.

tion algorithm by Adams and MacKay (2007) that is easily applicable to complex models. This is done by applying the Bayesian algorithm on data that correspond to predictive scores provided by the neural network during online training. Our proposed method consistently outperforms this and others baselines (see Section 6), which shows that model checkpoints provide an easy to use and simultaneously effective technique for changepoint detection in online learning of complex models.

The paper is organized as follows. Section 2 introduces the problem of changepoint detection and online model learning. Section 3 develops our framework for changepoint detection using checkpoints and Section 4 considers applications to continual learning with neural networks. Section 5 discusses related work, while Section 6 provides numerical experimental results and comparisons with other methods. The Appendix contains further details about the method and additional experimental results.

2 Problem Setup

2.1 Streaming Data with Unknown Changepoints

We consider the online learning problem with unknown changepoints in a stream of observations $\{y_t\}_{t \geq 0}$. Each y_t includes an input vector and possibly additional outputs such as a class label or a real-valued response. For instance, in a supervised learning setting each observation takes the form $y_t \equiv (x_t, c_t)$ where x_t is the input vector and c_t the desired response such as a class label, while in unsupervised learning y_t is an input vector alone, i.e. $y_t \equiv x_t$. In addition, for many applications, e.g. in deep learning (LeCun et al., 2015), y_t can be a small set or mini-batch of individual i.i.d. observations, i.e. $y_t = \{y_t^i\}_{i=1}^b$, that are received simultaneously at time t.

In the generation process of $\{y_t\}_{t \geq 0}$ we assume that there exist certain times, referred to as *changepoints* and denoted by $\{\tau_k\}_{k=1,2,\ldots}$, that result in abrupt changes in data distribution so that $y_{t \in [0,\tau_1)} \overset{iid}{\sim} \mathscr{P}_1, y_{t \in [\tau_1,\tau_2)} \overset{iid}{\sim} \mathscr{P}_2$ and in general

$$y_{t \in [\tau_{k-1},\tau_k)} \overset{iid}{\sim} \mathscr{P}_k, \quad k = 1, 2, \ldots \tag{1}$$

where $\mathscr{P}_{k-1} \neq \mathscr{P}_k$ and with the convention $\tau_0 = 0$. Each $\mathscr{P}_k$ is the segment or task-specific distribution that generates the k-th data segment. These assumptions are often referred to as partial exchangeability or the product partition model (Barry and Hartigan, 1992). To learn from such data we wish to devise schemes that can adapt online a parametrized model without knowing the changepoints τ_k and the distributions $\mathscr{P}_k$. Accurate sequential detection of changepoints can be useful since, knowing them, the learning system can dynamically decide to switch to a different parametric model or add new parameters to a shared model and etc. In Section 3, we introduce a general online learning and changepoint detection algorithm suitable for arbitrary models ranging from simple single-parameter models to complex deep networks having millions of parameters.

2.2 Online Model Learning with Changepoints

We consider a probabilistic model $p(y|\theta)$ with parameters θ that we wish to train online and simultaneously use it to detect the next changepoint τ_k. Online training of θ means that for each observation y_t we perform, for instance, a gradient update

$$\theta_t \leftarrow \theta_{t-1} - \rho_t \nabla \ell(y_t; \theta_{t-1}), \tag{2}$$

where ρ_t is the step size or learning rate. Given the non-stationarity of the learning problem this sequence should not satisfy the Robbins-Monro conditions (Robbins and Monro, 1951) and, for instance, ρ_t could be constant through time. The loss function in Eq. (2) is typically the negative log-likelihood function, i.e.

$$\ell(y_t; \theta_{t-1}) = -\log p(y_t|\theta_{t-1})$$

and θ_t denotes the parameter values after having seen t observations including the most recent y_t. We will refer to the evaluations of the loss $\ell(y_{t'}, \theta_{t-T})$, or any other score function $v(y_{t'}, \theta_{t-T})$ on any future data $y_{t'}$ (relative to θ_{t-T}), with $t' > t-T$, as *prediction* scores. Notice that given $y_{t'}$s have been drawn i.i.d. from the same data segment, the prediction scores are also i.i.d. random variables.

Suppose at time t the data segment or *task* is k and we observe y_t, $t \geq \tau_{k-1}$ where τ_{k-1} is the most recently detected changepoint, i.e. when the k-th task started as shown in Eq. (1). If we decide that data y_t comes from a new task $k+1$, we could set $\tau_k = t$, instantiate a new model with a fresh set of parameters $\theta^{(k+1)}$ and repeat the process. All these models could have completely separate parameters, i.e. $\theta^{(k)} \cap \theta^{(k')} = \emptyset, \forall k \neq k'$ or allow parameter sharing, i.e. $\theta^{(k)} \cap \theta^{(k')} \neq \emptyset$, as further discussed in Section 4 where we describe applications to continual learning.

3 Changepoint Detection with Checkpoints

Throughout this Section we will be interested to detect the next changepoint τ_k. Thus, to simplify notation we will drop index k and write this changepoint as τ and the current parameters as θ when it does not cause confusion. We will also assume that the previously detected changepoint is at time zero.

The iterative procedure for changepoint detection with checkpoints is illustrated in Fig. 1. This algorithm assumes that together with the current parameter values θ_t we cache in memory one or multiple copies of early values of the parameters referred to as model checkpoints or simply checkpoints. Checkpoints are cached and deleted periodically and statistical testing for changepoint detection is also performed periodically. When at iteration t, where model parameters are θ_t, we need to perform a changepoint detection test the algorithm activates the checkpoint θ_{t-T}, that has been cached T iterations before. This checkpoint forms predictions on all subsequent data (not seen by the checkpoint) in order to detect a change in the data distribution that possibly occurs in the data segment in $(t-T, t]$. Pseudo-code of the algorithm is provided in Algorithm 1.

In the remaining of this Section we will be discussing in detail how Algorithm 1 works. Some useful summarizing remarks to keep in mind are the following. The algorithm caches checkpoints every $D = T - 2\alpha$ iterations with the first checkpoint cached at $t = 0$. T is the window size, D is the stride and $\alpha > 0$ is the minimum sample size when computing the testing statistics. The first testing occurs at time $t = T$, i.e. when the data buffer $\mathscr{B}_t$ becomes full and the first cached checkpoint used is the initial parameter values θ_0. This constrains also the minimum size of the data segment (i.e. the distance between two consecutive changepoints) to be T. After the first test, testing occurs every D iterations and given that each checkpoint is deleted after a test the number of checkpoints in memory is roughly T/D.

3.1 Offline Changepoint Detection in a Window

Suppose a sliding window of observations $y_{t'}, t-T < t' \leq t$ (recall that $y_{t'}$ can generally be a set or mini-batch of b i.i.d. individual observations) of size T, i.e. all data observed strictly after the model checkpoint θ_{t-T}. Given a scalar prediction score function:

$$v(y_{t'}, \theta_{t-T}) = \frac{1}{b}\sum_{i=1}^{b} v(y^i_{t'}, \theta_{t-T}),$$

or $v_{t'}$ for short, we consider the offline changepoint detection problem in the interval $(t-T, t]$ with the independent and normal distribution assumption:

Assumption 1. *$v_{t'} \sim \mathscr{N}(\mu_{t'}, \sigma^2_{t'})$ independently for all t'.*

We consider the following hypothesis testing problem with unknown change time, mean and variance:

- $\mathscr{H}_0 : \exists \mu, \sigma^2$ s.t. $\mu_{t'} = \mu, \sigma^2_{t'} = \sigma^2, \forall t' \in (t-T, t]$.
- $\mathscr{H}_1 : \exists \tau \in (t-T+\alpha, t-\alpha], \mu_1, \sigma^2_1, \mu_2, \sigma^2_2$ s.t.
  $\mu_{t'} = \mu_1, \sigma^2_{t'} = \sigma^2_1, \ \forall t' \in (t-T, \tau)$,
  $\mu_{t'} = \mu_2, \sigma^2_{t'} = \sigma^2_2, \ \forall t' \in [\tau, t]$,

where $\alpha \in (0, T/2)$ is the minimum sample size in each segment of $\mathscr{H}_1$ for estimating the mean and variance. Using a model checkpoint and applying the testing on predictions is important to satisfy the independent assumption on scores.

Following the generalized likelihood ratio (GLR) test, we denote by Λ_τ the likelihood ratio of the two hypotheses at a changepoint of τ with the unknown variables taking the maximum likelihood estimates,

$$\Lambda_\tau = \frac{\sup_{\mu,\sigma^2} p(\mathbf{v}_{(t-T,t]}|\mu,\sigma^2)}{\sup_{\mu_1,\sigma_1^2} p(\mathbf{v}_{(t-T,\tau)}|\mu_1,\sigma_1^2)\,\sup_{\mu_2,\sigma_2^2} p(\mathbf{v}_{[\tau,t]}|\mu_2,\sigma_2^2)}, \tag{3}$$

and compute the statistics as follows:

$$\begin{aligned}Z &= \max_{\tau\in(t-T+\alpha,t-\alpha]} (-2\log\Lambda_\tau) \\ &= \max_{\tau\in(t-T+\alpha,t-\alpha]} \big\{T\log S(\mathbf{v}_{(t-T,t]}) \\ &\quad -(\tau-t+T-1)\log S(\mathbf{v}_{(t-T,\tau)}) \\ &\quad -(t-\tau+1)\log S(\mathbf{v}_{[\tau,t]})\big\}, \qquad (4)\end{aligned}$$

where S is the sample variance, $\mathbf{v}_{(t-T,\tau)}$ denotes the set of all values $v_{t'}, t-T<t'<\tau$, $\mathbf{v}_{[\tau,t]}$ the values $v_{t'}, \tau\le t'\le t$ and $\mathbf{v}_{(t-T,t]}$ their union.

