Contents lists available at ScienceDirect

MethodsX

journal homepage: www.elsevier.com/locate/mex

Method Article

# A divided and prioritized experience replay approach for streaming regression[☆,☆☆]

Mikkel Leite Arnø[a,*], John-Morten Godhavn[b], Ole Morten Aamo[a]

[a] *Department of Engineering Cybernetics, Norwegian University of Science and Technology, Trondheim 7491, Norway*
[b] *Equinor Research Center, Ranheim 7053, Norway*

A B S T R A C T

In the streaming learning setting, an agent is presented with a data stream on which to learn from in an online fashion. A common problem is catastrophic forgetting of old knowledge due to updates to the model. Mitigating catastrophic forgetting has received a lot of attention, and a variety of methods exist to solve this problem. In this paper, we present a divided and prioritized experience replay approach for streaming regression, in which relevant observations are retained in the replay, and extra focus is added to poorly estimated observations through prioritization. Using a real-world dataset, the method is compared to the standard sliding window approach. A statistical power analysis is performed, showing how our approach improves performance on rare, important events at a trade-off in performance for more common observations. Close inspections of the dataset are provided, with emphasis on areas where the standard approach fails. A rephrasing of the problem to a binary classification problem is performed to separate common and rare, important events. These results provide an added perspective regarding the improvement made on rare events.

- *We divide the prediction space in a streaming regression setting*
- *Observations in the experience replay are prioritized for further training by the model's current error*

© 2021 The Author(s). Published by Elsevier B.V.
This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/)

A R T I C L E   I N F O
*Method name:* Prioritized n-bin experience replay
*Keywords:* Streaming regression, Catastrophic forgetting, Nonstationarity
*Article history:* Received 2 June 2021; Accepted 31 October 2021; Available online 12 November 2021

[☆] This document is the result of research funded by Equinor and the Norwegian University of Science and Technology.
[☆☆] Co-submission: https://doi.org/10.1016/j.petrol.2021.109006
DOI of original article: 10.1016/j.petrol.2021.109006
[*] Corresponding author.
*E-mail address:* mikkel.l.arno@ntnu.no (M. Leite Arnø).

https://doi.org/10.1016/j.mex.2021.101571
2215-0161/© 2021 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/)

Specification table

| | |
|---|---|
| Subject area | *Computer science* |
| More specific subject area | *Streaming learning* |
| Method name | *Prioritized n-bin experience replay* |
| Name and reference of original method | *N/A* |
| Resource availability | *N/A* |

## Introduction

In supervised learning, generalizability is a primary focus during the development of a model. During training, the labels are available, and are used to supervise the learning of a function that maps from inputs to an output. The goal is for the model to learn a function that gives accurate predictions for unseen observations, for which labels are not available. This is the essence of generalization, and relies on the assumption that the unseen observations come from the same distribution as the training observations. However, this assumption fails for many real-word problems. Shifts in independent or dependent variables, an evolving underlying process, and dependence on variables not included in the model are all examples of nonstationarity, which is harmful to the predictive power of such models [1,2]. Nonstationarity occurs to some extent in most real world data sets, and has been a motivating factor for the paradigm of streaming learning, where a model is presented with a stream of data on which to continuously learn from in an online fashion. This allows adaptation to changing data distribution, although the approach is prone to catastrophic forgetting [3]. In the setting of streaming learning, the goal is to leverage newly available data to adapt to changing environments while still performing well on previous observations [4]. These two objectives might be conflicting, giving rise to the *stability-plasticity* dilemma [5], asking how one can stay stable to irrelevant events, while plastic to new information. This problem has been addressed in several ways, perhaps most commonly by use of experience replay, where new observations from the data stream are mixed with older observations as they become available [6].

Streaming data can occur in many situations, and several problems can be solved by learning from these streams. Classification with dynamic selection of appropriate window [7], and classification using a streaming random forest [8] have been investigated. Clustering of data streams [9], multi-task learning with a global loss function [10], and multiple output linear regression [11] are other examples. The challenges of streaming learning have been discussed in many earlier publications, with emphasis on managing catastrophic forgetting. Reducing the risk of catastrophic forgetting has been approached in several ways, among them regularization, Kirkpatrick et al. [12], Li and Hoiem [13], where weight updates are constrained so previously learned relationships are not erased, and ensembling methods, where multiple models are trained, and their outputs are combined with some form of majority voting [2,14,15]. Various experience replay configurations have also been developed for this purpose, among them stream clustering methods to retain valuable information [16], and prioritization of samples to train on [17], where the former has with benefit been applied to streaming classification, and the latter to reinforcement learning. These differ from the aforementioned methods, as they focus on which observations to retain and train on rather than on the model itself.

In the current work, a divided and prioritized experience replay approach for streaming regression is presented to mitigate the effects of catastrophic forgetting while allowing adaptation to nonstationarity. We adopt both the philosophy of retaining relevant knowledge in the replay, along with that of prioritization. A deep neural network (DNN) is trained and validated on historical data. This model serves as a baseline model which is deployed for streaming learning during operation, using this approach. To demonstrate its effect, the method is applied to a real-world dataset, and benchmarked against a standard sliding window approach. The method was first presented in Arnøet al. [18] as a case study. In the current work, however, a thorough comparison to the standard sliding window approach for streaming regression is given, with focus on rare, important events, and discussions of areas where the standard sliding window fails.

The paper is structured as follows: Section 2 presents the methods used in this work, and Section 3 presents our results in a case study, comparing our method with a standard sliding window. Lastly, Section 4 offers conclusions.

## Method

The selected model architecture for this work was a DNN. DNNs consist of multiple layers of interconnected neurons with nonlinear activation functions. This allows extraction of latent, possibly nonlinear features within the data. First, we provide notation for the DNN:

$$\begin{aligned}
L &: \text{number of layers in the DNN}\\
m &: \text{number of observations}\\
f &: \text{number of features}\\
x &: \text{features}\\
y &: \text{target variable}\\
s_l &: \text{number of neurons in layer } l \in 1, \dots, L\\
(x_i, y_i) &: i\text{th observation, } i \in 1, \dots, m\\
\mathbf{w}^{[l]} &: \text{trainable weight matrix for layer } l\\
\mathbf{b}^{[l]} &: \text{trainable bias vector for layer } l
\end{aligned}$$

and

$$\mathbf{x} = \begin{bmatrix} \vdots & \vdots & & \vdots \\ x_1 & x_2 & \dots & x_m \\ \vdots & \vdots & & \vdots \end{bmatrix} \in \mathbb{R}^{f \text{ x } m}, \tag{1}$$

$$\mathbf{y} = \begin{bmatrix} y_1 & y_2 & \dots & y_m \end{bmatrix} \in \mathbb{R}^{1 \text{ x } m}, \tag{2}$$

$$\mathbf{w}^{[l]} = \begin{bmatrix} \dots & w_1^{[l]\top} & \dots \\ \dots & w_2^{[l]\top} & \dots \\ & \vdots & \\ \dots & w_{s_{l-1}}^{[l]\top} & \dots \end{bmatrix} \in \mathbb{R}^{s_{l-1} \text{ x } s_l}, \tag{3}$$

$$\mathbf{b}^{[l]} = \begin{bmatrix} b_1^{[l]} \\ b_2^{[l]} \\ \vdots \\ b_{s_l}^{[l]} \end{bmatrix} \in \mathbb{R}^{s_l \text{ x } 1}. \tag{4}$$

