Failure Prediction with Adaptive Multi-Scale Sampling and Activation Pattern Regularization

Yujin Tang, Shinya Wada and Kiyohito Yoshihara
KDDI Research, Inc.
2-1-15 Ohara, Fujimino, Saitama, 356-8502 Japan
Email: {yu-tang, sh-wada, yosshy}@kddi-research.jp

***Abstract*—We treat failure prediction in a supervised learning framework using a convolutional neural network (CNN). Due to the nature of the problem, learning a CNN model on this kind of dataset is generally associated with three primary problems: 1) negative samples (indicating a healthy system) outnumber positives (indicating system failures) by a great margin; 2) implementation design often requires chopping an original time series into sub-sequences, defining a segmentation window size with sufficient data augmentation and avoiding serious multiple-instance learning issue is non-trivial; 3) positive samples may have a common underlying cause and thus present similar features, negative samples can have various latent characteristics which can "distract" CNN in the learning process. While the first problem has been extensively discussed in literatures, the last two issues are less explored in the context of deep learning using CNN. We mitigate the second problem by introducing a random variable on sample scaling parameters, whose distribution's parameters are jointly learnt with CNN and leads to what we call adaptive multi-scale sampling (AMS). To address the third problem, we propose activation pattern regularization (APR) on only positive samples such that the CNN focuses on learning representations pertaining to the underlying common cause. We demonstrate the effectiveness of our proposals on a past Kaggle contest dataset that predicts seizures from EEG data. Compared to the baseline method with a CNN trained in traditional scheme, we observe significant performance improvement for both proposed methods. When combined, our model *without any sophisticated hyper-parameter tuning or ensemble methods* shows a near 10% relative improvement on AUROC and is able to send us to the 14th place on the contest's leaderboard while the highest rank the baseline can reach is 77th.**

***Keywords*—Time sequence, deep learning, adaptive multi-scale sampling, activation pattern regularization.**

I. Introduction

Time series data mining has attracted a remarkable amount of interest in both academic and industry communities. Among which the branch of failure prediction, due to the drastic need in a variety of applications across several fields such as medical research, finance, system monitoring and etc, is given serious attention [1]–[4]. In this paper, we treat failure prediction in time series in a supervised learning framework. Concretely, in the training dataset, data samples in a pre-defined time frame prior to failures are regarded as positive, and those located relatively far away between the occurrences of failures are negative samples. Realtime prediction can then be performed on the target data stream with the trained model. A plentiful literatures ranging from heuristic methods to solutions utilizing probabilistic models on the topic of supervised learning in the time series domain have been published [5]–[9]. Encouraged by the recent success of deep learning, some researchers have shown the effectiveness of applying deep learning to time series problems [10]–[14]. Because deep learning is demonstrated to save engineers the effort on feature crafting while delivering strong results, we adopt convolutional neural network (CNN) as our predictive model in the same vein.

Due to the nature of the dataset, learning a CNN model is generally associated with three primary problems. First of all, the dataset can be seriously imbalanced because statistically negative samples often outnumber positive samples significantly. And the learning process may be overwhelmed by the massive amount of negative samples, in the worst case it can produce a useless model that predicts every sample to be negative. Secondly, implementation design often requires chopping an original time series into sub-sequences which is also an important step for data augmentation when training a deep learning model such as CNN. An appropriate window size for segmentation can provide the training process with sufficient amount of training data and avoid serious multiple-instance learning issues wherein a great portion of sub-sequences from an original positive sequence does not actually carry representative features of the positive class. Defining such a window size with sufficient high-quality data augmentation is non-trivial without domain knowledge. Last but not least, when considering predicting failures of a specific type, positive samples occurred due to a common cause, negative samples do not necessarily follow this restriction. In fact they present various latent characteristics (E.g., in the case of predicting seizure from EEG data, EEG waves can be different when the subject is undergoing different activities [15]) which can prevent CNN from learning discriminative representations.

The first problem has been extensively discussed in literatures and is technically solved. The solutions include down-sampling/up-sampling the negative/positive data samples, assigning higher weights to the positive class, etc [16]. However, the last two problems are less explored, specifically in the context of training a CNN. To address the second problem, we define a random variable on a set of sample scaling parameters. Following the random variable's distribution, we sample sub-sequences of different lengths from the original series and scale them according to the sampled scaling parameter to a common length. We fit the CNN to these scaled sub-sequences,

2375-9259/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDMW.2017.17

at the end of every k iterations we update the scaling parameters' distribution's parameters and thus it's trained jointly with the CNN. We show that as the training progresses, the distribution we sample from converges and is going to peak on a few scales that are optimal for the task. We call this process adaptive multi-scale sampling (AMS). With the assumption that for a specific kind of failure, positive samples occur due to a common cause, we mitigate the last problem using activation pattern regularization (APR) which acts as an extra term to the objective function that regularizes the activation patterns of only positive samples. Concretely speaking, when training the CNN we construct in each mini-batch a Gramian matrix for each positive sample that represents its activation pattern and we minimize the variances of the matrices' entries. This term shifts the CNN's attention to extracting discriminative features. To demonstrate the advantage of our proposals, we conducted experiments on seizure prediction EEG datasets from a past Kaggle contest. Compared to a vanilla CNN training scheme, we observe performance improvement for both AMS and APR. When combined, our model *without any sophisticated hyper-parameter tuning or ensemble methods* shows a near 10% relative improvement on AUROC and is able to send us to the 14th place on the contest's leaderboard while the highest rank the baseline can reach is 77th.