Asymptotic distribution of the statistics Z as $T\to\infty$ has been well studied in the literature for the normal distribution of $v_{t'}$ (Csorgo and Horváth, 1997; Jandhyala et al., 2002). For a finite window size T, we can also compute the critical region, $Z>h(\delta)$ at a given confidence level $1-\delta$ numerically; see the Appendix. When the null hypothesis is rejected, we claim there is a changepoint in the current window $(t-T,t]$, and the changepoint is selected with $\tau=\arg\max_{\tau'}(-2\log\Lambda_{\tau'})$.

It is important to note that the alternative hypothesis $\mathscr{H}_1$ is not a complement of the null, and we consider the candidate changepoint τ in a subset $(t-T+\alpha,t-\alpha]\subset(t-T,t]$ for reliable estimate of sample mean and variance. This means that when a true changepoint exists in the right border $[t-\alpha+1,t]$, it might cause a rejection and show up in the nearest location on the subset, i.e. $t-\alpha$, which subsequently could increase the error of the changepoint location estimation. To reduce this effect we can compute Λ_τ in the extended subset $(t-T+\alpha,t-\alpha+1]$, and do not reject $\mathscr{H}_0$ if $Z<-2\log\Lambda_{t-\alpha+1}$ (Z is still taken in $(t-T+\alpha,t-\alpha]$). Notice that there are α samples in the right side when $t'=t-\alpha+1$, satisfying our requirement for the minimum sample size.

We repeat the offline detection using a sliding window of size T with a stride $D=T-2\alpha$. This ensures that every time location will be included in the candidate subset for exactly one test. An illustration of this is shown in Fig. 1 where the green border is precisely the location $t-\alpha+1$ in the extended subset, which is ignored if the maximum occurs there, but it could be accepted in the next iteration where the green location becomes the first location in the new subset. Similarly, the possibility that a changepoint occurs in the left border $(t-T,t-T+\alpha]$ can be detected in a previous window.

3.2 Online Changepoint Detection across Windows

As the interval between two changepoints spans over multiple test windows, we would like to control the overall error δ of making a false rejection of the null hypothesis, that is making a false claiming that data distribution changes, in every data segment. Since the model checkpoints change at

Algorithm 1 Changepoint detection with checkpoints

```
**Procedure:** changepoint_detection($\theta_0,\alpha,T,\delta,\eta$, update_step)
**Input:** Initial parameters $\theta_0$, minimal sample size $\alpha$, window size $T$, error schedule parameters $\{\delta,\eta\}$, optimization step function update_step
**Output:** Changepoint location $\tau^*$, model parameter $\theta$
Initialize: test region size $D=T-2\alpha$, test index $i=0$, time step $t=0$, data buffer $\mathscr{B}_0=\emptyset$.
for $t=1,2,\dots$ do
  Receive mini-batch $y_t$, update data buffer $\mathscr{B}_t=\{y_{t'}:\max(0,t-T)<t'\le t\}$
  $\theta_t=$ update_step$(\theta_{t-1},y_t)$
  if $t$ mod $D=0$ then cache checkpoint $\theta_t$
  if $t=iD+T$ then
    Compute scores with cached checkpoint $\theta_{t-T}$:
      $v_{t'}\equiv v(y_{t'},\theta_{t-T}), t-T<t'\le t$
    $\delta_i=(1-\eta)\eta^i\delta$
    (reject, $\tau^*$) = offline_detection($\mathbf{v}_{(t-T,t]},\alpha,\delta_i$)
    if reject then
      return $(\tau^*,\theta_t)$
    end if
    $i\leftarrow i+1$, delete checkpoint $\theta_{t-T}$
  end if
end for
```

Algorithm 2 Offline changepoint detection

```
**Subroutine:** offline_detection($\mathbf{v},\alpha,\delta$)
**Input:** Scores $\mathbf{v}_{(t-T,t]}$, minimal sample size $\alpha$, error $\delta$
**Output:** Boolean reject, candidate location $\tau^*$
Compute the threshold $h=$ quantile$(1-\delta)$
for $t'=t-T+\alpha+1,\dots,t-\alpha+1$ do
  Compute $\Lambda_{t'}$ using Eq. (3)
end for
Compute $Z$ using (4), $\tau^*=\underset{t-T+\alpha<\tau\le t-\alpha}{\text{argmax}}(-2\log\Lambda_\tau)$
reject = $Z>h$ and $Z>-2\log\Lambda_{t-\alpha+1}$
return reject, $\tau^*$
```

every test window, it is difficult to apply a standard sequential likelihood ratio test across windows. Instead, we select the confidence level with an annealing schedule so that the overall error is bounded:

$$\delta_i=(1-\eta)\eta^i\delta, \qquad (5)$$

where δ_i is the confidence level for the i-th (0-based) test window after a new task is detected and η is the decaying rate.

Proposition 1. *Given Assumption 1 holds, the probability of making a Type I error (false changepoint detection) by Algorithm 1 between two real changepoints is upper bounded by δ.*

Proof. Let $y_1,y_2,\dots,y_N$ be the segment of data stream with the same distribution $\mathscr{P}$ where N is the time of last mini-batch of data, $y_N=\{y_N^i\}_{i=1}^b$. Offline change point detection is conducted in windows $(0,T]$, $(D,T+D]$, $(2D,T+2D]$,

..., $(nD, T+nD]$ before a changepoint occurs, where n is the maximum integer with $T+nD \le N$. Under Assumption 1, the probability of rejecting the null hypothesis at the i-th testing window with input error argument δ_i is

$$Pr(\text{reject}_i) \le Pr(Z > h(\delta_i)) = \delta_i, \tag{6}$$

where the first inequality is due to the possibility to ignore the rejection when $Z \le -2\log\Lambda_{t-\alpha+1}$, and the second equality follows the definition of h in Algorithm 2 as the $1-\delta$ quantile.

The probability that the null hypothesis is rejected in at least one testing window is then upper bounded with the union bound by

$$\begin{aligned} Pr(\cup_{i=0}^{n}\{\text{reject}_i\}) &\le \sum_{i=0}^{n} Pr(\text{reject}_i) = \sum_{i=0}^{n}\delta_i \\ &= \sum_{i=0}^{n}(1-\eta)\eta^i\delta < \delta. \end{aligned} \tag{7}$$

□

The time complexity of running Algorithm 1 on a datastream of length t is $O(tbT/D)$ where b is the mini-batch size and $D = T - 2\alpha$ is the stride of the sliding window, and the space complexity for storing the checkpoint and data buffer is $O(T/DS_m + TbS_y)$ where S_m and S_y denotes the size of a model checkpoint and a data point.

3.3 Setting the Hyperparameters and Prediction Scores

There are a few hyperparameters in our proposed algorithm, including the window size T, minimum sample size in a window α, Type I error δ, and the error decaying factor η.

A large window size provides more data for every offline detection and usually leads to a higher accuracy. However, the space complexity increases linearly as the window size in order to keep a data buffer of size T, and it has to be upper bounded by our prior assumption on the minimum distance between two consecutive changepoints. Also, when the score function $v(y_{t'}, \theta_{t-T})$ requires a good model fit in order to be discriminative between tasks, a smaller window size can be beneficial at the beginning of a new task because it would update the model checkpoint more frequently and thereby improve the discriminative power in detecting a changepoint more quickly. We study the effect of T empirically in our experiments.

A sufficiently large minimum sample size α is important to obtain reliable estimate of the sample variance and stabilize the distribution of the statistics Z. But too large value in α reduces the range of candidate locations and decreases the power in a single offline detection test. Also, because the sliding window has a stride of $D = T - 2\alpha$, the time complexity increases with α. In our experiments, we use a default value $\alpha = \lfloor T/4 \rfloor$, giving $D \approx T/2$. Notice that with such default settings only two checkpoints are needed to be kept in memory (since $T/D = 2$) resulting in small memory cost.

Given a total Type I error δ, the decaying factor η controls the exponential distribution of the error across windows with mean $D/\log(1/\eta)$. In principle, we would like the mass of the error to be distributed in the support of our prior about the changepoint frequency. In lack of this knowledge, we use $\eta = 0.99$ in all the experiments.

The prediction score function $v(y_{t'}, \theta_{t-T})$ must be discriminative with respect to data streams from different tasks. When the model parameter θ_t is well fitted to the current task, a properly chosen score function is usually sensitive to the change of the task. Nevertheless, we emphasize that being fitted to the current task is not a necessary condition for our changepoint detection method to work. As demonstrated in the example of Section 6.1, our algorithm in some cases can detect the changepoints robustly regardless of the learning rate in the update rule in Eq. (2) that affects how well the model is fitted to the data.