To make a prediction on a single observation of inputs $x_i$, the input is mapped to $\hat{y}_i$ by forward propagation through the DNN:

$$z_i^{[l]} = \mathbf{w}^{[l]\top} a_i^{[l-1]} + \mathbf{b}^{[l]} \quad l = 1, \dots, L, \tag{5}$$

$$a_i^{[l]} = g^{[l]}(z_i^{[l]}) \quad l = 1, \dots, L-1, \tag{6}$$

$$a_i^{[l]} = \max\{\gamma z_i^{[l]}, z_i^{[l]}\} \quad l = 1, \dots, L-1, \tag{7}$$

$$\hat{y}_i = a_i^{[L]} = z_i^{[L]}. \tag{8}$$

Eq. (5) describes the linear part of the forward propagation, mapping from one layer to the next throughout the DNN. This starts at $a_i^{[0]} = x_i$. At each hidden layer, these linear combinations are passed through nonlinear activation functions $g^{[l]}$, as described in Eq. (6). In this work, the leaky ReLU activation functions given in Eq. (7) are used, where $\gamma$ is the slope in the left half plane. Lastly, the output layer outputs the prediction, which for this work is a real number, resulting in a regression layer. This is described in Eq. (8).

For learning on a mini-batch of size $m_b$, inputs $\mathbf{x}_b$ and outputs $\mathbf{y}_b$ are used. The forward propagation for the mini-batch is given by:

$$\mathbf{z}^{[l]} = \mathbf{w}^{[l]\top}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}\mathbf{1} \quad l = 1, \ldots, L, \tag{9}$$

$$\mathbf{a}^{[l]} = g^{[l]}(\mathbf{z}^{[l]}) \quad l = 1, \ldots, L-1, \tag{10}$$

$$\mathbf{a}^{[l]} = \max\{\gamma \mathbf{z}^{[l]}, \mathbf{z}^{[l]}\} \quad l = 1, \ldots, L-1. \tag{11}$$

$$\hat{\mathbf{y}}_b = \mathbf{a}^{[L]} = \mathbf{z}^{[L]}, \tag{12}$$

where $\mathbf{1} \in \mathbb{R}^{1 \text{ x } m_b}$ is a row vector of ones broadcasting the bias term to each observation of the mini-batch.

The weight initialization is He normal [19], which pulls the weights in each layer from a truncated normal distribution with mean $\mu = 0$ and standard deviation $\sigma = \sqrt{\frac{1}{s_{[l-1]}}}$. The weight initialization for layer $l$ is then given by:

$$\mathbf{w}^{[l]} \in \mathbb{R}^{s_{l-1} \text{ x } s_l} \sim \mathcal{N}\left(\left[0, (s_{[l-1]})^{-1}\right]\right). \tag{13}$$

The biases, $\mathbf{b}^{[l]}$, are initialized as zeros. The optimizer used was Adam optimization [20], which adaptively estimates appropriate momentum for the gradient updates.

The parameters $\mathbf{w}^{[l]}$ and $\mathbf{b}^{[l]}$ are iteratively updated by means of a gradient-based optimizer to minimize some cost function describing a distance between true labels and predictions. This is done by finding the gradients of the cost function $J$, w.r.t the trainable parameters, $\frac{\partial J}{\partial \mathbf{w}}$ and $\frac{\partial J}{\partial \mathbf{b}}$. For this work the cost is the mean squared error between the true labels and the predictions, defined by:

$$J = \frac{1}{2m_b}\left|\hat{\mathbf{y}} - \mathbf{y}\right|^2. \tag{14}$$

For supervised learning, the procedure of iteratively updating the trainable parameters, along with other hyperparameters such as model architecture, is typically done repeatedly in a validation process to assess the model's ability to generalize to unseen observations. However, the ability to generalize relies on the assumption that the unseen observations come from the same distribution as the training data, which for many real-world datasets is an invalid assumption. The target variable may depend on features not included in the model, shifts may occur in the independent or dependent variables, or the underlying process may evolve. These can all be contributors to poor generalization, and are called nonstationarity. In streaming learning, the goal is to bridge the gap between data distributions by adapting to new available observations. For this to work, labels must be available so that supervision of the updates can be achieved. In addition to adaption to changing distributions, it is of interest to "remember" older, relevant observations.

To stay stable to older observations while being plastic to new ones, the prioritized $n-$bin experience replay was developed. This experience replay configuration allows retention of observations spanning the prediction space $y \in [y_{\min}, y_{\max}]$ by splitting it into bins. We denote the buffer as $\mathcal{D} \in \mathbb{R}^{n \text{ x } N}$, where $n$ is the number of bins, and $N$ is the capacity of each bin. When observation $(x_i, y_i)$ becomes available from the data stream, it will be placed in a bin after assessment of which bin in the prediction space $y_i$ belongs to. Subsequently, the oldest observation in the same bin is discarded. Using this configuration, observations spanning the prediction space are retained, eliminating bias towards the distribution of the latest available observations, which occurs in the standard sliding window. Additionally, the mini-batch sampled from the experience replay for further learning is sampled by prioritization using the softmax function, so that each observation is assigned a probability of being sampled for training by:

$$p_j = \frac{e^{(y_j - \hat{y}_j)^2}}{\sum_{c=1}^{C} e^{(y_c - \hat{y}_c)^2}}, \quad j = 1, \ldots, C, \tag{15}$$

where $C = nN$ is the total number of observations in the replay. Assigning the probabilities based on the softmax of the model's mean squared error on the observations results in added focus on observations for which the model performs poorly, as they are more likely to be sampled. By combining the prioritized $n$-bin with the DNN using the Adam optimizer, we obtain our streaming learning algorithm, given in Algorithm 1.

