Forecasting Oil temperature in Transformer using Adative Filtering

Shantanu Kapoor,Electrical and Computer Engineering, University of Florida, Gainesville, U.S.A

***Abstract*—- With the ongoing rapid increase in global energy prices, it is becoming increasingly critical to accurately predict future electric consumption. One potential method for predicting future load is through the use of oil, a key component of transformers. Due to its strong correlation properties, oil temperature can be utilized to provide insight into future load prediction. In this paper, we not only compare linear and non-linear adaptive filters trained on oil temperature from the ETTm2 time series dataset, but also assess their ability to accurately predict the next time step. Through the use of both Mean Square Error (MSE) and Maximum Correlation Criterion (MCC) cost functions, we are able to train these models to predict the next time step with greater accuracy.**

Additionally, we analyze the models capacity to predict the number of future time steps with 0.2 std of accuracy. We discovered that while linear filtering can provide a strong prediction of the next time step, non-linear filter KLMS outperforms the linear filter for better trajectory generation and multi-step prediction. By taking advantage of these advanced prediction tools, we can more accurately forecast electric consumption, and distribute electricity wisely.

***Index Terms*—oil temperature, adaptive filters, linear filtering, non-linear filtering, mean square error , maximum correlation criterion, multi-time step prediction.**

I. Introduction

Electricity was a result of the second industrial revolution, and since then, its consumption has been increasing rapidly. All major occupations, from industries to households, rely on electricity. To handle this constantly increasing demand with appropriate distribution, effective grid management is essential. One imperative aspect of grid management is total load distribution and prediction. This prediction can be utilized to reduce expenses associated to over- or under-load delivery, which can be particularly useful during sensitive global energy price fluctuations.

Various statistical methods have been used to tackle the challenge of load forecasting. These methods typically involve predicting the load trend based on past historical values, or using correlation with other features such as different metrics of loads or environmental factors associated with the load, such as oil temperature. The forecasts are made at certain intervals, which depend on the problem and can be divided into short-term, medium-term, and long-term load forecasting.[1]

Currently, Deep learning methodologies utilizing LSTM or Transformers are leading the current research on forecasting models. They have shown exceptional results in financial modeling, speech detection, language prediction, and more. However, these methods are data-hungry and require significant computational resources which cannot be implemented cost-effectively on small IoT devices.To address this resource problem, we can switch to traditional methods of adaptive linear and non-linear filtering. While these methods may not be as cutting-edge, they are still perfectly viable options. They are also relatively simple to implement and will be used for on-demand short-term oil temperature forecasting(ranging from 1 minute to 1 hour), as present in our case.

The following subsections briefly discuss and analyze the adaptive filters: Section 2 discusses the adaptive linear filter LMS, and adaptive non-linear kernel filters KLMS and QKLMS. It also covers the methodologies and adaptive algorithms used to train these models, such as MSE and MCC. Section 3 discusses the dataset used for forecasting. Section 4 presents the experiment and results, analyzing the effect of hyperparameters like step size, kernel size, and quantization based on MSE. It also compares multi-step prediction with time-ahead forecasting at 10 and 50 samples for both algorithms. Finally, Section 5 presents the conclusion.

II. Adaptive Filters

Adaptive filters are systems that use optimization algorithms to adjust their parameters. They consist of a digital filter with adjustable weight coefficients, which are modified according to a loss function. The filter produces an output that is compared to the expected output using a suitable distance measure. This measure then goes through a loss function that assesses the filter's performance. The evaluation is then processed through an adaptive algorithm that adjusts the filter weights until the loss measure is minimized. In the following subsections, we will discuss linear and non-linear adaptive algorithms in the context of forecasting methods.

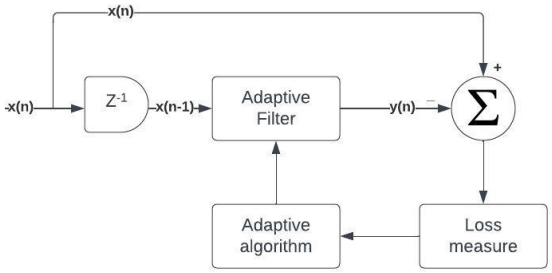

Fig. 1. linear adaptive filter

A. Adaptive linear filtering using LMS

From Fig1, we can see that the input to the linear filter is a vector X[n-1] containing past values up to [n-m] time