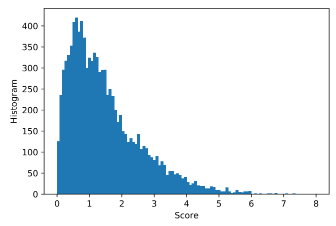
A key assumption about our detection algorithm is the normal distribution of the score function defined on every mini-batch of data. In experiments with continuous observations we use the average negative log-likelihood as score $v_{t'} = 1/b\sum_{i=1}^{b} -\log p(y_{t'}^i|\theta_{t-T}))$, and in experiments with discrete observations, we find it works better by applying another logarithm operation as

$$v_{t'} = 1/b\sum_{i=1}^{b}\log(-\log Pr(y_{t'}^i|\theta_{t-T}) + \varepsilon),$$

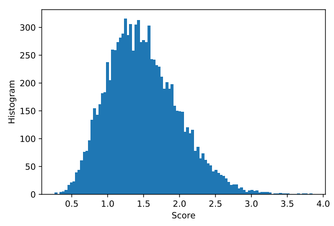
where $\varepsilon > 0$ is a small jittering term for numerical stability. Fig. 2 shows typical histograms of scores in a testing window from our continual learning experiments with real-world data; see Section 6. We also apply the D'Agostino and Pearson's normality test (d'Agostino Ralph B, 1971; D'Agostino and Pearson, 1973) on a sample of 100 scores in this setting and show the p-value in the caption of each plot. It is clear that the normality improves with a larger size b of mini-batch due to the central limit theorem, and the distribution of scores in the log-domain is closer to a normal distribution. We show in the experiments that the performance of our detection improves significantly with the mini-batch size.

4 Application to Continual Learning

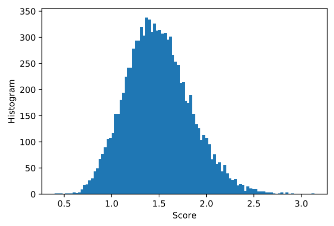
To test our method on a challenging online model fitting and changepoint detection problem we consider continual learning (CL) (Ring, 1994; Robins, 1995; Schmidhuber, 2013; Goodfellow et al., 2013), which requires training neural networks on a sequence of tasks. Many recent CL methods, see e.g. (Kirkpatrick et al., 2017; Nguyen et al.,



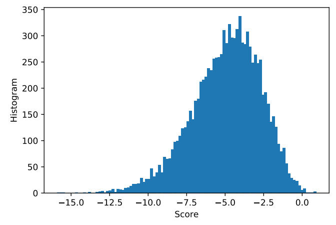
(a) Batch size = 10, p-value = 0.002.



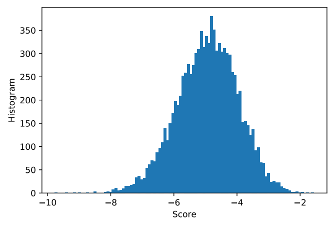
(b) Batch size = 50, p-value = 0.001.



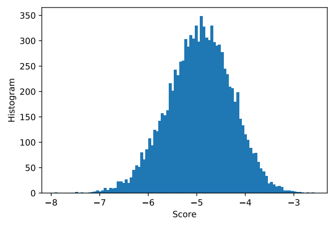
(c) Batch size = 100, p-value = 0.32.



(d) Batch size = 10, p-value = 0.42.



(e) Batch size = 50, p-value = 0.93.



(f) Batch size = 100, p-value = 0.29.

Figure 2: Typical histogram of scores in the Split-MNIST experiment when no changepoint occurs. Top row use the score of the mean negative log-likelihood, and bottom row applies another logarithm transformation. The mini-batch size from left to right is 10, 50 and 100 respectively. p-value of the normality test based on 100 samples is shown in the subfigure caption.

2017; Rusu et al., 2016; Li and Hoiem, 2017; Farquhar and Gal, 2018), typically assume known task changepoints, also called task boundaries. Instead, here we wish to train a CL method without knowing the task boundaries and investigate whether we can accurately detect the changepoint locations that quantify when the data distribution is changing from one task to the next. The sequential learning and changepoint detection Algorithm 1 can be easily incorporated to existent CL algorithms, since the essential component of the algorithm is the prediction score function $v(\cdot)$ used in hypothesis testing. In the following, we combine our algorithm with a standard experience replay CL method (Robins, 1995; Robins and McCallum, 1998; Lopez-Paz et al., 2017; Rebuffi et al., 2017), which regularizes stochastic gradient descent model training for the current task by replaying a small subset of observations from previous tasks, as detailed next.

As the main structure for the CL model we consider a feature vector $\phi(x;\theta^s) \in \mathbb{R}^M$, obtained by a neural network with parameters θ^s, where these parameters are shared across all tasks. For each detected k-th task there is a set of task-specific or private parameters $\theta^{p,k}$, which are added dynamically into the model each time our algorithm returns a detected changepoint indicating the beginning of a new task. In the experiments, we consider CL problems where each task is a binary or multi-class classification problem, so that each $\theta^{p,k}$ is a different head, i.e. a set of final output parameters, attached to the main network consisted of the feature vector $\phi(x;\theta^s)$. For instance, if the task is multi-class classification, then $\theta^{p,k}$ is a matrix of size $C \times M$, where C denotes the number of classes. In this case, such task-specific parameter allows the computation of the softmax or multinomial logistic regression likelidood

$$p(c|x,k) = \frac{\exp\{\sum_{m=1}^{M} [\theta^{p,k}]_{cm} \phi_m(x;\theta^s)\}}{\sum_{c'=1}^{C} \exp\{\sum_{m=1}^{M} [\theta^{p,k}]_{c'm} \phi_m(x;\theta^s)\}},$$

that models the categorical probability distribution for classifying input data points from the k-th task.

We assume that the CL model is continuously trained so that tasks occur sequentially and they are separated by random changepoints. For simplicity, we also assume that previously seen tasks never re-occur. At time $t = 0$, the shared parameters θ^s are initialized to some random value, and the parameters $\theta^{p,1}$ of the first task are also initialized arbitrarily, e.g. to zero or randomly. Then, learning of the model parameters progresses so that whenever a changepoint τ_k occurs a fresh set of task-specific parameters $\theta^{p,k}$ is instantiated while all existing parameters, such the shared parameters θ^s, maintain their current values, i.e. they are not re-initialized. However, this continuous updating can cause the shared feature vector $\phi(x;\theta^s)$ to yield poor predictions on early tasks, a phenomenon known in the literature of neural networks as catastrophic forgetting (Robins, 1995; Goodfellow et al., 2013; Kirkpatrick et al., 2017), and the prevention of this is one major challenge CL methods need to deal with.

More specifically, at each time instance the model receives a mini-batch of training examples $y_t = \{c_t^i, x_t^i\}_{i=1}^b$, where c_t^i is a class label and x_t^i is an input vector. At each time step the current detected task is k, so that in the shared feature vector $\phi(x;\theta^s)$ we have attached so far k heads each with task-specific parameters $\theta^{p,i}$, $i = 1,\dots,k$. The full set of currently instantiated parameters is denoted by $\theta^{(k)} = (\theta^s, \{\theta^{p,i}\}_{i=1}^k)$ to emphasize the dependence on the k-th task. Training with the current k-th task is performed by using the standard negative log-likelihood, i.e. cross-entropy loss, regularized by adding a sum of *replay buffers*, which correspond to negative log-likelihoods terms evaluated at small data subsets from all previous tasks,

$$L(\theta^{(k)}) = L_k(\{c_t^i, x_t^i\}_{i=1}^b; \theta^{p,k}, \theta^s) + \lambda \sum_{i=1}^{k-1} L_i(\mathscr{R}_i; \theta^{p,i}, \theta^s), \tag{8}$$

where $\lambda > 0$ is a regularization parameter and each $L_i(\cdot)$ is a sum of negative log-likelihood terms over the individual data points. Each $\mathscr{R}_i$ is a small random subset of data from the i-th task that is stored as soon as this task is detected and then used as an experience replay (to avoid forgetting the i-th task) when training in future tasks.

Pseudo-code of the whole procedure for training the CL model with simultaneous changepoint detection based on our checkpoint framework is outlined in Algorithm 3. For simplicity in Algorithm 3 we assumed that the replay buffers $\mathscr{R} = \{\mathscr{R}_i\}_{i=1}^{k-1}$ are global variables that affect the subroutine cl_update_step without having to be passed as inputs. A second simplification is that each task replay buffer $\mathscr{R}_k$ in practice is actually created inside Algorithm 1, where a few data mini-batches of the current task are stored into the fixed-size memory to form $\mathscr{R}_k$.[1]

Finally, an interesting aspect of using checkpoints for changepoint detection in CL is that once the next changepoint τ_k is detected, and thus we need to instantiate a new task parameters $\theta^{p,k+1}$, we can reuse one of the checkpoints to avoid the full set of model parameters $\theta_t \equiv (\theta^s, \{\theta^{p,i}\}_{i=1}^k)$ being contaminated by training updates using data from a new task in iterations $t' \in [\tau_k, t]$, without knowing yet the task change. Specifically, we can re-set this full parameter vector to the nearest checkpoint that exists on the left of the changepoint location τ_k. This allows the checkpoint to act as a *recovery state* that can mitigate forgetting of the current k-th task parameters caused by these extra updates, i.e. for $t' \in [\tau_k, t]$.