$\beta_1$, $\beta_2$ and $\varepsilon$ are tunable hyperparameters for Adam optimization, $V_{\mathbf{dw}}$ and $V_{\mathbf{db}}$ are biased first moment estimates, and $S_{\mathbf{dw}}$ and $S_{\mathbf{db}}$ are biased second raw moment estimates. Superscript $c$ denotes their bias-corrected counterparts. $\alpha$ is the learning rate.

## Case study

### *Problem description*

The method presented in this work was developed in order to estimate the density of drilled lithology. This is traditionally measured using the density logging tool, which is a specialized logging while drilling (LWD) tool. Determining the density of the drilled lithology is of interest for several reasons, among them best-practice selection of drilling parameters to ensure a safe and efficient operation. Especially, accurate separation of high-density and low-density lithology can reduce the risk of dysfunctions like buckling, severe doglegs, washout and vibrations, resulting in lost time. However, the density log is delayed due to its placement behind the bit, making mechanical drilling parameters the earliest indicators of change in drilled lithology, although it is very difficult for humans to directly interpret density from these.

To eliminate the density log delay, we propose to estimate a virtual density log using parameters available at the bit, i.e. mechanical drilling parameters. Since the true label of the density log is available after the distance from the log to the bit is drilled, we can pair the delayed density log with drilling parameters measured earlier at the same depth to obtain complete input/output observations that can be used to further supervise updates to our model during operation. This turns the problem into a streaming regression problem with delayed labels. Drilling data from wells on a field operated by Equinor was used for this work. After data cleaning and removal of irrelavant observations, the training set contained approximately 740 000 observations from 5 different wellbores, while the validation set contained 365 000 observations from one wellbore. Lastly, the test set contained 227 000 observations from one wellbore. As per standard convention for deep learning, the data was normalized and scaled so that each feature had zero mean and unit variance.

Several mechanical drilling parameters are available during drilling. $v$ is the drilling velocity (ft/hr), $w$ is the weight on bit (lb), $T$ is the torque (lb-ft), $\phi$ is the drillstring rotation (rpm). The driller may directly control $v$, $w$, and $\phi$, while $T$ is dependent on a variety of factors, among them rock properties, $w$, and $\phi$. A metric commonly monitored during drilling is the mechanical specific energy, $U_{ms}$, which quantifies the energy required to remove a unit volume of rock. It is independent of the driller's actions, is different for different lithologies [21], and is given by:

$$U_{ms} = \frac{w}{a_b} + \frac{120\pi \cdot \phi \cdot T}{a_b \cdot v}, \tag{16}$$

where $a_b$ is the bit area (in$^2$). To account for weakening of the rock ahead of the bit due to flow through the nozzles, the hydraulic mechanical specific energy [22], $U_{hms}$, can be defined by:

$$U_{hms} = \frac{w}{a_b} + \frac{120\pi \cdot \phi \cdot T}{a_b \cdot v} + \frac{1154\eta \cdot \Delta p_b \cdot q}{a_b \cdot v}, \tag{17}$$

where $\eta$ is the hydraulig energy reduction factor, $\Delta p_b$ is the bit pressure drop at the nozzle (psi), and $q$ is the flow rate of drilling fluid (gpm). From the available mechanical parameters, we define the

```
 1: Load baseline model
 2: Load prioritized $n-$bin replay (pre-filled with historical data)
 3: $V_{\mathbf{dw}} = 0, S_{\mathbf{dw}} = 0, V_{\mathbf{db}} = 0, S_{\mathbf{db}} = 0,$
 4: while learning (on iteration $i$) do
 5:     Receive $x_i$                                              ⎫
 6:     Forward prop on $x_i$ to estimate $\hat{y}_i$:             ⎪
 7:     $z_i^{[1]} = \mathbf{w}^{[1]\top} x_i + \mathbf{b}^{[1]}$  ⎪
 8:     $a_i^{[1]} = g^{[1]}(z_i^{[1]})$                           ⎪  Prediction
 9:     $z_i^{[2]} = \mathbf{w}^{[2]\top} a_i^{[1]} + \mathbf{b}^{[2]}$ ⎬ when input $x_i$
10:     $a_i^{[2]} = g^{[2]}(z_i^{[2]})$                           ⎪  is available.
11:     $\vdots$                                                   ⎪
12:     $a_i^{[L]} = z_i^{[L]}$                                    ⎪
13:     $\hat{y}_i = a_i^{[L]}$                                    ⎭
14:     if new $(x_i, y_i)$ pair available then
15:         Enqueue $(x_i, y_i)$ in appropriate bin                ⎫ Prioritized
16:         Dequeue appropriate bin                                ⎬ $n-$bin expe-
17:         Calculate $p_j, j = 1, ..., C$                         ⎪ rience replay.
18:         Sample observations by probabilities $p_j$ to obtain mini-batch $\mathbf{x}_b$ ⎭
19:         Forward prop on $\mathbf{x}_b$:                        ⎫
20:         $\mathbf{z}^{[1]} = \mathbf{w}^{[1]\top} \mathbf{x}_b + \mathbf{b}^{[1]} \mathbf{1}$
21:         $\mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]})$
22:         $\mathbf{z}^{[2]} = \mathbf{w}^{[2]\top} \mathbf{a}^{[1]} + \mathbf{b}^{[2]} \mathbf{1}$
23:         $\mathbf{a}^{[2]} = g^{[2]}(\mathbf{z}^{[2]})$
24:         $\vdots$
25:         $\mathbf{a}^{[L]} = \mathbf{z}^{[L]}$
26:         $\hat{\mathbf{y}}_b = \mathbf{a}^{[L]}$
27:         Compute cost $J$
28:         Backpropagate using the chain rule to compute gradients $\frac{\partial J}{\partial \mathbf{w}}$ and $\frac{\partial J}{\partial \mathbf{b}}$
29:         $V_{\mathbf{dw}} = \beta_1 V_{\mathbf{dw}} + (1 - \beta_1) \frac{\partial J}{\partial \mathbf{w}}$
30:         $V_{\mathbf{db}} = \beta_1 V_{\mathbf{db}} + (1 - \beta_1) \frac{\partial J}{\partial \mathbf{b}}$
31:         $S_{\mathbf{dw}} = \beta_2 S_{\mathbf{dw}} + (1 - \beta_2) \frac{\partial J}{\partial \mathbf{w}}^2$   ⎬ Adam op-
32:         $S_{\mathbf{db}} = \beta_2 S_{\mathbf{db}} + (1 - \beta_2) \frac{\partial J}{\partial \mathbf{b}}^2$     timization.
33:         $V_{\mathbf{dw}}^c = \frac{V_{\mathbf{dw}}}{1-\beta_1^i}$
34:         $V_{\mathbf{db}}^c = \frac{V_{\mathbf{db}}}{1-\beta_1^i}$
35:         $S_{\mathbf{dw}}^c = \frac{S_{\mathbf{dw}}}{1-\beta_2^i}$
36:         $S_{\mathbf{db}}^c = \frac{S_{\mathbf{db}}}{1-\beta_2^i}$
37:         $\mathbf{w} = \mathbf{w} - \alpha \frac{V_{\mathbf{dw}}^c}{\sqrt{S_{\mathbf{dw}}^c} + \epsilon}$
38:         $\mathbf{b} = \mathbf{b} - \alpha \frac{V_{\mathbf{db}}^c}{\sqrt{S_{\mathbf{db}}^c} + \epsilon}$   ⎭
39:     end if
40: end while
```