steps before x[n-1], where m is the total number of delay samples. This input vector is passed through the adaptive filter weight vector w to predict the desired time step x[n]. The difference between the desired output and predicted e(n) is passed through different error measures that define the equation of the adaptive algorithms. For MSE loss function, equations are as follows:

$$e(n) = x(n) - X(n-1)^T W_{n-1} \tag{1}$$

$$W_n = W_{n-1} + 2\eta e(n-1)X(n-1) \tag{2}$$

Other error measure that has been used is MCC. This is a Information theoretical similarity measure based on Reyni's entorpy that maps the information into RKHS space. The adaptive equations for MCC are as follows:

$$G_\sigma(e(n)) = \frac{1}{\sqrt{2\pi}\sigma}\frac{e^{-e^2(n)}}{2\sigma^2} \tag{3}$$

$$W_n = W_{n-1} + \frac{\eta}{\sigma^2}G_\sigma(e(n))e(n-1)X(n-1) \tag{4}$$

Here σ is the kernel size that is used for attenuating outliers. The η denotes the step size of the algorithm. Here the G_σ function allows weighted gradient descent for lms adaptation.

B. Adaptive Non-linear filtering using KLMS and QKLMS

In non-linear filtering, the input features are passed through a fixed basis of non-linear functions that map them into function space. From there, these input vectors are linearly weighted to produce the desired output. Usually, these mappings can span infinite dimensions. Instead of using these infinitely mapped dimension functions, we use kernel methods. In our case, we use Gaussian kernels k(X[j],.), which are positive definite and universal, spanning infinite dimensions. These kernels are evaluated on aforementioned input feature vector X[n-1]. For MSE the desired equations are as follows:

$$k = \frac{e^{-\|X_j - X\|_2}}{h^2} \tag{5}$$

$$f_{n-1} = \eta\sum_{j=1}^{n-1} e(j)k(X(j),.) \tag{6}$$

$$f_{n-1}(X[n-1]) = \eta\sum_{j=1}^{n-1} e(j)k(X(j),X[n-1]) \tag{7}$$

$$e(n) = x[n] - f_{n-1}(X[n-1]) \tag{8}$$

$$f_n = f_{n-1} + \eta e(n)k(X[n-1],.) \tag{9}$$

For MCC measure we change equation 6 and 7 by apply additional weighted adaptation with kernel G_σ as follows:

$$f_{n-1} = \frac{\eta}{\sigma^2}\sum_{j=1}^{n-1} G_\sigma(e(j)e(j)k(X(j),.) \tag{10}$$

$$f_{n-1}(X[n-1]) = \frac{\eta}{\sigma^2}\sum_{j=1}^{n-1} G_\sigma(e(j)e(j)k(X(j),X[n-1]) \tag{11}$$

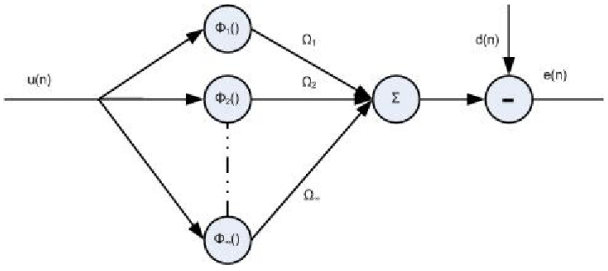

Fig. 2. linear adaptive filter

Here, the variable h denotes the kernel size for kernel filtering, η denotes the step size, and σ denotes the kernel size for the MCC measure. These equations show that the network architecture grows with an increasing number of training samples, with greater network complexity and better capability. However, sometimes the RKHS space is compact and doesn't require an increasing number of network sizes which increases time complexity. To solve this problem, QKLMS is used instead. In QKLMS, the input space is quantized and each input is assigned to its center C(i). Its coefficients a(i) are updated with each input vector u(i). The algorithm to apply this method online are as follows:

Algorithm 1 QKLMS Algorithm

Initalization: step size η , kernel width h, quantization threshold ϵ , center dictionary C(1) = u(1) and coefficient vector a(1) = [ηd(1)]

while {u(1),d(1)} is available **do**

$$e(i) = d(i) - \sum_{j=1} K(i-1)a_j k_h(u(i), C_j(i-1)$$

$$dis(u(i), C(i-1)) = \min_{i\le j\le k(i-1)} \|u(i) - C_j(i-1)\|$$

$$j^* = \min_{1\le j\le K(i-1)} \|u(i) - C_j(i-1)\|$$

if $dis(u(i), C(i-1) \le \epsilon$ **then**

$$C(i) = C(i-1), a_{j^*}(i) = a_{j^*}(i) + \eta e(i)$$

else

$$C(i) = C(i-1), u(i), a(i) = [a[i-1], \eta e(i)$$

III. Dataset Used

These adaptive models are trained using EETm2 time series. [reference]. This is real-world data based on an electric transformer station, and it has been used for studying long-sequence time series forecasting. The data-set contains two years of data sampled at an hourly frequency, resulting in approximately 17K samples. These 17K samples were divided into sets of 10K,1K and 6K sample for training, validation and testing respectively. Each data-point contains six external load features and the "oil temperature". In our case, we will use the oil temperature for forecasting. The oil temperature exhibits

short-term continuity but has a long-term trend that is visible using the auto-correlation function. It remains consistent for a day or two. For our experiment, we neglect de-trending the signal due to complex post-processing, but we normalize the signal as it has shown to improve prediction.

$$x(n) = \frac{x(n) - x_{mean}}{x_{std}} \tag{12}$$

IV. Experiment and Results

This section discusses obtaining hyper-parameters like step size, kernel size, and quantization threshold for filtering models. We also evaluate the effect of two losses, MSE and MCC, on filter training and compare linear and non-linear filtering performance. Additionally, we discuss these models' capability in multi-step predictions. The MSE was used for measuring their evaluations

A. Estimating Hyperparameter for LMS, KLMS and QKLMS

To train the adaptive models, we divided the training dataset into 11 blocks, each containing 1K continuous samples. For each particular combination of hyperparameters, we trained the model on 10 blocks for training and 1 block for validation. We evaluated the performance of the model using the average validation score for leave-one-out cross-validation. We determined the range of values for each hyperparameter by taking each one in order of 10 initially. After finding the best set of hyperparameters in this set, we trained the model on the surrounding set of values to find the best hyperparameter.

The best η for lms with MSE was 0.003 meanwhile for MCC was 0.0007 with kernel size σ 0.25. The step size for MCC was small as the step size was scaled by the sigma hyper parameter of MCC. The weight tracks of LMS with both mse and MCC are given below. Both got the weight tracks converged to almost same value which also suggest the robustness of this model. We can also see that weight tracks 2&7,3&6 and 4&5 converged to same value which might be due to some periodicity between these time step values. In our test set the lms with MCC outperformed MSE.

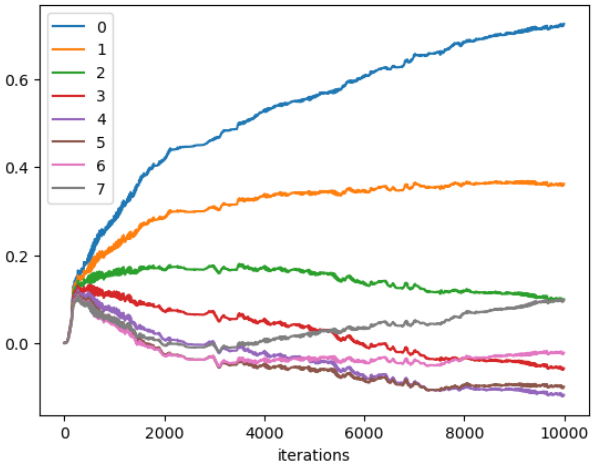

Fig. 3. change in weights with time step for lms with mse

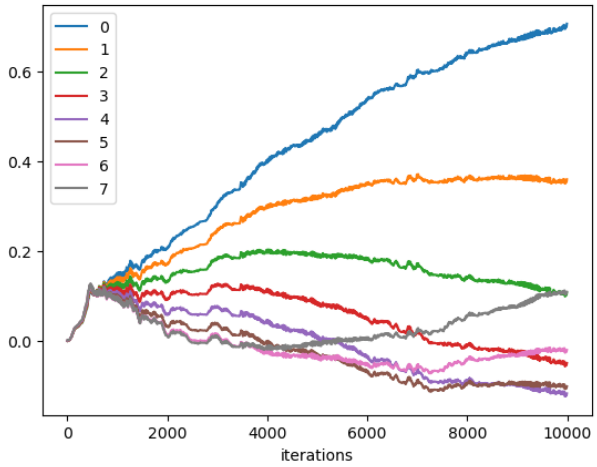

Fig. 4. change in weights with time step for lms with mcc

For KLMS with MSE, we found that the best step size η was 0.109 with kernel size h 0.922. For training with MCC, we used σ 0.8 with kernel size h 1.4 and step size 0.02. To train KLMS with MCC, we performed a sweep around all three parameters and drew heatmaps for each kernel size with varying step size and MCC kernel size. We found that for every kernel size, σ 0.8 showed the least value, so we fixed it and performed a sweep on kernel size h and step size η. For QKLMS best kernel size h was 0.4 and step size η 0.9 with threshold ϵ 0.2. and for MCC σ is 2.