Algorithm 3 Continual learning

```
Procedure: continual_learning
Input: Initial shared model parameter $\theta^s$, parameters for changepoint_detection: $\alpha$, $T$, $\{\delta, \eta\}$
Output: Model parameters $\Theta$, list of changepoints T
Initialize: list of parameters $\Theta = \theta^s$, list of replay buffers $\mathscr{R} = []$, and list of changepoints T = []
for $k = 1, 2, \dots$ do
    Initialize task private parameter $\theta^{p,k}$
    Concatenate all current parameters: $\theta^{(k)} = (\theta^s, \{\theta^{p,i}\}_{i=1}^k)$
    $\tau^*, \theta^{(k)}$ = changepoint_detection($\theta^{(k)}, \alpha, T, \delta, \eta$, cl_update_step)
    Append $\tau^*$ to list T
    Construct a task replay buffer $\mathscr{R}_k$ and append it to the list $\mathscr{R}$
    $\Theta = \theta^{(k)}$
end for
return $(\Theta, \mathrm{T}, \mathscr{R})$

Subroutine: cl_update_step($\theta^{(k)}, y_t$)
Input: Full set of model parameters $\theta^{(k)}$, data mini-batch $y_t$,
Output: Updated model parameters $\theta^{(k)}$
$\theta^{(k)} \leftarrow \theta^{(k)} - \rho_t \nabla L(\theta^{(k)})$, where $L$ is from Eq. (8)
return $\theta^{(k)}$
```

5 Related Work

Changepoint detection methods are categorized into offline and online settings (Aminikhanghahi and Cook, 2017; Truong et al., 2018). Offline algorithms such as the recent linear time dynamic programming algorithms (Killick et al., 2012; Maidstone et al., 2017) operate similarly to the Viterbi algorithm in hidden Markov models (Bishop, 2006), where they need to observe the full data sequence in order to retrospectively identify multiple changepoints. In contrast, in online changepoint detection the aim is to detect a change as soon as it occurs while data arrive online. Online detection has a long history in statistical process control (Page, 1957; Hawkins et al., 2003) where typically we want to detect a change in a mean parameter in time series. More recently, Bayesian online changepoint detection methods have been developed in (Fearnhead, 2006; Fearnhead and Liu, 2007; Adams and MacKay, 2007), that consider conjugate exponential family models and online Bayesian updates. These latter techniques can be extended to also allow online point estimation of some model parameters (Caron et al., 2012; Yildirim et al., 2013), but they remain computationally too expensive to use in deep learning where models consist of neural networks. This is because they are based on Bayesian inference procedures that require selecting suitable priors over model parameters and they rely on applying accurate online Bayesian inference which is generally intractable, unless the model has a simple conjugate form. Also approximate inference can be too costly and inaccurate for highly non-linear and parametrized models.

The method we introduced differs from these previous approaches, since it relies on the idea of a checkpoint which allows to detect changepoints by performing multistep ahead predictions. This setup provides a stream of 1-dimensional numbers with a simple distribution on which we can apply standard statistical tools to detect whether there exists an abrupt change in a window of these predictions. The checkpoint is updated over time by tracking slowly (within a distance T) the actual model, which can

[1]This second simplification was made to keep the structure of Algorithm 1 in its general form, while the minor modification regarding the replay buffers is only needed for this specific CL application.

improve the discriminative power overtime as the task persists and the checkpoint becomes more specialized to the task distribution. The method can be considered as a combination of offline and online detection (Aminikhanghahi and Cook, 2017) since, while model parameter learning is online, each testing with a checkpoint involves an offline subroutine; see Algorithm 2.

More distantly related work is from the recent continual learning literature such as the so-called task-free or task-agnostic methods (Aljundi et al., 2018, 2019; Kaplanis et al., 2018; Zeno et al., 2018; Rao et al., 2019) that learn without knowing or assuming task boundaries. However, the objective there is typically not to explicitly detect changepoints, but instead to maintain an overall good predictive performance, by avoiding catastrophic forgetting of the neural network model. In contrast, our method aims to explicitly detect abrupt changes in arbitrary online learning systems, either traditional few-parameter models or neural networks used in continual learning. As we discussed in Section 4 and will demonstrate next in Section 6 our algorithm can be combined with existent continual learning methods and enhance them with the ability of changepoint detection.

6 Experiments

6.1 Time Series Example

Fig. 3 shows online changepoint detection on an artificial time series dataset, a small snapshot of which was used in the illustrative example in Fig. 1(b). The task is to track a data stream of 1-dimensional noisy observation (each y_t is a scalar value) with abrupt changes in the mean. The model has a single parameter θ: a moving average that is updated as data arrive online, where the underlying loss is $0.5(y_t - \theta)^2$ (which up to a constant is the negative log-likelihood of a normal distribution with a fixed variance). Fig. 3 shows changepoint detection achieved by the proposed algorithm. The panel on the top row shows the data, the moving average parameter and the testing windows together with the corresponding checkpoints that lead to all seven detections. The panel on the bottom row shows the GLR statistics, $-2\log\Lambda_\tau$, computed through time which clearly obtains maximal values at the changepoint locations. The window size was $T = 50$, $\alpha = \lfloor T/4 \rfloor = 12$ and $\delta = 10^{-3}$. All the changepoints are detected by our algorithm without a false alarm. Every changepoint corresponds to a clear spike in the GLR statistics significantly higher than the normal range of values.

We also study the impact of a sub-optimal choice of learning rate of the tracking model to our changepoint detection algorithm in Fig. 4. As discussed in Section 3.3, as long as the score function, negative log-likelihood in this example, is able to differentiate between different tasks the changepoints can still be robustly detected regardless of whether the model is under-fit or over-fit to the data. We should point out, however, that in more complex models might not be feasible to come up with such discriminate scores. For example, a neural network with random weights most likely will not be so discriminative, i.e. it shall provide similar (random) predictions for data coming from different tasks. Nevertheless, with reasonable training the neural network can become more specialized to a certain task and provide predictions that can significantly differ from those associated with data from other tasks, that the network hasn't trained on. Therefore, in more complex models the learning rate needs to be chosen carefully to allow quick adaptation to the task data so that the score function, computed under checkpoints, can become more discriminative of task changes.

6.2 Experiments on Continual Learning

In all CL experiments throughout this section the proposed Algorithm 1, checkpoint-based changepoint detection (CheckpointCD) is applied, in conjunction with Algorithm 3, with the following settings:

$$\delta = 10^{-4}, \eta = 0.99, T = 100, \alpha = \lfloor T/4 \rfloor = 25.$$

Note that $\eta = 0.99$ and $\alpha = \lfloor T/4 \rfloor$ are default values, while δ and T were specified by few preliminary runs on one of the datasets (Split-MNIST). I.e. the cutoff value of the Type I error was set to $\delta = 10^{-4}$ to maximize performance (Jaccard index) on Split-MNIST while for all remaining experiments the same cutoff is used and is never re-optimized. The effect of the window size T is also analyzed in Fig. 6.

As a strong baseline for comparison we consider Bayesian online changepoint detection (BayesCD) by Adams and MacKay (2007); see also Fearnhead and Liu (2007). We define an instant of this method that is fully applicable to complex models such as deep neural networks. This is expressed by treating the one-step predictive scores $v_t = v(y_t, \theta_{t-1})$ (averaged over the mini-batch at time t so they are close to normality) as sequential observations following a univariate Gaussian, $y_t \sim \mathcal{N}(y_t|\mu, \sigma^2)$, where the parameters (μ, σ^2) are task-specific. Then, the algorithm detects when (μ, σ^2) undergoes an abrupt change, by performing full Bayesian inference and estimating recursively the marginal posterior probability of each time being a changepoint, i.e. the so-called task or run length value to return to zero value (Adams and MacKay, 2007). This involves placing a conjugate normal inverse-gamma prior on (μ, σ^2)[2]:

$$(\mu, \sigma^2) \sim \mathcal{N}(\mu|\mu_0, \sigma^2/\kappa) IG(\sigma^2|\alpha, \beta),$$

[2]The values of the hyperparameters where chosen as $\mu_0 = 0, \alpha = 0.1, \beta = \kappa = 1$.

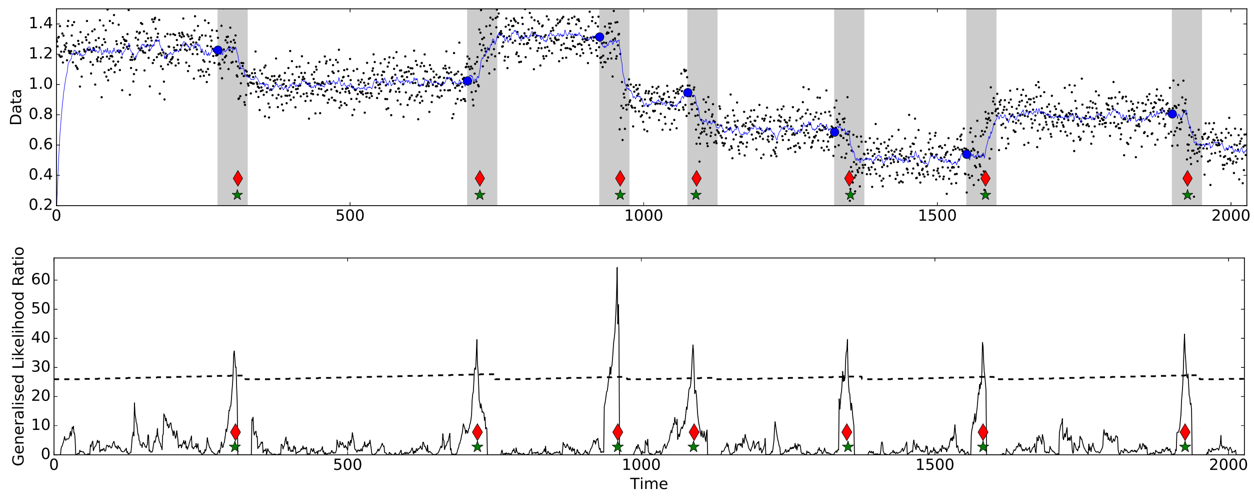


Figure 3: Changepoint detection in an 1-D time series. The moving average parameter θ is initialized at 0 and updated in each step with the gradient update $\theta \leftarrow \theta + 0.1(y_t - \theta)$, where $\rho = 0.1$ is the learning rate. (**Top**) Data (black dots), θ (blue line) and the detected changepoints, shown as green stars, while the red diamonds are the ground-truth values. For each detection all data used in the corresponding testing window are highlighted by the shaded areas. The blue dots on the left-borders of these areas are checkpoints. (**Bottom**) The GLR test values, $-2\log\Lambda_\tau$ (solid back line) and the detection threshold, i.e. $h(\delta_i) = \text{quantile}(1-\delta_i)$ (dotted line), where the latter increases with any new test and resets to its initial value after a detection.