**Algorithm 1.** Streaming Learning Using Prioritized $n-$bin Experience Replay & Adam Optimization

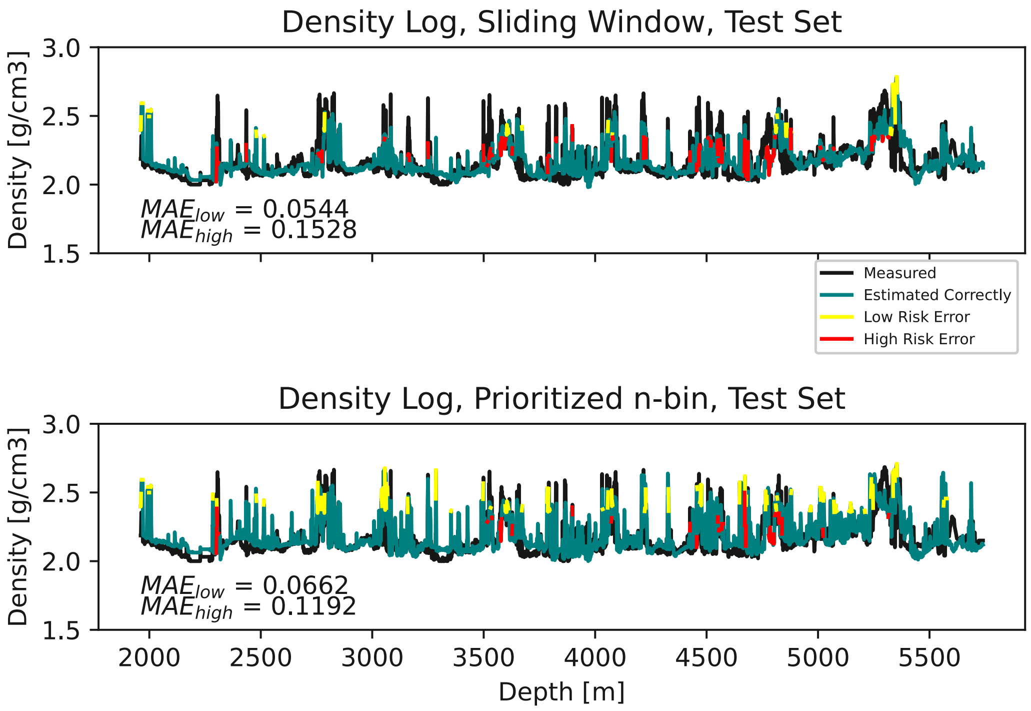

**Fig. 1. Top:** Measured and estimated density log using the standard sliding window. **Bottom:** Measured and estimated density log using the prioritized *n*-bin sliding window.

input vector of predictors for the DNN, **x**, as:

$$\mathbf{x} = [\nu, \quad \phi, \quad w, \quad T, \quad U_{ms}, \quad U_{hms}]^\top. \tag{18}$$

*Pre-training and validation*

Pre-training and validation of the baseline model was performed in an informal search. Training was performed using Adam optimization, where the training set was randomly shuffled, and divided into mini-batches of size $m_b$. The calculated gradients for each mini-batch are noisy estimates of the true gradients of the entire data set, and this additive noise is useful for avoiding getting stuck in poor local minima or saddle points early in training, and for improving generalization [23]. Neural network architecture and training configuration hyperparameters were iteratively tuned on the training and validation set split. Upon completion of this process, the streaming hyperparameters were tuned iteratively on the validation set. These hyperparameters are related to the streaming learning during operation. The density log limits on this field lies in the range 2.0–2.7 (g/cm$^3$). Tuning of the experience replay parameters resulted in 3 bins with limits at 2.115 (g/cm$^3$) and 2.535 (g/cm$^3$), effectively dividing the prediction space so that one bin retains observations below 2.115 (g/cm$^3$), the second between 2.115 (g/cm$^3$) and 2.535 (g/cm$^3$), and the last retains observations above 2.535 (g/cm$^3$). These parameters can be seen in Table 1, which summarizes the hyperparameters. We can also see that the learning rate is decreased by one order of magnitude for the streaming learning, which was necessary in order to facilitate stability.