Our main contribution in this paper is the proposal of the deep learning scheme with a combination of AMS and APR for failure prediction. To the best of our knowledge, we are the first to give tentative solutions to the aforementioned last two problems in the context of training a CNN model for time series classification and have demonstrated their effectiveness on real-world datasets.

The rest of this paper is organized as follows. We briefly introduce some related literatures in Sec. II. We then give a detailed description of AMS and APR in Sec. III, following which experiments to prove the effectiveness of our proposals are introduced in Sec. IV. Finally in Sec. V, we conclude our work and give possible directions for future works.

II. Related Work

A. Failure Prediction

Failure prediction is a topic that has attracted interest for more than 30 years and has seen myriad applications in various fields such as system monitoring, finance, medical analysis, etc. An exhaustive survey of early works on this topic can be found in [1]. Although in the early years, the combination of domain knowledge and rule-based approaches proved to be successful in some scenarios, these methods have very limited generalization to noisy observations or scenarios with highly variant environments. With the growth of computing power and the abundance of data, more and more researches are transiting to data-driven learning methods, these methods are demonstrated to outperform simple heuristic methods by a large margin. For instance, the authors in [2] proposed to predict hard driver failures using a naive Bayesian classifier that takes multiple-instance learning into its account. The derived probabilistic model is considerably better than the threshold method currently implemented in drivers. By comparing their model to other supervised learning methods such as support vector machine (SVM), the authors advocate non-parametric statistical tests is preferred for predicting rare events in time series data. In another work, [4] employed an ensemble of auto regressive moving average model and fault tree analysis for failure predictions in data centers, and have achieved a very high prediction accuracy. With a different approach, the authors in [3] proposed a semi-supervised learning method for oil production wells' failure prediction wherein they incorporated domain knowledge into their learning framework to extract robust features and employed clustering-based labelling methods to mitigate the noisy labelling problem. Although failure prediction using data-driven learning methods can have different forms, we take the supervised learning approach in this paper and treat failure prediction as a classification problem.

B. Time Series Classification

A plenty of literatures ranging from heuristic methods to solutions utilizing probabilistic models on the topic of time series classification have been published [5]–[9]. Encouraged by the huge success of deep learning applications recently, some researchers have demonstrated the effectiveness of applying deep learning models to time series classification problems. For instance, some approaches encode raw time series inputs into images first, and then fit CNN with 2D convolutional filters to the images, thus reducing the problem entirely to image classification and all relevant tools and parameter tuning techniques can be exploited [10], [11]. In another flavour, [12], [13] chopped the target time series input into equal length segments and convolutions are then performed on these transformed data. In these cases, 1D convolutional filters shared across data channels are applied along the time dimension and a fusion mechanism such as probability voting is adopted to determine the label of the original time series. To get rid of the uniform input length constraint, [14] adopted a sequence-to-sequence model that is common in natural language tasks, wherein the authors squash the original input sequences of various lengths into common length feature sequences, these feature sequences are then fed into the trailing fully-connected layers. To help learning the feature sequences, they also introduced the attention mechanism that helps the model to extract discriminative features by focusing on the most relevant part of the original sequences during feature generation. Crafting informative features requires domain knowledge and incurs heavy load on engineering. With such a background, deep learning's ability to automatically extract task oriented representations is a huge relief for engineers, and for the very same reason we adopt CNN as our predictive model.

III. Our Proposals

A. Network Architecture and Notations

Fig. 1 shows the architecture of the CNN we adopt as our predictive model, and the notations we use frequently to describe our ideas in this paper are summarized in Table

I. In our network, convolutional layers have 1D filters that convolves along the time dimension with parameters [(32, 8, 4), (64, 5, 2), (64, 2, 2)] the format of which is (*#filter*, *filter_size*, *stride*). Both fully connected layers have 1024 units. All layers have ReLU activations [17] except the output layer which has a sigmoid activation because we treat the problem as a binary classification task.

It is noteworthy that bearing two thoughts in mind, we purposefully designed the network to be simple and we did not tune its architecture in our experiments. The first reason is that we want to make sure all the performance advantages are from our proposals and thus we restricted the network to its minimum and avoided fancy setups such as batch normalization or skipped connections though they are theoretically compatible. Secondly, it is well known that tuning the network architecture to a good shape on a dataset brings a performance boost. However, our aim in this work is not to push the state-of-the-art on a specific dataset, but to demonstrate the problem in the CNN training scheme specific to failure prediction and prove the effectiveness of our proposals.

The objective function that we minimize for our learning task is defined as follow:

$$l(\mathbf{X}, \mathbf{y}; \theta, \phi) = \mathbb{E}[f_\theta(\mathbf{X}, \mathbf{y}) + \lambda g_\theta(\mathbf{X}) \mid \phi] \tag{1}$$

where $f_\theta(\mathbf{X}, \mathbf{y})$ is the binary cross-entropy loss and $g_\theta(\mathbf{X})$ is the term from APR, both of which are parameterized by the CNN's parameters θ. λ is a hyper-parameter that determines how much weight we should put on APR. The expectation is taken over the training dataset that depends on the parameter ϕ from AMS. If we remove both AMS and APR, the objective function becomes simply $f_\theta(\mathbf{X}, \mathbf{y})$ and is the typical loss for training CNN for a binary classification problem. Also notice that neither AMS nor APR adds new network parameters to CNN, the network's capacity remains the same. Therefore, our proposal is purely an extension of the traditional CNN training scheme.

B. Adaptive Multi-Scale Sampling (AMS)

Although designing a CNN that accepts variable input sequence length is possible, input length constraint is not the only problem to address. Training a deep learning model requires a huge amount of data, and for the performance's sake data augmentation is usually a necessity. As is briefly described in Sec. I, defining sub-sequence lengths that provide sufficient high-quality data augmentation without causing serious multiple-instance learning problems is non-trivial without domain knowledge. A natural question to raise here is can we define it as a variable and learn it jointly with the CNN? The answer turns out to be yes and is exactly what AMS does.