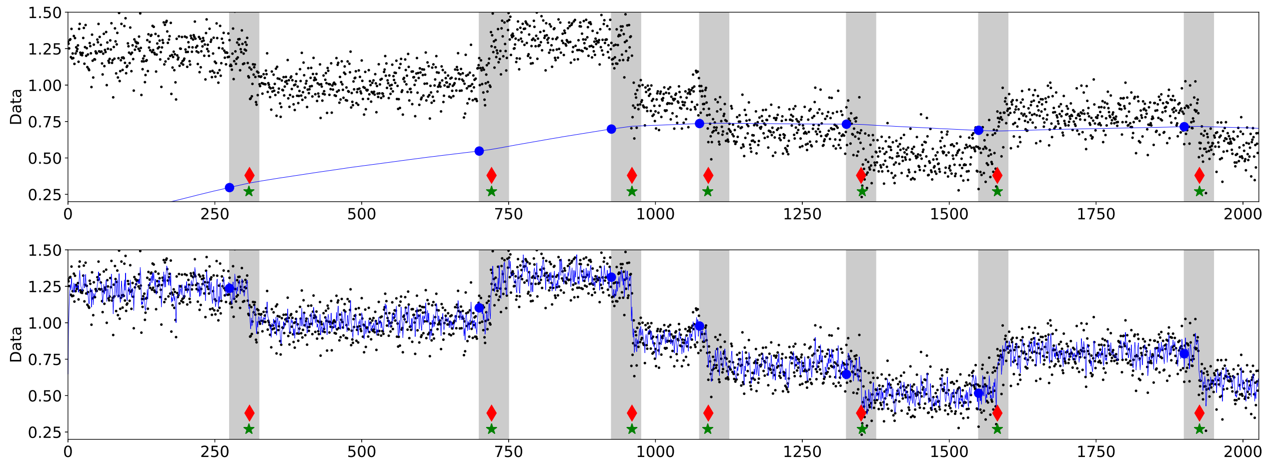
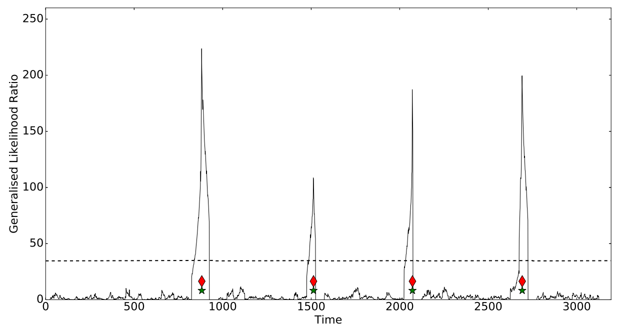


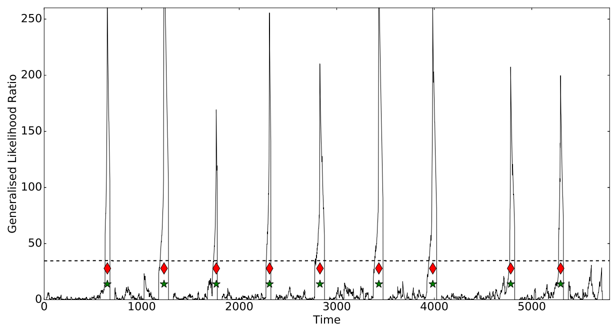
Figure 4: Effect of misspecified learning rate. (**Top**) Detection under a very small learning rate having value $\rho = 0.001$ that leads to under-fitting of the data. (**Bottom**) Detection under a very large learning with value $\rho = 0.5$ that leads to over-fitting. Despite that both rather bad choices of the learning rate result in poor fits to the data, changepoint detection remains accurate with no errors.

together with a prior distribution $p(\tau)$ over changepoints, defined through a Hazard function on the run length (Adams and MacKay, 2007), that models the prior probability of each time being a changepoint. Then Bayesian online learning requires marginalizing out all unknowns, i.e. (μ, σ^2) and the run length. Because of the conjugate and Markov structure of the model all computations are analytic and the marginal posterior probability of a changepoint, $p(t = \tau|\text{data})$, across time follows a simple and efficient recursion; see Adams and MacKay (2007) for full details. To apply the algorithm to CL we need to choose a cut-off threshold for $p(t = \tau|\text{data})$ that will allow to claim a changepoint. We consider a search over different cut-offs and we report the best-performing values in Table 1. As a changepoint prior we consider a constant hazard $H = 1/500$.

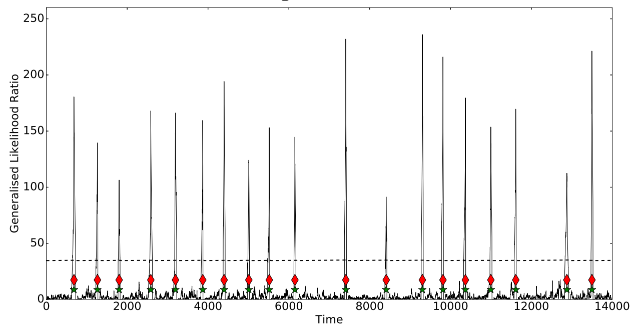
We also included in the comparison a simpler (SimpleCD) baseline based on purely online statistical testing scheme (without requiring storage of checkpoints)



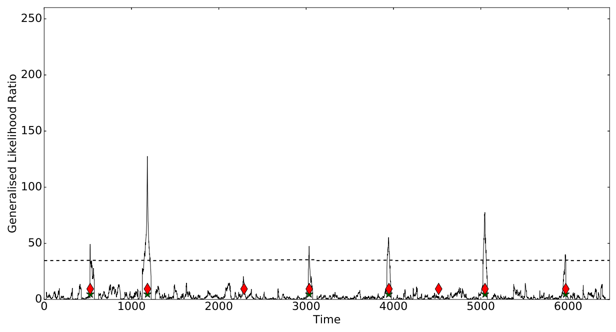
(a) Split-MNIST



(b) Permuted-MNIST



(c) CIFAR100



(d) Incr-Class-MNIST

Figure 5: GLR test values, $-2\log\Lambda_\tau$, for Split-MNIST, Permuted-MNIST, CIFAR100 and Incr-Class-MNIST. The mini-batch size for all cases was 10.

using the one-step ahead predictive score values $\mathbf{v}_t = \{v(y_t^i, \theta_{t-1})\}_{i=1}^b$, where here $\mathbf{v}_t$ is a vector of b values and b is the mini-batch size. Then a standard paired Welch's t test $t(\mathbf{v}_{t-1}, \mathbf{v}_t)$ can be used to detect a changepoint by using a cut-off critical value. In all experiments we considered a set of different critical values and we report the best-performing one in Table 1.

Furthermore, for both BayesCD and SimpleCD algorithms we added the constraint that after a detection the algorithm must wait $T = 100$ time steps to search for a new changepoint, i.e. the minimum distance between two consecutive detections was set to T. Without this constraint the behaviour of these algorithms can become very noisy resulting in many false positive detections around a previous detected changepoint. Note that this T minimum distance constraint is by definition satisfied by CheckpointCD, as shown in Algorithm 1, where T is the window size hyperparameter.

6.2.1 Datasets, CL tasks and results

We first applied the algorithms to three standard CL classification benchmarks: Split-MNIST, Permuted-MNIST and Split-CIFAR100. Split-MNIST, introduced by Zenke et al. (2017), assumes that five binary classification tasks are constructed from the original MNIST (LeCun and Cortes, 2010) hand-written digit classes and they are sequentially presented to the algorithm in the following order: 0/1, 2/3, 4/5, 6/7, 8/9. Each task is a binary classification problem so that any mini-batch $y_t = \{x_t^i, c_t^i\}_{i=1}^b$ is such that each $c_t^i \in \{0,1\}$, i.e. the task identity cannot be revealed by inspecting these binary labels. In Permuted-MNIST (Goodfellow et al., 2013; Kirkpatrick et al., 2017), each task is a variant of the initial 10-class MNIST classification problem where all input pixels have undergone a fixed (random) permutation. A sequence of 10 tasks is considered so that each task is a 10-class classification problem. For the Split-CIFAR100 we assume a sequence of 20 tasks of 5 classes each from the initial CIFAR100 dataset that contains images of 100 visual categories, indexed as $1, 2, \ldots, 100$. We follow Lopez-Paz et al. (2017), so that the first task contains the classes $(1,2,3,4,5)$ the second $(6,7,8,9,10)$ and etc.

For Split and Permuted-MNIST we consider a neural network with a shared representation $\phi(x;\theta^s)$ obtained by a fully connected multi-layer perceptron (MLP) network with two hidden layers of size 100 and rectified linear units (ReLU) activations. For the Split-CIFAR100 we used a much more complex residual network architecture (He et al., 2016) with 18 layers (ResNet-18), as used by Lopez-Paz et al. (2017). We created a simulated experiment where a true task changepoint can occur independently at each time step with some small probability with the additional constraint that we need to observe at least a minimum number of mini-batches from each true task before the next

Table 1: Average Jaccard index scores, with one standard deviations, and tolerance 5 on all CL changepoint detection tasks. The numbers inside brackets for the BayesCD method indicate different cut-offs in the changepoint posterior probability $p(t = \tau|\text{data})$.