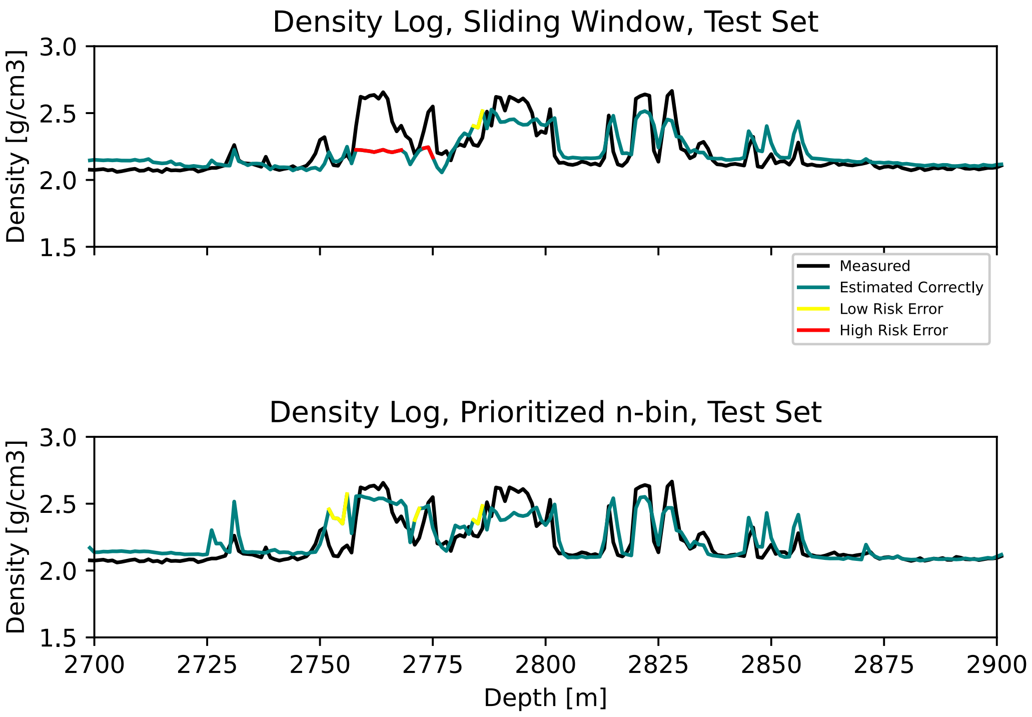

**Fig. 2.** Zoom 1. **Top:** Measured and estimated density log using the standard sliding window. **Bottom:** Measured and estimated density log using the prioritized *n*-bin sliding window.

**Table 1**
Summary of hyperparameters.

| Pre-training hyperparameter | Value |
|---|---|
| Learning rate | $7.5 \times 10^{-5}$ |
| Hidden layers | 3 |
| Neurons in hidden layers | 12 |
| Mini-batch size | 128 |
| Epochs | 25 |
| Optimizer | Adam |
| **Streaming learning hyperparameter** | **Value** |
| Learning rate | $7.5 \times 10^{-6}$ |
| Experience replay bins | 3 |
| Experience replay bin sizes | 256 |
| Experience replay bin limits | [2.115, 2.535] |
| Mini-batch size | 16 |
| Epochs | 1 |
| Optimizer | Adam |

## *Test set results*

After pre-training and validation of the baseline model, it was deployed for streaming regression during operation on the test set, where the density logging tool is placed 20 m behind the bit. This was done using both the prioritized *n*-bin experience replay, and a standard sliding window for comparison. As summarized in Table 1, the prioritized *n*-bin replay consisted of 3 bins, each containing 256 observations at all times. The standard sliding window used for comparison contained

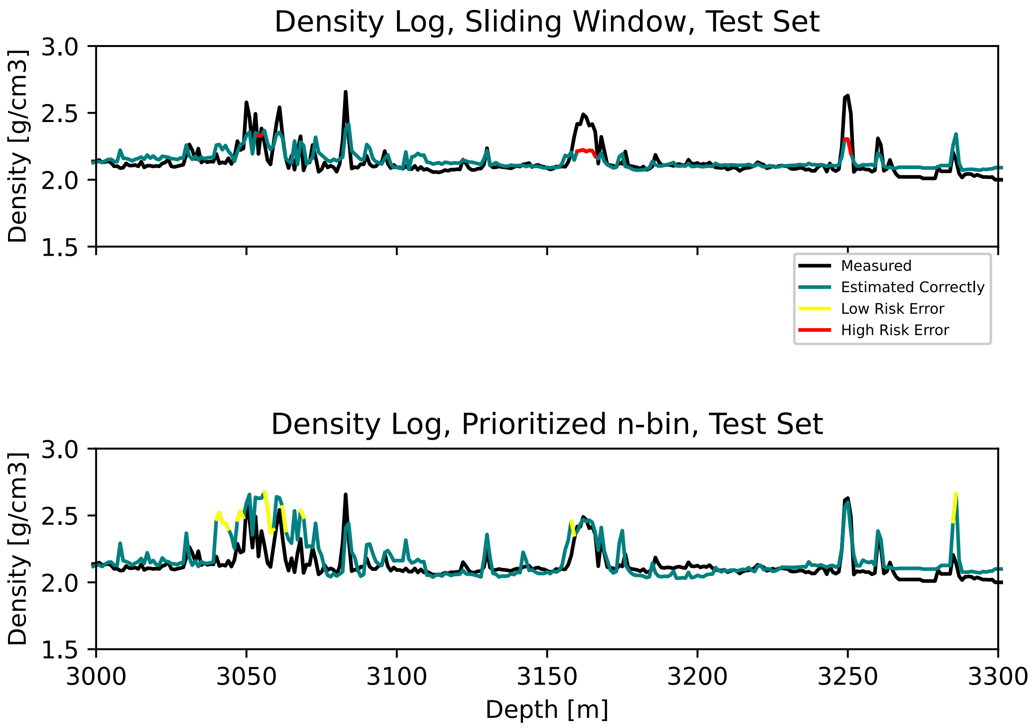

**Fig. 3.** Zoom 2. **Top:** Measured and estimated density log using the standard sliding window. **Bottom:** Measured and estimated density log using the prioritized *n*-bin sliding window.

the same amount of observations in total, 768. Although the algorithm is run on data in time domain, the density log (along with other LWD tools) are most interesting in depth domain. For this reason, the presented results are converted to depth domain with equidistant points at 1 m resolution by downsampling. At every integer depth, observations within 0.5 m are averaged. Since detection of the hard stringers is important, we evaluate both approaches in terms of mean absolute error (MAE) separately for low density ($< 2.35$ (g/cm$^3$)) and high density ($\geq 2.35$ (g/cm$^3$)) observations.

Fig. 1 illustrates measured and estimated at-bit density vs. depth on the entire test set for both methods. We define a false high density estimate a *low risk* error. If the driller takes action, lowering $T$ and increasing $w$ based on such an estimate, the drilling will simply be sub-optimal. Conversely, we define a false low density estimate as a *high risk* error. If hard stringers are not detected, and action is not taken, risk of dysfunctions such as buckling, severe doglegs, washout and vibrations are increased. On the test set as a whole, the prioritized $n$-bin experience replay leads to a 22% increase in MAE for true low density observations and a 22% decrease in MAE for true high density observations, compared to the standard sliding window.