B. Evaluation of the Adaptive Models

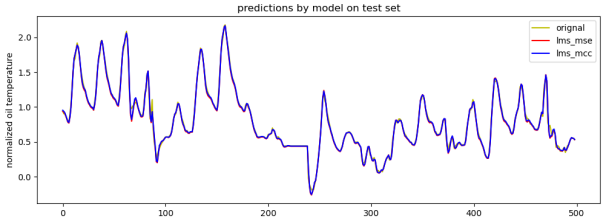

Fig. 5. test set predictions for lms

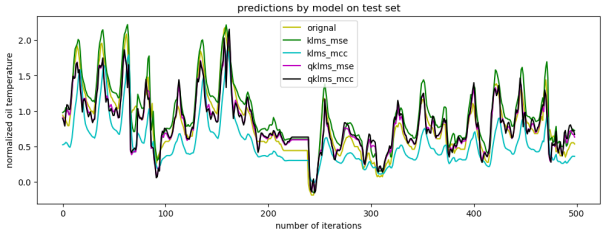

Fig. 6. test set predictions for klms

In our evaluation of the models mentioned above, we compared their average test scores and convergence on the training set. Our results showed that LMS performed better overall than KLMS/QKLMS. Specifically, LMS with MCC

had the best overall score on the train, validation, and test sets after training. Although in the convergence graph LMS_MCC is above KLMS, this is because major learning did not happen until around 500-600 iterations, during which initial large error values were encountered and reflected in the average MSE. However, LMS_MCC exhibited tighter model fitting with the steepest slope. Even for KLMS, learning happened after 900 iterations. We also observed that MSE converges and trains faster than MCC in all of the adaptive filtering methods

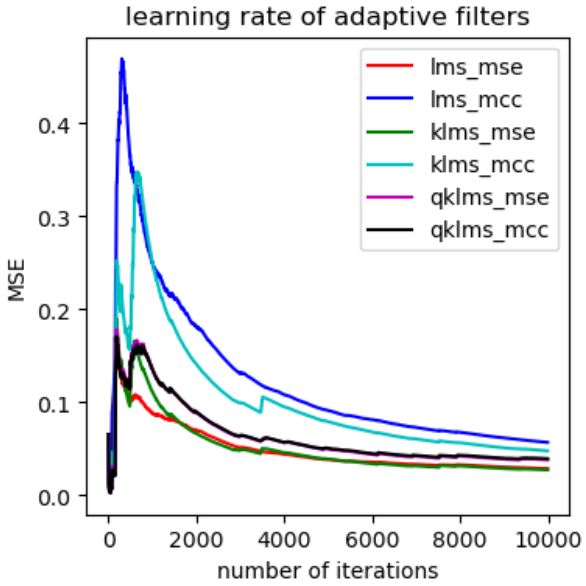

Fig. 7. convergence plots for LMS and KLMS model

TABLE I
ADAPTIVE FILTER MSE TABEL

Adaptive Models	Train		Validation		Test	
	MSE	MCC	MSE	MCC	MSE	MCC
LMS	0.00124	0.0010	0.00095	0.0080	0.00093	0.00069
KLMS	0.0140	0.070	0.0092	0.036	0.056	0.083
QKLMS	0.040	0.052	0.0153	0.032	0.082	0.104

KLMS performs better than its counterpart QKLMS due to its network complexity, as shown in Table I. However, computation times increase exponentially with an increase in complexity. To mitigate this issue, we can look at the tradeoff between accuracy and the number of samples to choose appropriate complexity. We found that MSE accuracy plateaus at 2000 samples in Fig 8a) and the same number of samples correspond to threshold ϵ 0.2 in Fig 8b), which we discovered while spanning the hyperparameter space

C. Comparison between MSE and MCC training

Apart from the relative difference between MSE and MCC in error convergence shown in Fig 7, we can compare these loss functions using the test set error histogram. Furthermore, it can also be used to show the effect of effective kernel size σ in MCC adaptive algorithms for better performance.