Dataset	method	batch size=10	batch size=20	batch size=50	batch size=100
Split-MNIST	CheckpointCD	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$
	BayesCD(0.3)	0.38 ± 0.12	0.71 ± 0.12	0.93 ± 0.15	$\mathbf{1.00} \pm 0.00$
	BayesCD(0.4)	0.48 ± 0.16	0.89 ± 0.14	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$
	BayesCD(0.5)	0.58 ± 0.16	0.91 ± 0.11	$\mathbf{1.00} \pm 0.00$	0.97 ± 0.07
	BayesCD(0.6)	0.71 ± 0.18	0.93 ± 0.11	0.97 ± 0.07	0.95 ± 0.10
	SimpleCD	0.35 ± 0.21	0.82 ± 0.13	0.98 ± 0.06	0.92 ± 0.13
Permuted-MNIST	CheckpointCD	$\mathbf{0.77} \pm 0.13$	$\mathbf{1.00} \pm 0.00$	0.98 ± 0.06	$\mathbf{1.00} \pm 0.00$
	BayesCD(0.3)	0.42 ± 0.09	0.97 ± 0.05	0.99 ± 0.03	$\mathbf{1.00} \pm 0.00$
	BayesCD(0.4)	0.44 ± 0.09	0.93 ± 0.12	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$
	BayesCD(0.5)	0.57 ± 0.11	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$
	BayesCD(0.6)	0.66 ± 0.13	0.99 ± 0.03	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$
	SimpleCD	0.30 ± 0.08	0.92 ± 0.06	0.96 ± 0.07	0.99 ± 0.03
Split-CIFAR100	CheckpointCD	$\mathbf{0.98} \pm 0.04$	$\mathbf{1.00} \pm 0.00$	0.99 ± 0.03	$\mathbf{1.00} \pm 0.00$
	BayesCD(0.3)	0.96 ± 0.04	0.95 ± 0.07	0.98 ± 0.02	$\mathbf{1.00} \pm 0.00$
	BayesCD(0.4)	0.93 ± 0.04	0.99 ± 0.02	0.99 ± 0.02	$\mathbf{1.00} \pm 0.00$
	BayesCD(0.5)	0.93 ± 0.05	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$
	BayesCD(0.6)	0.85 ± 0.08	0.99 ± 0.02	$\mathbf{1.00} \pm 0.00$	$\mathbf{1.00} \pm 0.00$
	SimpleCD	0.14 ± 0.08	0.76 ± 0.06	0.99 ± 0.02	0.95 ± 0.04
Incr-Class-MNIST	CheckpointCD	$\mathbf{0.72} \pm 0.14$	$\mathbf{0.96} \pm 0.06$	$\mathbf{0.96} \pm 0.09$	$\mathbf{0.98} \pm 0.07$
	BayesCD(0.3)	0.41 ± 0.09	0.64 ± 0.15	0.85 ± 0.10	0.81 ± 0.07
	BayesCD(0.4)	0.41 ± 0.12	0.75 ± 0.11	0.82 ± 0.10	0.85 ± 0.09
	BayesCD(0.5)	0.45 ± 0.13	0.68 ± 0.12	0.74 ± 0.08	0.86 ± 0.07
	BayesCD(0.6)	0.43 ± 0.11	0.60 ± 0.17	0.81 ± 0.12	0.81 ± 0.06
	SimpleCD	0.04 ± 0.05	0.09 ± 0.08	0.58 ± 0.19	0.89 ± 0.09
Incr-Class-CIFAR44	CheckpointCD	$\mathbf{0.24} \pm 0.05$	$\mathbf{0.80} \pm 0.06$	$\mathbf{0.98} \pm 0.03$	$\mathbf{0.98} \pm 0.05$
	BayesCD(0.3)	0.07 ± 0.04	0.43 ± 0.04	0.81 ± 0.05	0.96 ± 0.03
	BayesCD(0.4)	0.04 ± 0.02	0.36 ± 0.09	0.76 ± 0.04	0.92 ± 0.03
	BayesCD(0.5)	0.02 ± 0.02	0.26 ± 0.06	0.71 ± 0.06	0.88 ± 0.04
	BayesCD(0.6)	0.02 ± 0.02	0.20 ± 0.05	0.64 ± 0.05	0.86 ± 0.05
	SimpleCD	0.00 ± 0.01	0.01 ± 0.01	0.12 ± 0.04	0.74 ± 0.05

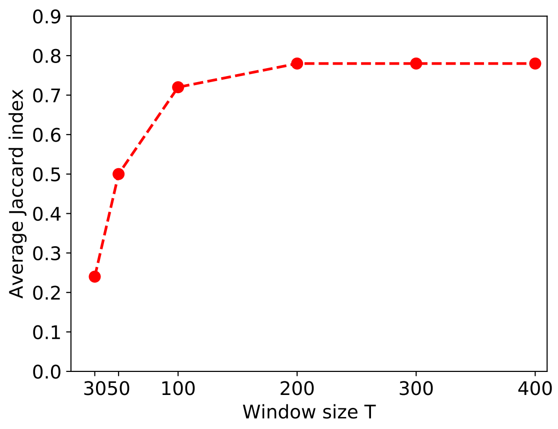


Figure 6: Jaccard index averaged over 100 repeats in Incr-Class-MNIST for varying window sizes T and fixed mini-batch size 10.

changepoint.[3] We compare the proposed CheckpointCD method with BayesCD and SimpleCD under four different values of the mini-batch size b: 10, 20, 50, 100. Training of each CL model was based on Algorithm 3, modified accordingly for BayesCD and SimpleCD so that to apply their respective changepoint detection subroutine instead of Algorithm 1 used by CheckpointCD. The learning rate sequence ρ_t in stochastic gradient optimization of the objective in Eq. (8) was set in a dimension-wise manner by using the Adam optimizer (Kingma and Ba, 2014) which the standard approach for training neural networks; see Appendix for further details.

To measure performance we used the intersection over union score (also called Jaccard index) defined as the number of correctly detected changepoints divided by the union of the detected and the true changepoints

$$\text{Jaccard index} = \frac{|True \cap Detected|}{|True \cup Detected|} \in [0,1],$$

where the larger the score the better. Note that the Jaccard index is the hardest among other related scores such as recall, precision and F1, which are softer/upper bounds (i.e. closer to 1) than Jaccard index. For completeness full tables with precision and recall score values are given in the Appendix. When computing Jaccard index we also allow some tolerance when declaring that a pair (τ^{tr}, τ^{det}) of a true and detected locations correspond to the same true changepoint τ^{tr}. A tolerance equal to 5 time steps distance was used which means that only when $|\tau^{tr} - \tau^{det}| \leq 5$ the detected τ^{det} is considered correct.

Furthermore, in order to create harder changepoint detection problems, we consider two class-incremental variants of MNIST and and CIFAR so that each task differs from the previous task by changing only a single class and without affecting the labeling of the remaining classes. This creates the Incr-Class-MNIST with 9 tasks: $0/1, 2/1, 2/3, \ldots, 8/7, 8/9$. To speed up the experiment in CIFAR we consider only the first 44 classes and create a very challenging changepoint detection problem, Incr-Class-CIFAR44, with 40 tasks: (1,2,3,4,5), (6,2,3,4,5), (6,7,3,4,5) and etc.

Table 1 reports all results obtained by 10 random repetitions of the experiments. The table shows that the proposed algorithm is consistently better than the other methods and it provides accurate changepoint detection even with mini-batch size as small as 20. Notice also that, as expected, all methods improve as the mini-batch size increases.

Fig. 5 visualizes the GLR values, $-2\log\Lambda_\tau$, in some of the runs with Split-MNIST, Permuted-MNIST, CIFAR100 and Incr-Class-MNIST. Similarly, to Fig. 3 most changepoints are detected by our algorithm and every changepoint corresponds to a clear spike in the GLR statistics. Note that the plots in Fig. 5 are obtained for the most difficult case where the data mini-batch size when fitting the CL model is 10, while for larger mini-batch sizes the detection is more robust and the spikes of the GLR statitics become sharper.

Finally, Fig. 6 studies the effect of the window size T in changepoint detection performance, which shows that too small value of T could decrease the performance presumably due to very small sample size when performing each test. This corroborate our discussion in Section 3.3 that a large value of T increases the power of hypothesis testing, although it should not be larger than the minimum length of a task from our prior knowledge to avoid including multiple changepoints in the same testing window.

7 Discussion

We have introduced an algorithm for online changepoint detection that can be easily combined with online learning of complex non-linear models such as neural networks. We have demonstrated the effectiveness of our method in challenging continual learning tasks for automatically detecting the task changepoints. The use of checkpoints allowed us to define a sequential hypothesis testing procedure to control a predetermined Type I error upper-bound, and evaluate empirically the overall performance of both Type I and II error using Jaccard index and or other metrics.