We wish to investigate these results in more detail to provide some insight into the effect of applying the prioritized $n$-replay, both on low density and high density observations. First, we perform paired, two-tailed $t$-tests to investigate the statistical significance of the changes in MAE, through obtaining $p$-values. The null hypothesis becomes $H_0 : \mu_1 = \mu_2$, where $\mu_1$ is the population mean absolute error for the standard sliding window approach, and $\mu_2$ is the population mean absolute error for our method. For this test, we select a significance level of $\alpha_0 = 0.05$. To quantify a standardized *effect size*, we also calculate Cohen's $d$. This value, in combination with $\alpha_0$, can in turn be used to calculate the statistical power, $1 - \beta$, where $\beta$ is the probability of a type II error. Table 2 summarizes the results of our statistical analysis, where $m$ is number of observations, MAE$_s$ is the

**Table 2**
Results of power analysis.

| True observation value | $m$ | $\text{MAE}_s$ [g/cm$^3$] | $\text{MAE}_n$ [g/cm$^3$] | $\Delta\text{MAE}$ [g/cm$^3$] | $d$ | $p$ | $1-\beta$ |
|---|---|---|---|---|---|---|---|
| $< 2.35$ | 3422 | 0.0544 | 0.0662 | $-0.01175$ | $-0.3021$ | $4.63 \times 10^{-35}$ | $\approx 1$ |
| $\geq 2.35$ | 358 | 0.1528 | 0.1192 | 0.03359 | 0.4578 | $2.51 \times 10^{-9}$ | $\approx 1$ |

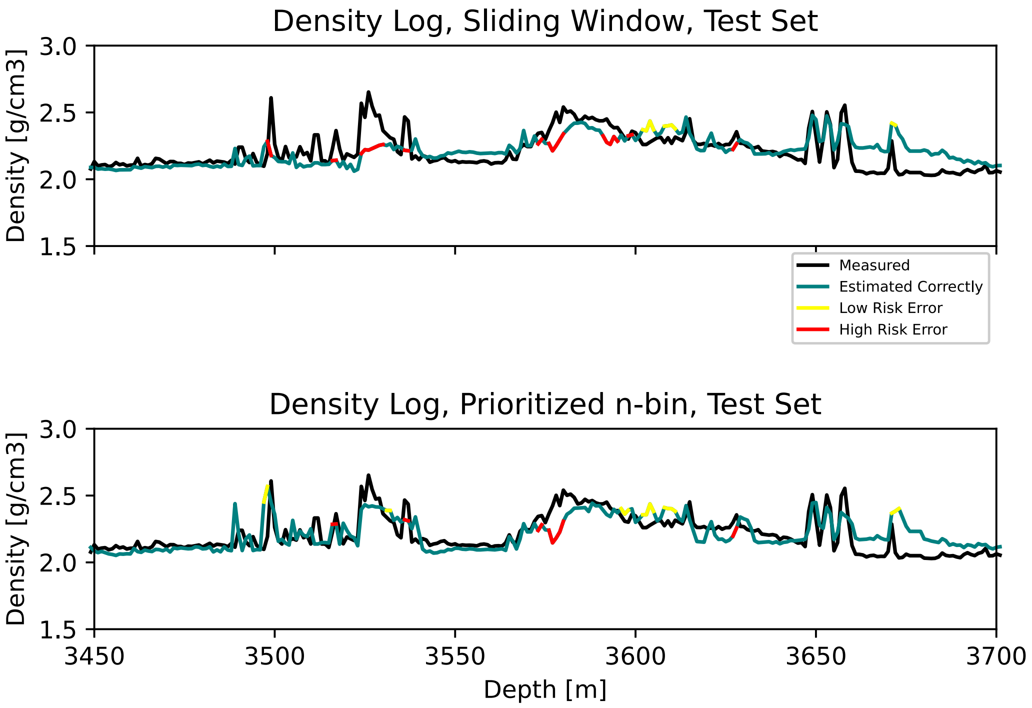

**Fig. 4.** Zoom 3 **Top:** Measured and estimated density log using the standard sliding window. **Bottom:** Measured and estimated density log using the prioritized $n$-bin sliding window.

mean absolute error using a sliding window, $\text{MAE}_n$ is the mean absolute error using our method, and $\Delta\text{MAE} = \text{MAE}_s - \text{MAE}_n$. Through the $t$-tests, we confirm the statistical significance at our chosen significance level. Observing Cohen's $d$ show that for true low density observations, our method leads to a standardized effect size of $d = -0.3021$ (where the negative sign signifies worsening), while for the true high density observations, $d = 0.4578$. Interpreting these along a continuum, as proposed in literature [24], where 0.2, 0.5, and 0.8 are low, medium and high effect sizes, we see that our approach leads to a small to medium worsening on low density observations, and a medium to large improvement on high density observations.

Figs. 2–5 are zoomed plots on the test set. In Fig. 2, at 2755 m, a hard stringer occurs. Using the standard sliding window, this event is completely undetected. We can see at 2779 m that the model accurately detects the next stringer. Here, 24 m further down, high density observations from the missed stringer has been passed by the density logging tool (20 m behind), and these observations are available for training in the sliding window. We can assume that the model misses the first stringer since the sliding window only contains observations from the earlier low density observations, resulting in *catastrophic forgetting*. Using our approach, we can see that also the stringer missed by the standard sliding window is detected.

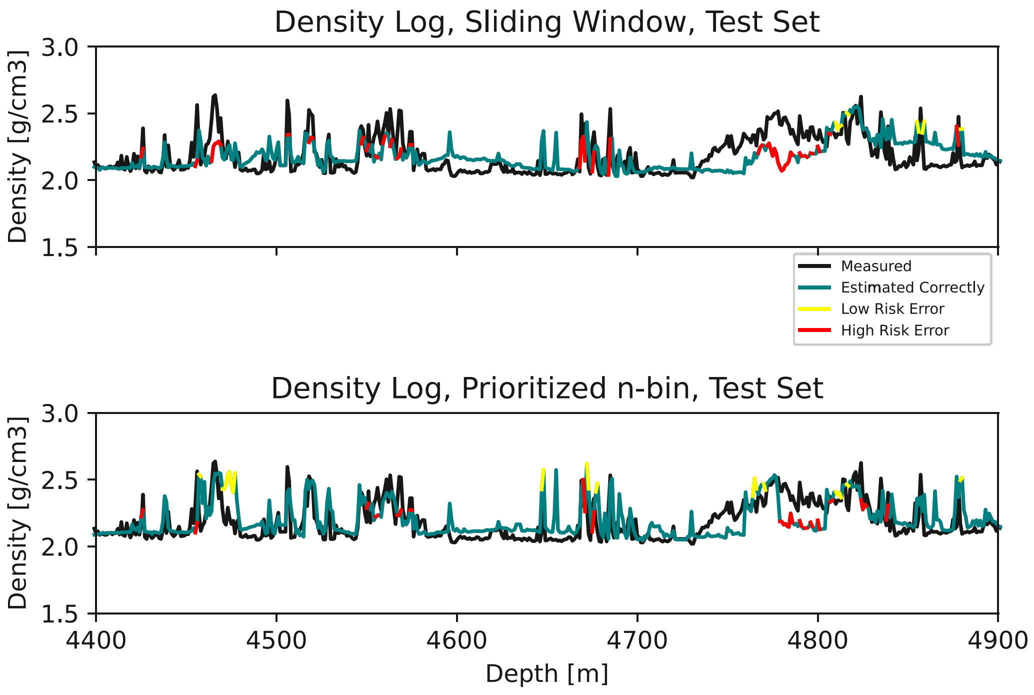

**Fig. 5.** Zoom 4 **Top:** Measured and estimated density log using the standard sliding window. **Bottom:** Measured and estimated density log using the prioritized *n*-bin sliding window.