1) Multi-Scale Sampling: In AMS, we define several sample scaling parameters $\{s_1, s_2, \cdots\}$ and assign a random variable S over them. S is categorical, and we define its distribution to be $P(S = s_i) \triangleq \frac{\exp(\phi_i/\eta)}{\sum_j \exp(\phi_j/\eta)}$ where $\phi = [\phi_1, \phi_2, \cdots, \phi_{|S|}]$ are its parameters and η is a constant that acts as the temperature for this softmax function. We initialize ϕ to be a vector of all ones and during CNN training we randomly crop a mini-batch of sub-sequences of lengths $l \cdot S$ by drawing samples from $P(S)$. We scale all sub-sequences to be of a common length l that is pre-defined as our network's input dimension and feed the mini-batch to the CNN. Mathematically, let us denote $x_{1,\cdots,lS}$ as a randomly cropped sub-sequence, $x'_{1,\cdots,l}$ as the sub-sequence after scaling, and $h_S(\cdot)$ as the scaling operator, we can have several scaling strategies. For example, we can take the mean of every S samples from $x_{1,\cdots,lS}$ and hence $h_S(x_{1,\cdots,lS}) = \{x'_{1,\cdots,l} \mid x'_i = mean(x_{(i-1)S+1,\cdots,iS})\}$. Or in a simpler form, we can just define x'_i to be the j-th element of every S consecutive samples which leads to $h_S(x_{1,\cdots,lS}) = \{x'_{1,\cdots,l} \mid x'_i = x_{(i-1)S+j}\}$.

To keep notations uncluttered, we merge some operators into

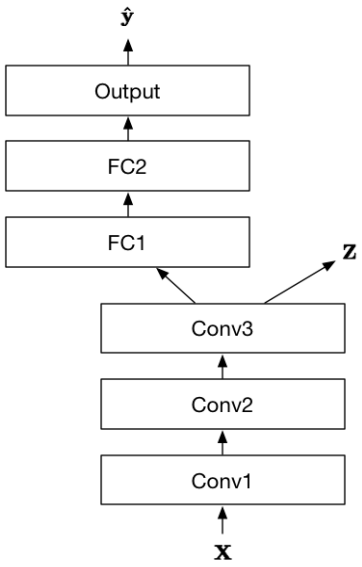

Fig. 1. Network architecture.

TABLE I
NOTATIONS

Notation	Description
N	mini-batch size
l	CNN input sequence length
m	number of data channels (E.g. #senor)
$\mathbf{X}$	$\mathbf{X} \in \mathcal{R}^{N \times l \times m}$ is a mini-batch of input sequences
$\mathbf{X}_i$	$\mathbf{X}_i \in \mathcal{R}^{l \times m}$ is the i-th sample in the batch
$\mathbf{y}$	$\mathbf{y} = [y_1, \cdots, y_N]$ is the vector of binary valued ground truths
$\hat{\mathbf{y}}$	$\hat{\mathbf{y}} = [\hat{y_1}, \cdots, \hat{y_N}]$ are CNN predictions
n	number of filters in the last convolutional layer
l'	filter size in the last convolutional layer
z_i	$z_i \in \mathcal{R}^{l' \times n}$ is sample i's activation of the last conv layer
$\mathbf{Z}$	$\mathbf{Z} \in \mathcal{R}^{N \times n^2}$ are flattened Gramian matrices
$\mathbf{Z}_i$	$\mathbf{Z}_i \in \mathcal{R}^{1 \times n^2}$ is a flattened Gramian matrix from sample i
p	number of positive samples in the mini-batch
$\mathbf{A}$	$\mathbf{A} \in \mathcal{R}^{N \times p}$ is the matrix that selects positive samples
S	random variable over candidate scaling parameters
$h_S(\cdot)$	sequence scaling operator
$\pi_\theta(\cdot)$	CNN parameterized by θ
ϕ	$\phi \in \mathcal{R}^{\lvert S \rvert}$ are parameters of S's distribution
η	temperature for softmax function
λ	weight for APR

$h_S(\cdot)$ and $\pi_\theta(\cdot)$. Firstly, we merge sub-sequencing operator into $h_S(\cdot)$ such that if its input is longer than $l{\cdot}S$ then sub-sequencing (random crop in training; segmentation with minimum overlapping in evaluation) is performed prior to scaling. Furthermore, because we segment the original time series, we need a fusion mechanism for predictions. E.g., if we want the prediction score of the i-th sample in a test batch, $\pi_\theta\big(h_S(\mathbf{X}_i)\big)$ gives several scores each of which for the sub-sequences originated from $h_S(\mathbf{X}_i)$, we need to merge these scores to get the prediction score for $\mathbf{X}_i$. In this paper, we define the score for the original series $score(\mathbf{X}_i) \triangleq \mathbb{E}_{P(S)}[mean(\pi_\theta\big(h_S(\mathbf{X}_i)\big))]$, we assume this fusion operation is merged into $\pi_\theta(\cdot)$ and is applied whenever necessary.