The simplicity of checkpoints means that practitioners can use them for changepoint detection without having to modify their preferred way of estimating or fitting models to data. For instance, in deep learning (LeCun et al., 2015) the dominant approach to model fitting is based on point parameter estimation with stochastic gradient descent (SGD), where the model is typically a neural network. As seen in this paper this can be easily combined with checkpoints to detect changepoints, without having to modify this stan-

[3] In all experiments this number was 500 and the probability of having a changepoint at each step (after the 500 steps) was 0.005.

dard SGD model fitting procedure. Similarly, checkpoints could be also combined with other ways of fitting models to data, e.g. Bayesian approaches, since the essence of the algorithm is a *cached model representation* (not necessarily a point parametric estimate) that together with a prediction score can detect changes. For instance, if we follow a Bayesian model estimation approach, online learning will require updating a posterior probability distribution $p_t(\theta)$ through time. Then, a checkpoint becomes an early version of this posterior distribution, i.e. $p_{t-T}(\theta)$, while the predictive score will be obtained by averaging some function under this checkpoint posterior. In this setting, the use of the algorithm remains the same and the only thing we need to modify, to accommodate this Bayesian way of model fitting, is to change the online model update rule (i.e. the line $\theta_t = \text{update\_step}(\theta_{t-1}, y_t)$ in Algorithm 1) together with the definition of the score function $v(\cdot)$, where the latter should correspond now to a Bayesian predictive score. While Bayesian model fitting is very difficult for complex models, such as neural networks, it is certainly feasible for simple conjugate Bayesian models where we could apply the checkpoint method as outlined above. We leave the experimentation with this more Bayesian way of using checkpoints as a future work.

Finally, another topic for future research is to consider checkpoints to detect changes at different time scales, such as long-term and short-term changes.

References

Adams RP, MacKay D (2007) Bayesian online changepoint detection. Tech. rep.

Aljundi R, Kelchtermans K, Tuytelaars T (2018) Task-free continual learning. CoRR abs/1812.03596

Aljundi R, Lin M, Goujaud B, Bengio Y (2019) Online continual learning with no task boundaries. CoRR abs/1903.08671

Aminikhanghahi S, Cook DJ (2017) A survey of methods for time series change point detection. Knowl Inf Syst 51(2):339–367

Bansal R, Zhou H (2002) Term structure of interest rates with regime shifts. The Journal of Finance 57(5):1997–2043

Barry D, Hartigan JA (1992) Product partition models for change point problems. Annals of Statistics 20(1):260–279

Bishop CM (2006) Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., Secaucus, NJ, USA

Caron F, Doucet A, Gottardo R (2012) On-line changepoint detection and parameter estimation with application to genomic data. Statistics and Computing 22(2):579–595

Csorgo M, Horváth L (1997) Limit theorems in change-point analysis. John Wiley & Sons Chichester

D'Agostino R, Pearson E (1973) Tests for departure from normality. empirical results for the distributions of b2 and b1. Biometrika pp 613–622

Farquhar S, Gal Y (2018) Towards Robust Evaluations of Continual Learning. arXiv preprint arXiv:180509733

Fearnhead P (2006) Exact and efficient Bayesian inference for multiple changepoint problems. Statistics and Computing 16(2):203–213

Fearnhead P, Liu Z (2007) Online inference for multiple changepoint problems. Journal of the Royal Statistical Society, Series B 69:589–605

Goodfellow IJ, Mirza M, Xiao D, Courville A, Bengio Y (2013) An empirical investigation of catastrophic forgetting in gradient-based neural networks. arXiv preprint arXiv:13126211

Hawkins DM, Qiu P, Kang CW (2003) The changepoint model for statistical process control. Journal of Quality Technology 35(4):355–366

He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, IEEE Computer Society, pp 770–778

Jandhyala VK, Fotopoulos SB, Hawkins DM (2002) Detection and estimation of abrupt changes in the variability of a process. Computational statistics & data analysis 40(1):1–19

Kaplanis C, Shanahan M, Clopath C (2018) Continual reinforcement learning with complex synapses. arXiv preprint arXiv:180207239

Killick R, Fearnhead P, Eckley I (2012) Optimal detection of changepoints with a linear computational cost. Journal of the American Statistical Association 107(500):1590–1598

Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. Cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015

Kirkpatrick J, Pascanu R, Rabinowitz N, Veness J, Desjardins G, Rusu AA, Milan K, Quan J, Ramalho T, Grabska-Barwinska A, et al. (2017) Overcoming catastrophic forgetting in neural networks. Proceedings of the National Academy of Sciences p 201611835

LeCun Y, Cortes C (2010) MNIST handwritten digit database

LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444, DOI 10.1038/nature14539

Li Z, Hoiem D (2017) Learning without forgetting. IEEE Transactions on Pattern Analysis and Machine Intelligence

Lopez-Paz D, et al. (2017) Gradient episodic memory for continual learning. In: Advances in Neural Information Processing Systems, pp 6470–6479

Maidstone R, Hocking T, Rigaill G, Fearnhead P (2017) On optimal multiple changepoint algorithms for large data. Statistics and Computing 27(2):519–533

Nguyen CV, Li Y, Bui TD, Turner RE (2017) Variational continual learning. arXiv preprint arXiv:171010628

Page ES (1957) On problems in which a change in a parameter occurs at an unknown point. Biometrika 44(1-2):248–252

d'Agostino Ralph B (1971) An omnibus test of normality for moderate and large size samples. Biometrika 58(2):341–348

Rao D, Visin F, Rusu A, Pascanu R, Teh YW, Hadsell R (2019) Continual unsupervised representation learning. In: Neural Information Processing Systems

Rebuffi SA, Kolesnikov A, Sperl G, Lampert CH (2017) icarl: Incremental classifier and representation learning. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, pp 5533–5542

Ring MB (1994) Continual learning in reinforcement environments. PhD thesis, University of Texas at Austin Austin, Texas 78712

Robbins H, Monro S (1951) A stochastic approximation method. Ann Math Statist 22(3):400–407, DOI 10.1214/aoms/1177729586

Robins A (1995) Catastrophic forgetting, rehearsal and pseudorehearsal. Connection Science 7(2):123–146

Robins A, McCallum S (1998) Catastrophic forgetting and the pseudorehearsal solution in hopfield-type networks. Connection Science 10(2):121–135

Rusu AA, Rabinowitz NC, Desjardins G, Soyer H, Kirkpatrick J, Kavukcuoglu K, Pascanu R, Hadsell R (2016) Progressive neural networks. arXiv preprint arXiv:160604671

Schmidhuber J (2013) Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. Frontiers in psychology 4:313

Truong C, Oudre L, Vayatis N (2018) Selective review of offline change point detection methods. 1801.00718

Yildirim S, Singh SS, Doucet A (2013) An online expectation–maximization algorithm for changepoint models. Journal of Computational and Graphical Statistics 22(4):906–926

Zenke F, Poole B, Ganguli S (2017) Continual learning through synaptic intelligence. arXiv preprint arXiv:170304200

Zeno C, Golan I, Hoffer E, Soudry D (2018) Task agnostic continual learning using online variational bayes

A Quantile of Z statistics in Algorithm 2

For every window size T, we compute the quantile of the Z statistics (threshold $h(\delta) = \text{quantile}(1-\delta)$ in Algorithm 2) numerically with 10^8 simulations, and fit a linear function for $h(\delta)$. Table **??** show the computed threshold values as a function of T, border size α, and eror δ. We also show the fitted line of $h_T(\delta)$ when $\alpha = \lfloor T/4 \rfloor$ as used in the experiments in Figure 7. We observe that the threshold is close to convergence when $T \geq 100$.

Table 2: Z-statistics Quantile for T=30.

Error δ	$\alpha = 5$	$\alpha = 7$
0.1	10.024	9.245
0.05	11.909	11.084
0.01	16.089	15.173
0.001	21.837	20.786
0.0001	27.464	26.273
1e-05	33.124	31.706
1e-06	38.882	37.244

Table 3: Z-statistics Quantile for T=50.

Error δ	$\alpha = 5$	$\alpha = 10$	$\alpha = 12$
0.1	10.661	9.318	8.948
0.05	12.518	11.095	10.711
0.01	16.633	15.043	14.635
0.001	22.283	20.454	20.018
0.0001	27.824	25.721	25.253
1e-05	33.200	30.771	30.429
1e-06	38.841	35.969	35.716

Table 4: Z-statistics Quantile for T=100.

Error δ	$\alpha = 5$	$\alpha = 10$	$\alpha = 15$	$\alpha = 20$	$\alpha = 25$
0.1	11.270	10.303	9.726	9.244	8.789
0.05	13.099	12.069	11.471	10.976	10.507
0.01	17.143	15.971	15.334	14.818	14.332
0.001	22.705	21.315	20.628	20.088	19.578
0.0001	28.152	26.500	25.769	25.180	24.676
1e-05	33.576	31.591	30.852	30.249	29.740
1e-06	38.957	36.549	35.712	35.165	34.720

Table 5: Z-statistics Quantile for T=200.

Error δ	$\alpha = 5$	$\alpha = 20$	$\alpha = 35$	$\alpha = 50$
0.1	11.780	10.257	9.505	8.833
0.05	13.587	11.989	11.223	10.537
0.01	17.588	15.817	15.028	14.323
0.001	23.080	21.057	20.248	19.516
0.0001	28.482	26.173	25.320	24.611
1e-05	33.874	31.153	30.254	29.514
1e-06	38.938	35.961	35.187	34.499

Table 6: Z-statistics Quantile for T=300.

Error δ	$\alpha = 5$	$\alpha = 25$	$\alpha = 45$	$\alpha = 65$	$\alpha = 75$
0.1	12.023	10.466	9.769	9.171	8.877
0.05	13.823	12.194	11.486	10.879	10.578
0.01	17.799	16.007	15.285	14.666	14.357
0.001	23.271	21.214	20.471	19.848	19.534
0.0001	28.667	26.270	25.498	24.868	24.546
1e-05	34.061	31.411	30.528	29.960	29.650
1e-06	39.142	36.424	35.684	35.072	34.756

Table 7: Z-statistics Quantile for T=400.