In Fig. 3, we can see that the sliding window misses the stringer at 3157–3170 m, along with the one at 3250 m. Our approach accurately detects these events. At the same time, we observe that some observations at approximately 3050 m and 3285 are overestimated. In Fig. 4, stringers at 3500 m and 3523–3536 m are missed by the sliding window, but detected by our approach. Lastly, Fig. 5 shows missed stringers by the sliding window at 4450–4530 m, along with a poor transition to low density observations at approximately 4850 m. These are all improved using our approach. However, we observe some false high estimates at 4650–4700 m.

Since the density log is naturally divided into low- and high density observations, we rephrase the problem into a binary classification problem. Although this is an oversimplification, it adds to the previous analysis in terms of detection of hard stringers, and high risk/low risk errors, as defined previously. As can be seen from $m$ in Table 2, less than 10% of the observations in the test set are stringers, which makes this an unbalanced data set. Fig. 6 shows the resulting confusion matrices for both approaches, dividing the observations into low density and high density observations as previously. We see that the sliding window is very accurate in prediction of low density observations, with an accuracy rate of 0.97. However, only an accuracy of 0.55 is achieved for high density observations. Using the prioritized $n$-bin, the accuracy on low density observations is slightly decreased, to 0.92, while detection of high density observations is greatly improved, scoring an accuracy of 0.8. The balanced accuracies for the sliding window and the prioritized $n$-bin are 0.76 and 0.86, respectively. Due to drillstring compression and elongation, measurement error on depth can occur. To account for this, a 1 m acceptance was implemented, meaning that if a prediction is within 1 m of the correct label, it is accepted as correct. The confusion matrices using the 1 m acceptance are shown in Fig. 7. We observe that the results for both approaches are improved. Here, the balanced accuracies are 0.89 for the sliding window and 0.95 for our approach.

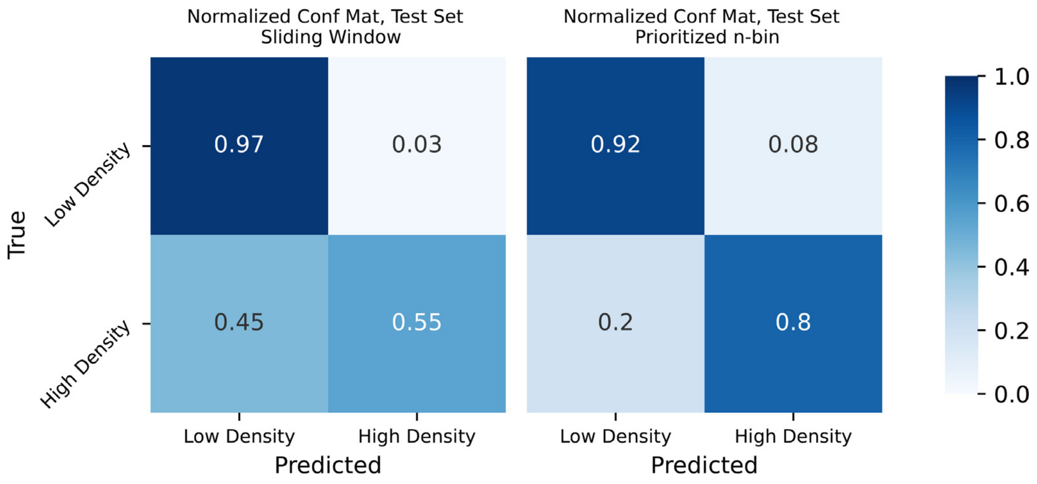

**Fig. 6.** Confusion matrices. **Left:** Standard sliding window. **Right:** Prioritized $n$-bin method.

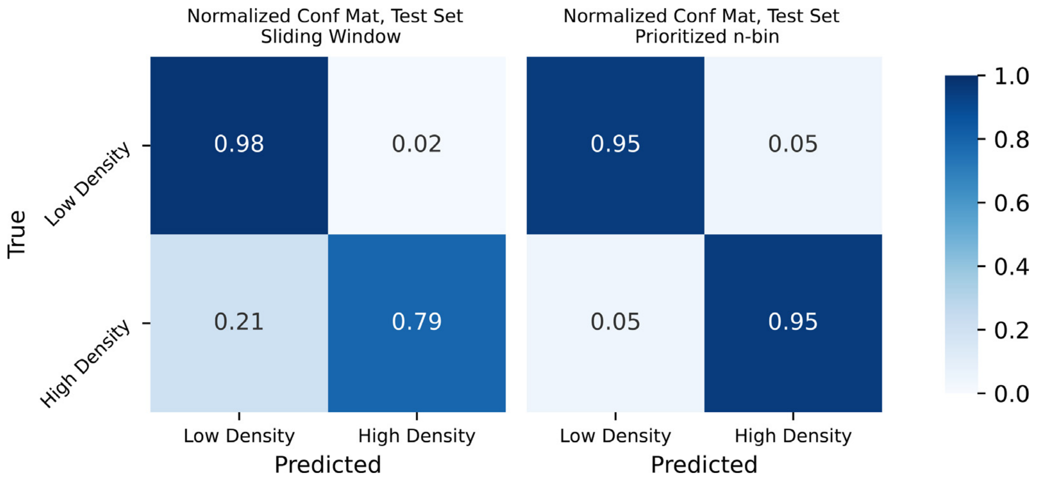

**Fig. 7.** Confusion matrices applying a 1 m acceptance. **Left:** Standard sliding window. **Right:** Prioritized $n$-bin method.