2) Adaptive Update: At the beginning of training, a mini-batch consists of sub-sequences of multiple scales, each of which has equal sampling weight (because we initialize ϕ to be a vector of all ones, thus $P(S)$ is uniform). We train the CNN with sub-sequences of mixed scales, and at the end of every k-th training iteration, we randomly sample some time series from the entire training set, apply scaling operator $h_S(\cdot)$ to this set and feed the scaled dataset to the CNN trained so far. We then calculate the first term in Eq. (1) by evaluating Eq. (2), the definition of $f_\theta(h_S(\mathbf{X}), \mathbf{y})$ is given in Eq. (3). Minimization of Eq. (2) thus becomes jointly learning the CNN's parameters θ and the parameters ϕ for $P(S)$. In our experiments, at the end of every k CNN training iterations, we perform a one-step gradient descent on ϕ using Eq. (4). The gradient of ϕ is easy to derive and is given in Eq. (5), where matrix $\mathbf{J}$ has entry $\mathbf{J}_{ij} = \frac{1}{\eta}P(S_i)\big(1 - P(S_i)\big)$ if $i = j$ or $\mathbf{J}_{ij} = -\frac{1}{\eta}P(S_i)P(S_j)$ if $i{\neq}j$. Adaptive update is similar to one iteration of back propagation, the additional computational cost is negligible when considering its parameters' size.

$$\mathbb{E}[f_\theta(\mathbf{X}, \mathbf{y}) \mid \phi] = \mathop{\mathbb{E}}_{P(S)}[f_\theta(h_S(\mathbf{X}), \mathbf{y})] \tag{2}$$

$$\begin{aligned} f_\theta(h_S(\mathbf{X}), \mathbf{y}) = -\frac{1}{N}\sum_{i=1}^{N} \big[y_i \log\big(\pi_\theta(h_S(\mathbf{X}_i))\big) \\ +(1 - y_i)\log\big(1 - \pi_\theta(h_S(\mathbf{X}_i))\big)\big] \end{aligned} \tag{3}$$

$$\phi \coloneqq \phi - \nabla\phi \tag{4}$$

$$\nabla\phi = \mathbf{J}^\intercal\mathbf{L} = \begin{bmatrix} \frac{\partial P(S_1)}{\partial\phi_1} & \cdots & \frac{\partial P(S_1)}{\partial\phi_{|S|}} \\ \vdots & \ddots & \vdots \\ \frac{\partial P(S_{|S|})}{\partial\phi_1} & \cdots & \frac{\partial P(S_{|S|})}{\partial\phi_{|S|}} \end{bmatrix}^\intercal \begin{bmatrix} f_\theta(h_{S_1}(\mathbf{X}), \mathbf{y}) \\ \vdots \\ f_\theta(h_{S_{|S|}}(\mathbf{X}), \mathbf{y}) \end{bmatrix} \tag{5}$$

C. Activation Pattern Regularization (APR)

Under the assumption that failures of a specified type have a common underlying cause, we propose APR and we apply it to only the positive samples in a mini-batch during the training of a CNN. APR takes the form as an augmented term $g_\theta(\mathbf{X})$ given by:

$$g_\theta(\mathbf{X}) = \frac{1}{n^2}\sum_{i=1}^{n^2} var(\mathbf{A}^\intercal\mathbf{Z}, i) \tag{6}$$

where $\mathbf{Z} \in \mathcal{R}^{N\times n^2}$ are flattened Gramian matrices calculated from the activations of the last convolutional layer, $\mathbf{A} \in \mathcal{R}^{N\times p}$ is a matrix that selects only the positive samples in $\mathbf{X}$. The variance operator is taken along each column i of the matrix product $\mathbf{A}^\intercal\mathbf{Z} \in \mathcal{R}^{p\times n^2}$.

To be concrete, let $z_i \in \mathcal{R}^{l'\times n}$ be the activation of the last convolutional layer from the i-th sample in the batch, then each row of $\mathbf{Z}$ is given by $\mathbf{Z}_i = flat(z_i^\intercal z_i) \in \mathcal{R}^{1\times n^2}$. It is easy to see that $\mathbf{Z}_i$ encodes the relationships between each filter's activation from the i-th sample, therefore the flattened Gramian matrices $\mathbf{Z}$ describe the activation pattern of each sample across a batch. Because we are imposing regularizations on only the positive samples, we construct a mask matrix $\mathbf{A}$ whose entry is defined as $\mathbf{A}_{i,j} = 1$ if $\mathbf{X}_i$ is the j-th positive sample in the batch and 0 otherwise for $i = 1, \cdots, N$ and $j = 1, \cdots, p$. Since the variance operator in Eq. (6) is applied along columns, $g_\theta(\mathbf{X})$ is a measure of activation patterns' variance among positive samples. When this term from APR is augmented to the original objective function, the training process becomes a multiple-task learning problem, and we impose a hyper-parameter λ on it to control its strength. It is worth pointing out that although we set λ fixed for each trial in our experiments, it is possible to make it adjustable (E.g. annealing λ as the training progresses). Following Eq. (1), when APR is applied together with AMS, the gradient in Eq. (5) should be updated by re-defining $\mathbf{L}$'s entry to be $\mathbf{L}_i = f_\theta(h_{S_i}(\mathbf{X}), \mathbf{y}) + \lambda g_\theta(h_{S_i}(\mathbf{X}))$.