Error δ	$\alpha = 5$	$\alpha = 35$	$\alpha = 65$	$\alpha = 95$	$\alpha = 100$
0.1	12.182	10.431	9.682	9.017	8.907
0.05	13.974	12.154	11.396	10.721	10.609
0.01	17.935	15.957	15.187	14.496	14.382
0.001	23.389	21.160	20.378	19.681	19.567
0.0001	28.729	26.232	25.423	24.730	24.621
1e-05	34.091	31.130	30.309	29.661	29.568
1e-06	39.388	35.642	35.106	34.543	34.319

B Further details and results

For all CL experiments in Section 6 we used the Adam optimizer (Kingma and Ba, 2014) with its default parameter settings and with base learning rate value $\alpha = 0.1/b$ where b is the mini-batch size in each experiment. The hyperparameter λ in the loss function in Eq. (8) was set to $\lambda = 1$ and the size of the replay buffer of each previous task was set to 100, i.e. $|\mathscr{R}_i| = 100$.

Table 8 provides the precision scores and Table 9 the recalls for all algorithms applied to the CL benchmarks.

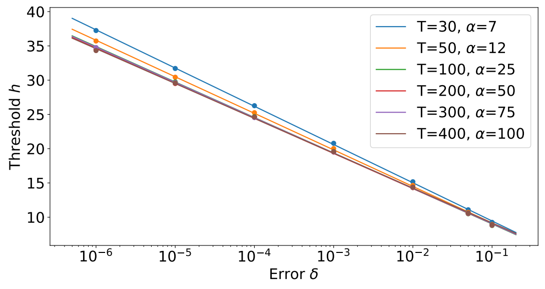


Figure 7: Threshold as a function of error $h(\delta)$ for different window size T. The border size α is set as $\lfloor T/4 \rfloor$. The circles are computed numerically, and the straight lines are linearly fitted functions.

Table 8: Average precision, with one-standard deviations, and tolerance 5 on all CL changepoint detection tasks. The numbers inside brackets for the BayesCD method indicate different cut-offs in the changepoint posterior probability $p(t = \tau|\text{data})$.

Dataset	method	batch size=10	batch size=20	batch size=50	batch size=100
Split-MNIST	CheckpointCD	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	BayesCD(0.3)	0.41 ± 0.12	0.72 ± 0.11	0.94 ± 0.12	1.00 ± 0.00
	BayesCD(0.4)	0.51 ± 0.15	0.89 ± 0.14	1.00 ± 0.00	1.00 ± 0.00
	BayesCD(0.5)	0.63 ± 0.14	0.96 ± 0.08	1.00 ± 0.00	1.00 ± 0.00
	BayesCD(0.6)	0.88 ± 0.15	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	SimpleCD	0.62 ± 0.31	0.96 ± 0.09	0.98 ± 0.06	0.94 ± 0.10
Permuted-MNIST	CheckpointCD	0.79 ± 0.12	1.00 ± 0.00	0.99 ± 0.03	1.00 ± 0.00
	BayesCD(0.3)	0.44 ± 0.09	0.97 ± 0.05	0.99 ± 0.03	1.00 ± 0.00
	BayesCD(0.4)	0.45 ± 0.09	0.94 ± 0.10	1.00 ± 0.00	1.00 ± 0.00
	BayesCD(0.5)	0.59 ± 0.12	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	BayesCD(0.6)	0.66 ± 0.13	0.99 ± 0.03	1.00 ± 0.00	1.00 ± 0.00
	SimpleCD	0.77 ± 0.18	0.94 ± 0.05	0.97 ± 0.05	0.99 ± 0.03
Split-CIFAR100	CheckpointCD	0.99 ± 0.02	1.00 ± 0.00	0.99 ± 0.02	1.00 ± 0.00
	BayesCD(0.3)	0.98 ± 0.03	0.95 ± 0.07	0.98 ± 0.02	1.00 ± 0.00
	BayesCD(0.4)	0.98 ± 0.03	0.99 ± 0.02	0.99 ± 0.02	1.00 ± 0.00
	BayesCD(0.5)	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	BayesCD(0.6)	0.99 ± 0.03	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	SimpleCD	0.54 ± 0.27	0.94 ± 0.07	0.99 ± 0.02	0.96 ± 0.04
Incr-Class-MNIST	CheckpointCD	0.95 ± 0.08	1.00 ± 0.00	0.97 ± 0.05	0.99 ± 0.04
	BayesCD(0.3)	0.47 ± 0.08	0.76 ± 0.13	0.97 ± 0.07	0.97 ± 0.05
	BayesCD(0.4)	0.52 ± 0.12	0.89 ± 0.10	1.00 ± 0.00	1.00 ± 0.00
	BayesCD(0.5)	0.62 ± 0.15	0.89 ± 0.11	0.99 ± 0.04	1.00 ± 0.00
	BayesCD(0.6)	0.73 ± 0.16	0.94 ± 0.13	1.00 ± 0.00	1.00 ± 0.00
	SimpleCD	0.11 ± 0.14	0.37 ± 0.37	0.92 ± 0.11	0.99 ± 0.03
Incr-Class-CIFAR44	CheckpointCD	0.96 ± 0.05	0.96 ± 0.02	0.99 ± 0.02	0.99 ± 0.02
	BayesCD(0.3)	0.71 ± 0.27	0.89 ± 0.06	0.97 ± 0.02	1.00 ± 0.01
	BayesCD(0.4)	0.75 ± 0.25	0.94 ± 0.05	0.98 ± 0.04	0.99 ± 0.01
	BayesCD(0.5)	0.58 ± 0.44	0.95 ± 0.10	1.00 ± 0.01	1.00 ± 0.00
	BayesCD(0.6)	0.60 ± 0.49	0.98 ± 0.05	1.00 ± 0.00	1.00 ± 0.00
	SimpleCD	0.02 ± 0.05	0.11 ± 0.17	0.86 ± 0.18	0.96 ± 0.03

Table 9: Average recall, with one-standard deviations, and tolerance 5 on all CL changepoint detection tasks. The numbers inside brackets for the BayesCD method indicate different cut-offs in the changepoint posterior probability $p(t = \tau|\text{data})$.

Dataset	method	batch size=10	batch size=20	batch size=50	batch size=100
Split-MNIST	CheckpointCD	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	BayesCD(0.3)	0.85 ± 0.17	0.97 ± 0.07	0.97 ± 0.07	1.00 ± 0.00
	BayesCD(0.4)	0.85 ± 0.17	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	BayesCD(0.5)	0.85 ± 0.17	0.95 ± 0.10	1.00 ± 0.00	0.97 ± 0.07
	BayesCD(0.6)	0.80 ± 0.19	0.93 ± 0.11	0.97 ± 0.07	0.95 ± 0.10
	SimpleCD	0.42 ± 0.23	0.85 ± 0.12	1.00 ± 0.00	0.97 ± 0.07
Permuted-MNIST	CheckpointCD	0.97 ± 0.05	1.00 ± 0.00	0.99 ± 0.03	1.00 ± 0.00
	BayesCD(0.3)	0.91 ± 0.07	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	BayesCD(0.4)	0.91 ± 0.10	0.98 ± 0.04	1.00 ± 0.00	1.00 ± 0.00
	BayesCD(0.5)	0.94 ± 0.06	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	BayesCD(0.6)	0.99 ± 0.03	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	SimpleCD	0.34 ± 0.10	0.98 ± 0.04	0.99 ± 0.03	1.00 ± 0.00
Split-CIFAR100	CheckpointCD	0.99 ± 0.02	1.00 ± 0.00	0.99 ± 0.02	1.00 ± 0.00
	BayesCD(0.3)	0.98 ± 0.03	0.99 ± 0.02	1.00 ± 0.00	1.00 ± 0.00
	BayesCD(0.4)	0.94 ± 0.03	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	BayesCD(0.5)	0.93 ± 0.05	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
	BayesCD(0.6)	0.86 ± 0.08	0.99 ± 0.02	1.00 ± 0.00	1.00 ± 0.00
	SimpleCD4.0	0.15 ± 0.09	0.81 ± 0.05	1.00 ± 0.00	0.99 ± 0.02
Incr-Class-MNIST	CheckpointCD	0.74 ± 0.12	0.96 ± 0.06	0.97 ± 0.05	0.99 ± 0.04
	BayesCD(0.3)	0.74 ± 0.16	0.78 ± 0.12	0.88 ± 0.08	0.82 ± 0.06
	BayesCD(0.4)	0.64 ± 0.13	0.82 ± 0.10	0.82 ± 0.10	0.85 ± 0.09
	BayesCD(0.5)	0.61 ± 0.13	0.74 ± 0.12	0.75 ± 0.08	0.86 ± 0.07
	BayesCD(0.6)	0.51 ± 0.13	0.61 ± 0.15	0.81 ± 0.12	0.81 ± 0.06
	SimpleCD	0.05 ± 0.06	0.10 ± 0.09	0.60 ± 0.18	0.90 ± 0.09
Incr-Class-CIFAR44	CheckpointCD	0.24 ± 0.05	0.82 ± 0.05	0.99 ± 0.02	0.99 ± 0.02
	BayesCD(0.3)	0.07 ± 0.04	0.45 ± 0.04	0.83 ± 0.04	0.96 ± 0.02
	BayesCD(0.4)	0.04 ± 0.02	0.37 ± 0.09	0.77 ± 0.03	0.92 ± 0.03
	BayesCD(0.5)	0.02 ± 0.02	0.27 ± 0.06	0.72 ± 0.06	0.88 ± 0.04
	BayesCD(0.6)	0.02 ± 0.02	0.20 ± 0.05	0.63 ± 0.05	0.86 ± 0.05
	SimpleCD	0.00 ± 0.01	0.01 ± 0.01	0.13 ± 0.05	0.76 ± 0.04