## Conclusions

A divided and prioritized experience replay suited for streaming regression has been presented, making use of known ideas such as retention of relevant observations along with prioritization. This makes the model less biased to the distribution of the latest available observations, and results in more frequent sampling of observations where the model performs poorly.

Comparison to a standard sliding window has been made on real-world data. From our results, we can deduce that the standard sliding window results in forgetting of old, rare events, leading to failure to detect them. Especially in cases where exclusively common events have been observed, the model becomes biased towards these observations, failing to detect new, important events. The presented $n$-bin method, on the other hand, retains observations from the entire range of the prediction space, resulting in more accurate estimates for these observations at a small cost in accuracy on the common events. Also, some false detections are observed. In addition to analyses on the regression formulation, a simplified rephrasing to a binary classification problem has been performed. These results divide the observations into two classes to provide added insight into the improved performance on the rare events.

## Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

The authors wish to thank Equinor and NTNU for funding this project. We would also like to thank the Troll licence for sharing their data.

## References

[1] G. Ditzler, M. Roveri, C. Alippi, R. Polikar, Learning in nonstationary environments: a survey, IEEE Comput. Intell. Mag. 10 (4) (2015) 12–25, doi:10.1109/MCI.2015.2471196.
[2] R. Elwell, R. Polikar, Incremental learning of concept drift in nonstationary environments, IEEE Trans. Neural Netw. 22 (10) (2011) 1517–1531, doi:10.1109/TNN.2011.2160459.
[3] M. McCloskey, N.J. Cohen, Catastrophic interference in connectionist networks: the sequential learning problem, in: Psychology of Learning and Motivation, vol. 24, Elsevier, 1989, pp. 109–165, doi:10.1016/S0079-7421(08)60536-8.
[4] T.R. Hoens, R. Polikar, N.V. Chawla, Learning from streaming data with concept drift and imbalance: an overview, Prog. Artif. Intell. 1 (1) (2012) 89–101. 10.1007%2Fs13748-011-0008-0
[5] S. Grossberg, Nonlinear neural networks: principles, mechanisms, and architectures, Neural Netw. 1 (1) (1988) 17–61, doi:10.1016/0893-6080(88)90021-4.
[6] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533, doi:10.1038/nature14236.
[7] C.C. Aggarwal, J. Han, J. Wang, P.S. Yu, A framework for on-demand classification of evolving data streams, IEEE Trans. Knowl. Data Eng. 18 (5) (2006) 577–589, doi:10.1109/TKDE.2006.69.
[8] H. Abdulsalam, D.B. Skillicorn, P. Martin, Classification using streaming random forests, IEEE Trans. Knowl. Data Eng. 23 (1) (2010) 22–36, doi:10.1109/TKDE.2010.36.
[9] C.-D. Wang, J.-H. Lai, D. Huang, W.-S. Zheng, Svstream: a support vector-based algorithm for clustering data streams, IEEE Trans. Knowl. Data Eng. 25 (6) (2011) 1410–1424, doi:10.1109/TKDE.2011.263.
[10] O. Dekel, P.M. Long, Y. Singer, Online learning of multiple tasks with a shared loss, J. Mach. Learn. Res. 8 (10) (2007) 2233–2264.
[11] C. Li, F. Wei, W. Dong, X. Wang, Q. Liu, X. Zhang, Dynamic structure embedded online multiple-output regression for streaming data, IEEE Trans. Pattern Anal. Mach. Intell. 41 (2) (2018) 323–336, doi:10.1109/TPAMI.2018.2794446.
[12] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A.A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, Overcoming catastrophic forgetting in neural networks, Proc. Natl. Acad. Sci. 114 (13) (2017) 3521–3526, doi:10.1073/pnas.1611835114.
[13] Z. Li, D. Hoiem, Learning without forgetting, IEEE Trans. Pattern Anal. Mach. Intell. 40 (12) (2017) 2935–2947, doi:10.1109/TPAMI.2017.2773081.
[14] H. Wang, W. Fan, P.S. Yu, J. Han, Mining concept-drifting data streams using ensemble classifiers, in: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003, pp. 226–235, doi:10.1145/956750.956778.
[15] K. Nishida, K. Yamauchi, Adaptive classifiers-ensemble system for tracking concept drift, in: Proceedings of the International Conference on Machine Learning and Cybernetics, vol. 6, IEEE, 2007, pp. 3607–3612, doi:10.1109/ICMLC.2007.4370772.
[16] T.L. Hayes, N.D. Cahill, C. Kanan, Memory efficient experience replay for streaming learning, in: Proceedings of the International Conference on Robotics and Automation (ICRA), IEEE, 2019, pp. 9769–9776, doi:10.1109/ICRA.2019.8793982.
[17] T. Schaul, J. Quan, I. Antonoglou, D. Silver, Prioritized experience replay, in: Proceedings of the International Conference on Learning Representations, ICLR, San Juan, Puerto Rico, May 2–4, 2016 https://arxiv.org/abs/1511.05952v4.
[18] M.L. Arnø, J.-M. Godhavn, O.M. Aamo, At-bit estimation of rock density from real-time drilling data using deep learning with online calibration, J. Pet. Sci. Eng. (2021), doi:10.1016/j.petrol.2021.109006.
[19] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, December 7–13, 2015 https://doi.ieeecomputersociety.org/10.1109/ICCV.2015.123.
[20] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, in: Proceedings of the International Conference on Learning Representations (ICLR), San Diego, California, USA, May 7–9, 2015 https://arxiv.org/abs/1412.6980.
[21] F.E. Dupriest, W.L. Koederitz, Maximizing drill rates with real-time surveillance of mechanical specific energy, in: Proceedings of the SPE/IADC Drilling Conference, Amsterdam, Netherlands, February 23–25, Society of Petroleum Engineers, 2005. SPE-92194-MS https://doi.org/10.2118/92194-MS
[22] A.U. Osarogiagbon, O. Oloruntobi, F. Khan, R. Venkatesan, S. Butt, Gamma ray log generation from drilling parameters using deep learning, J. Pet. Sci. Eng. 195 (2020) 107906, doi:10.1016/j.petrol.2020.107906.

[23] N.S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, T. PTP, On large-batch training for deep learning: generalization gap and sharp minima, in: Proceedings of the International Conference of Learning Representations, ICLR, Toulon, France, April 24–26, 2017 https://arxiv.org/abs/1609.04836v2.

[24] L.G. Portney, Foundations of Clinical Research: Applications to Evidence-Based Practice, fourth ed., FA Davis Company, Pennsylvania, United States, 2020.