IV. Experiments

A. Dataset Description

We demonstrate the effectiveness of our proposed methods on tasks that predict seizures in intracranial EEG recordings, the datasets are from a past Kaggle data analysis contest [18]. In the datasets, intracranial EEG was recorded from 5 dogs with naturally occurring epilepsy using an ambulatory monitoring system. EEG was sampled from electrodes at 400 Hz, and recorded voltages were referenced to the group average. In addition, data from 2 patients with epilepsy undergoing intracranial EEG monitoring to identify a region of brain that can be resected to prevent future seizures are also provided. The human datasets have varying numbers of electrodes and are sampled at 5000 Hz, with recorded voltages referenced to an electrode outside the brain. The task is to distinguish between ten minute long data clips covering an hour prior to a seizure (preictal segments), and ten minute iEEG clips of interictal activity (interictal segments). Preictal segments are provided covering one hour prior to seizure with a five minute seizure horizon, this pre-seizure horizon ensures that seizures could be predicted with enough warning to allow administration of fast-acting medications. As for negative samples, one hour sequences of interictal ten minute data segments are provided. The interictal data were chosen randomly from the full data record, with the restriction that interictal segments be as far from any seizure as can be practically achieved, to avoid contamination with preictal or postictal signals. Table. II summarizes the statistics of the datasets. As is similar to most

TABLE II
DATASET STATISTICS

	Dog 1	Dog 2	Dog 3	Dog 4	Dog 5	Patient 1	Patient 2
# Preictal Segments	24	42	72	97	30	18	18
# Interictal Segments	480	500	1440	804	450	50	42
% of Precital Segments	4.76	7.75	4.76	10.77	6.25	26.47	30.00
# Test Segments	502	1000	907	990	191	195	150
# Electrodes	16	16	16	16	15	15	24
Sampling Frequency (Hz)	400	400	400	400	400	5000	5000
Segment Length (min)	10	10	10	10	10	10	10

failure prediction tasks, positive samples (preictal segments) are greatly outnumbered by negatives (interictal segments).

B. Experimental Setup

1) Pre- and Post-processing: For stable measurements, we conducted 3-fold cross validation on each subject with each fold having the same preictal to interictal ratio. For fair comparison across different methods, we fixed the random seed such that the split of folds are exactly the same for each run. We calculated the mean and standard deviation of each segment from the training split and we average them to make the channel-wise mean and standard deviation for data normalization. Because the datasets are seriously imbalanced, we up-sample preictal segments to make the preictal to interictal ratio $1:1$ in training. At training time we randomly sample sub-sequences from the training split, and at test time we chop each validation segment with minimum overlap. After channel-wise data normalization, these sub-sequences are fed into the CNN to produce a prediction score for each sub-sequence. To merge sub-sequences' scores for the original series, we simply take their averaged value. In the case when AMS is involved, we use the simple strategy $h_S(x_{1,\cdots,lS}) = \{x'_{1,\cdots,l} \mid x'_i = x_{(i-1)S+j}\}$ where $j = 1$. And we take the expected value of scores from all scaling parameters as the final score for the original series.

2) Parameter Setup: We use the CNN shown in Fig. 1 with an input sequence length of 4000 and a mini-batch size of 256, we did *not* tune the network architecture specific to any subject. The default sample scaling parameter is $S = \{1.0\}$, and it becomes $S = \{1.0, 2.0, 3.0, 4.0\}$ when AMS is involved. Considering the 4000 input sequence length, the input sequence can include data samples in a 10 to 40 seconds time frame for dogs (400 Hz sampling frequency) but is much shorter for humans (5000 Hz sampling frequency). For this reason, we multiply S by 10 for human datasets. Adam [19] with default parameters ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-8$) is used to train the CNN and we fixed the initial learning rate at 0.001 for every subject except Patient 1 after several test runs. We set Patient 1's initial learning rate at 0.0001 because we found the training loss exploded in the middle of learning when it was 0.001. Regardless of its initial value, learning rate is exponentially decayed with a step size of 2000 iterations and a decay rate of 0.95. To prevent the CNN's parameters from getting too large, we imposed L2 regularization with a weight decay of 0.00001 for all trials. After making sure all methods could converge, we restricted the maximum number of training iterations to 15K and applied early stopping on (valid_accuracy, valid_f_score) pair with a patience of 3K iterations. In AMS trials, we update the parameter ϕ every 100 iterations by performing a one-step gradient descent. To avoid overfitting in the training, for all runs we save only the CNN snapshot with the best (valid_accuracy, valid_f_score) scores and we report the performance of this model in the following sections. In our experiments, λ the weight for APR is the only hyper-parameter we tuned. Specifically, we searched a grid in the space $[1e-5, 1e-1]$ with five log-linearly spaced values. Therefore we ran in total 12 trials (1 for the baseline, 1 for AMS, 5 for APR and 5 for the combination of the two proposals), and each trial is a 3-fold cross validation.

3) Evaluation Metrics: We report the Kaggle submission scores and ranks for each method. To submit to Kaggle, we need to combine predictions on each subject to generate a submission file, and we consider two combination strategies. In its simplest form, for each method we choose the predicted results from the learnt models in the trial that achieved the highest mean accuracy and F-score on the validation sets (valid_accuracy, valid_f_score), cross validation scores with ties are broken by smaller standard errors. We then average these models' predicted scores and we call this method mean ensemble. In a second way, we let each fold's result represent as the trial's result independently and simply pick for each subject the fold run that generated the highest (valid_accuracy, valid_f_score) scores. If there are ties, we generate a submission file with each of them and we report the best. We call this method max ensemble.

For all metrics, we report the performance of 4 methods:

- Baseline – CNN accepting sub-sequence of fixed length trained with the traditional binary cross-entropy loss
- AMS – CNN trained with adaptive multi-scale sampling
- APR – CNN trained with activation pattern regularization
- AMS + APR – CNN trained with both AMS and APR

C. Results and Discussion

1) Kaggle Submissions: We report the scores and ranks we achieved when submitting predictions on the provided test set to Kaggle. In this Kaggle contest, AUROC was set to be the evaluation metric and 504 teams in total made submissions. Because this contest is over, we are able to receive a (Public, Private) scores pair from each submission to Kaggle. According to the website, public score reflects AUROC resulted from approximately 40% of the entire test set, and it serves as a feedback to contesters as to how well

TABLE III
KAGGLE SUBMISSION (SCORES ARE AUROCS)

	Private Score	Public Score	Rank
Baseline (mean)	0.65254	0.70972	131st
APR (mean)	0.66489	0.72716	102nd
AMS (mean)	0.69183	0.75419	71st
AMS + APR (mean)	**0.70535**	**0.77922**	**58th**
Baseline (max)	0.68635	0.75125	77th
APR (max)	0.68656	0.77887	77th
AMS (max)	0.70489	0.77643	58th
AMS + APR (max)	**0.77619**	**0.80976**	**14th**

their current model is doing. Private score is calculated from the remaining 60% of the test data and is the metric based on which rank is determined. As we described in Sec. IV-B3, we report two groups of scores, one for the mean ensemble method and the other for the max ensemble. We summarize the scores and ranks in Table. III, for APR and AMS + APR, the best scores from the grid search on λ is reported.

What surprised us a lot is that even the baseline method could achieve high ranks (top 26% for mean ensemble and top 15% for max ensemble), considering we barely did anything for feature engineering! This is consistent with the recent reports on successful applications of deep learning to general datasets besides images and audios. Secondly we notice the max ensemble gives higher scores than mean ensemble, but the trend in either group is consistent. To be concrete, while we observe slight degradation in APR when comparing cross validation results to that of the baseline's, it gives slight improvement in Kaggle submissions. A possible explanation is the size of the validation set is much smaller than the test size and thus validation scores did not well reflect the model's generalizability. We notice obvious improvements on both public and private scores when applying AMS for the same reason we discussed in the previous section. Compared to the baseline, the combination of AMS and APR gives an average relative improvement of near 10% (8.09% and 9.79% for mean ensemble, 13.09% and 7.79% for max ensemble), and sent us to as high as the 14-th place on the leaderboard.

It is responsible for us to point out that even the scores from AMS + APR are far from the winner of the contest (AUROC@Private=0.83993, AUROC@Public=0.90316). However, we did not use the entire or a larger proportion of the datasets for training, we did not tune our network's architecture or apply advanced setups like batch normalization or dropout, we did not even fine tune other possible hyper-parameters or try sophisticated ensemble methods as is common in data analysis contests. Rather, our goal is to show the problem in traditional CNN training scheme and to demonstrate the effectiveness of our proposals instead of beating the state-of-the-arts. We believe our current achievement in Table. III well served our purpose.

2) Experimental Analysis of AMS: After some macro view over the experimental results, let us zoom in to see the effects of each proposed method. Fig. 2 gives the distribution of scaling parameters for each subject after AMS application. As

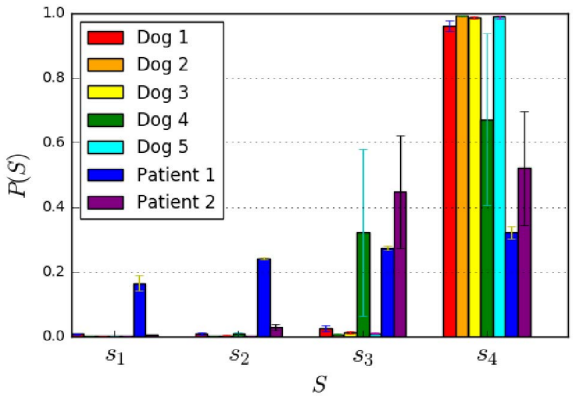

Fig. 2. Distribution of Scaling Parameters from 3-fold Cross Validation AMS.

we can see, our initial guess of input length 4000 (equivalent to 10sec/8sec of time frame for Dogs/Humans) is not optimal and AMS learnt to find the combinations of sub-sequence lengths better suited for the task. Although AMS puts most weight on the largest scaling parameter for dogs 1, 2, 3 and 5, it considers combinations of sub-sequence lengths for dog 4 and both human patients. This suggests AMS does not always prefer longer sequences but is indeed looking for patterns residing in sub-sequences of different lengths that is optimal for the task. Given abundant time, one can try a wide range of scaling parameters and based on the results from AMS, it is possible for one to find a theoretical explanation specified to the underlying domain about why the selected combination of lengths makes sense.

We plot the evolution of training loss, validation accuracy and validation f-score for both the baseline and AMS in Fig. 3. Depending on the line the Y-axes can stand for binary cross-entropy loss, accuracy or f-score, the X-axes are training iterations in hundred. For almost all subjects and on any of the three criteria, we find AMS learns faster than the baseline. One may consider the faster training loss convergence is due to that in these trials AMS kept selecting longer sequences and thus led to smaller sample spaces (the longer the sub-sequences, the less samples for training), and a large network can easily fit to a smaller sample space, leading to faster convergence. This might be partially responsible, but it is also well known that with the same network capacity and smaller sample spaces, a flexible model like CNN can easily overfit the training dataset. However, we see strong faster rising accuracy and f-score lines, and do not observe any sign of overfitting in Fig. 3. We hence argue the faster learning speed (in terms of both training loss and validation scores) is indeed due to the involvement of AMS.

3) Experimental Analysis of APR: Though not significant, we also observed improvement from the baseline when APR is applied in Table. III, we shall look into the learnt network and give an insight on when APR might deliver better performance. To analyze the difference between the baseline and APR, we select a random batch of validation data for both models

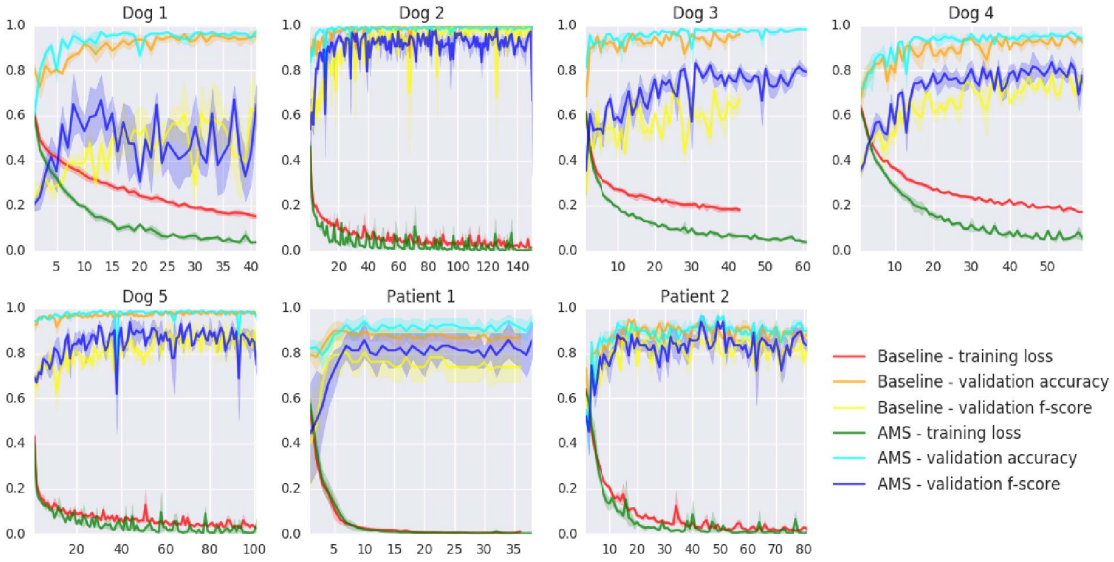

Fig. 3. Training Statistics for the Baseline and AMS (depending on the line the Y-axes can stand for binary cross-entropy loss, accuracy or f-score, the X-axes are training iterations in hundred; shorter lines are due to early stopping).

and we check the differences of their activation patterns. Concretely speaking, from the models we learnt in cross validation, we pick a baseline model and a series of APR models with different λ settings from the same fold (such that they share the same validation dataset). We input the validation batch of preictal segments into the baseline model and the series of APR models, and we record the Gramian matrices $\mathbf{Z}$ that encodes their activation patterns. For each model, we calculated the variances of each entry of the Gramian matrices and analyze the difference $var(\mathbf{Z}^{(\mathbf{baseline})}) - var(\mathbf{Z}^{(\mathbf{APR})})$ where the variance is taken on each entry of the matrices. Fig. 4 gives an image of what we described. We have scaled the values to have unit standard deviation and have labelled the 0-level in each diagram for visual convenience. In order to find patterns, we have put the settings that have higher or equal (valid_accuracy, valid_f_score) scores than the baseline in green and leave the rest in black.

Although with a few exceptions, we find from this figure that winning settings tend to have more positive $var(\mathbf{Z}^{(\mathbf{baseline})}) - var(\mathbf{Z}^{(\mathbf{APR})})$ entries (upward pointing spikes), meaning lower activation pattern variances compared to the baseline. This is due to the augmented term to the total objective function introduced by APR, and thus we can attribute the performance improvement to APR.

V. Conclusion and Future Works

In this paper, we discussed the two previously unexplored problems in failure prediction using CNN: 1) determining without domain knowledge optimal sub-sequence length for CNN that provides sufficient data augmentation and at the same time avoids serious multiple-instance learning problems; 2) helping CNN to concentrate on learning the common representations that capture the characteristics of the underlying cause. We proposed AMS to solve the first problem which automatically searches for a combination of sub-sequence lengths by learning a set of parameters that controls the sub-sequences lengths jointly with the learning of CNN parameters. From experimental results, we also find it learns faster and generalizes better than the vanilla CNN training scheme. To address the second issue, we propose APR that shifts the CNN's attention to positive samples by minimizing the variances of the Gramian matrices' entries formed from the last convolutional layer's activations of only positive samples. Our experiments on real-world seizure prediction EEG datasets demonstrated the significant improvement from both AMS and APR.

APR is suggested to constrain the variance of activation patterns of only positive samples. But such a setting is not restricted to binary classification problems where samples from only one class presents common pattern. It is applicable to multiple-class classification problems if similar data characteristics can be observed. Concretely, APR can be applied on samples from each class and we are interested in its performance in applications such as environmental voice recognition using CNN.

References

[1] Felix Salfner, Maren Lenk, and Miroslaw Malek. A survey of online failure prediction methods. *ACM Comput. Surv.*, 42(3):10:1–10:42, March 2010.

[2] Joseph F. Murray, Gordon F. Hughes, and Kenneth Kreutz-Delgado. Machine learning methods for predicting failures in hard drives: A multiple-instance application. *J. Mach. Learn. Res.*, 6:783–816, December 2005.

[3] Y. Liu, K. T. Yao, S. Liu, C. S. Raghavendra, O. Balogun, and L. Olabinjo. Semi-supervised failure prediction for oil production wells. In *2011 IEEE 11th International Conference on Data Mining Workshops*, pages 434–441, Dec 2011.

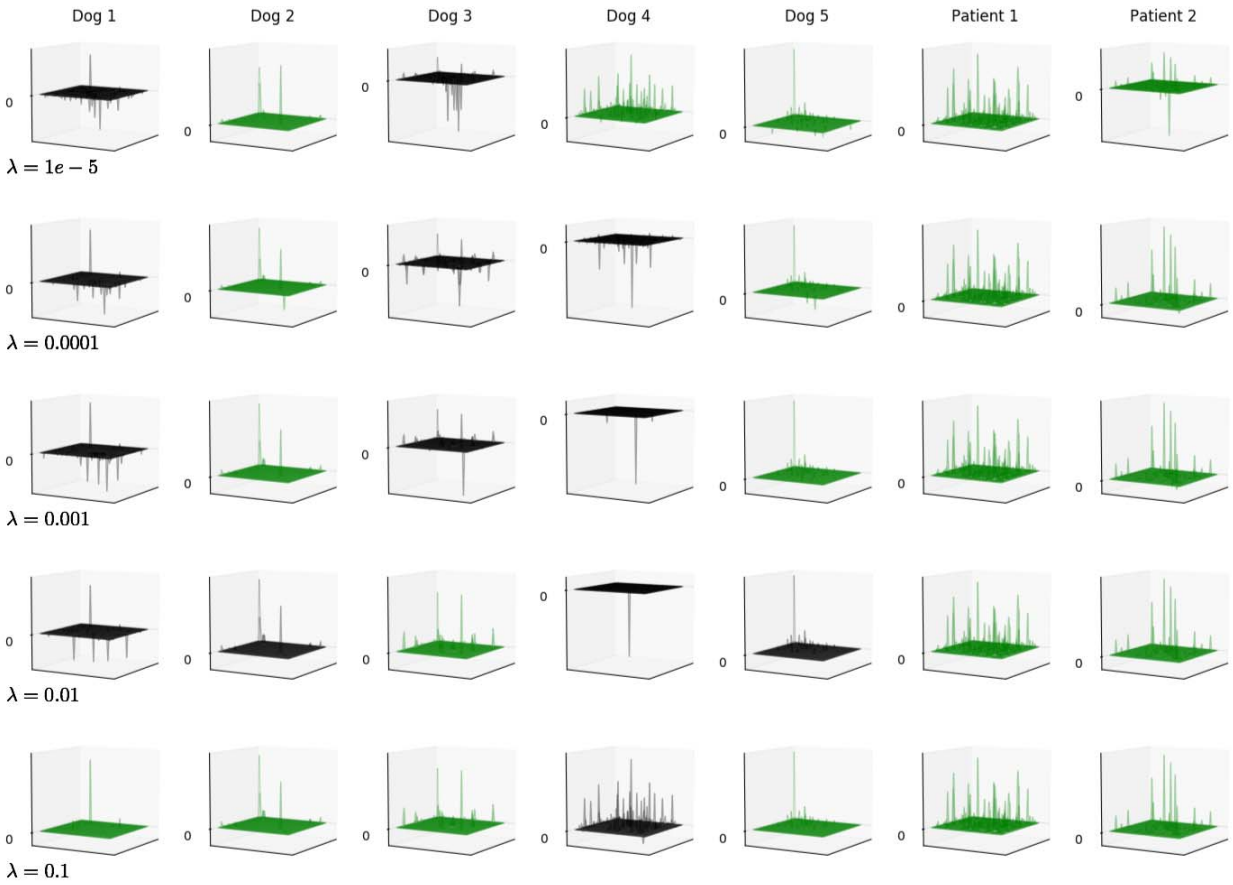

Fig. 4. Difference of Positive Samples' Activations Variances (settings with higher or equal (valid_accuracy, valid_f_score) scores are in green).

[4] T. Chalermarrewong, T. Achalakul, and S. C. W. See. Failure prediction of data centers using time series and fault tree analysis. In *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, pages 794–799, Dec 2012.

[5] Lexiang Ye and Eamonn Keogh. Time series shapelets: A new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 947–956, New York, NY, USA, 2009. ACM.

[6] Jason Lines, Luke M. Davis, Jon Hills, and Anthony Bagnall. A shapelet transform for time series classification. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 289–297, New York, NY, USA, 2012. ACM.

[7] Bing Hu, Yanping Chen, and Eamonn Keogh. *Time Series Classification under More Realistic Assumptions*, pages 578–586. 2013.

[8] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping. *ACM Trans. Knowl. Discov. Data*, 7(3):10:1–10:31, September 2013.

[9] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 369–376, New York, NY, USA, 2006. ACM.

[10] Zhiguang Wang and Tim Oates. Encoding time series as images for visual inspection and classification using tiled convolutional neural networks. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI, 2015.

[11] Tim Oates Zhiguang wang. Imaging time-series to improve classification and imputation. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3939–3945. AAAI, 2015.

[12] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J. Leon Zhao. Time series classification using multi-channels deep convolutional neural networks. In *Web-Age Information Management - 15th International Conference, WAIM 2014, Macau, China, June 16-18, 2014. Proceedings*, pages 298–310, 2014.

[13] Jian Bo Yang, Minh Nhut Nguyen, Phyo Phyo San, Xiao Li Li, and Shonali Krishnaswamy. Deep convolutional neural networks on multichannel time series for human activity recognition. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 3995–4001. AAAI Press, 2015.

[14] Y. Tang, J. Xu, K. Matsumoto, and C. Ono. Sequence-to-sequence model with attention for time series classification. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 503–510, Dec 2016.

[15] Ernst Niedermeyer and Fernando L. da Silva. *Electroencephalography: Basic Principles, Clinical Applications, and Related Fields*. Lippincott Williams & Wilkins, 5th edition, November 2004.

[16] Haibo He and Edwardo A. Garcia. Learning from imbalanced data. *IEEE Trans. on Knowl. and Data Eng.*, 21(9):1263–1284, September 2009.

[17] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In Johannes Frnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814. Omnipress, 2010.

[18] American epilepsy society seizure prediction challenge. https://www.kaggle.com/c/seizure-prediction.

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.