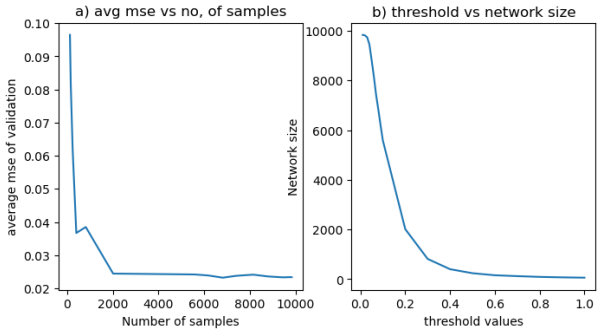

Fig. 8. Effect of QKLMS network size on Avg MSE

According to our evaluation of LMS linear filter, both MSE and MCC loss are highly concentrated towards zero error, as shown in Fig 9a). Meanwhile, for KLMS, MCC has a much wider spread than MSE due to its poor performance. This could be due to the inability to find the right hyperparameter set for KLMS-MCC. The effects of σ can be properly seen in Fig 9b) and 10b), respectively. With an increase in kernel size from 1 to 2.5 in Fig 9b), we can see that the error concentrates significantly; however, it is centered away from zero, which could be due to overfitting of the model and discarding potential samples as outliers. For an appropriate kernel size, the standard deviation should be at a minimum and mean should be approximately 0. This can be seen for σ 0.25 for LMS and 0.8 for KLMS. Although KLMS has more spread, it offers a more general solution for the model.

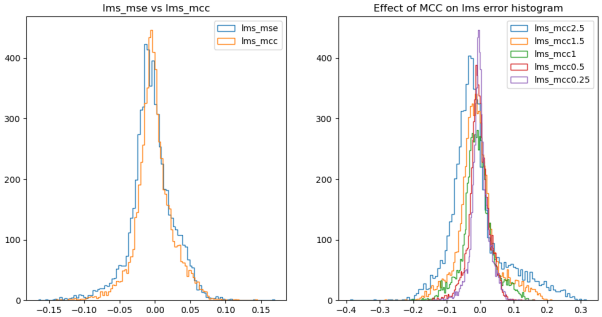

Fig. 9. Effect of MCC on error histogram for LMS

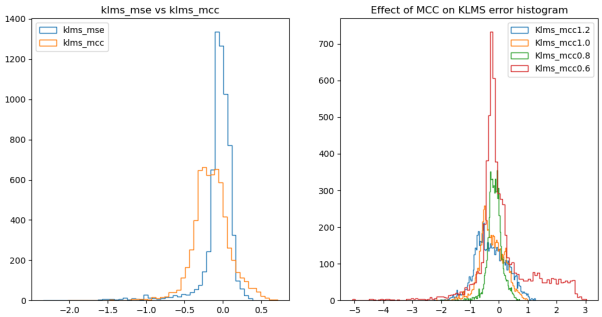

Fig. 10. Effect of MCC on error histogram for KLMS

D. Multi-step prediction analysis

Multi-step prediction can be used as an effective evaluation metric to determine if an algorithm is overfitting or generalizing the generating distribution. In our case, we predicted the next 10 and 50 samples using past values generated by the algorithm itself. From the weight tracks in Fig 3, it can be seen that LMS gives huge importance to n-1 and n-2 weights. Moreover, most of the weights are positive. Hence, the positive or negative errors accumulated either increase or decrease the value, thus causing large MSE or MCC errors. Meanwhile, for KLMS, the errors are low. Since the KLMS has a more complex network it has better chances of genralizing the input distribution.

TABLE II
SAMPLE AHEAD PREDICTION ON TRAIN AND TEST DATASET USING ADAPTIVE FILTERS

Adaptive Models	Samples Ahead on training set					
	1		10		50	
	MSE	MCC	MSE	MCC	MSE	MCC
LMS	0.0012	0.0010	3.0035	3.2694	1.6787	2.6926
KLMS	0.0140	0.0706	0.3614	0.4459	0.6866	0.9194
Adaptive Models	Samples Ahead on test set					
	1		10		50	
	MSE	MCC	MSE	MCC	MSE	MCC
LMS	0.00094	0.00069	4.0333	4.5701	2.4122	3.8539
KLMS	0.056	0.083	0.4505	0.5225	0.6358	0.9855

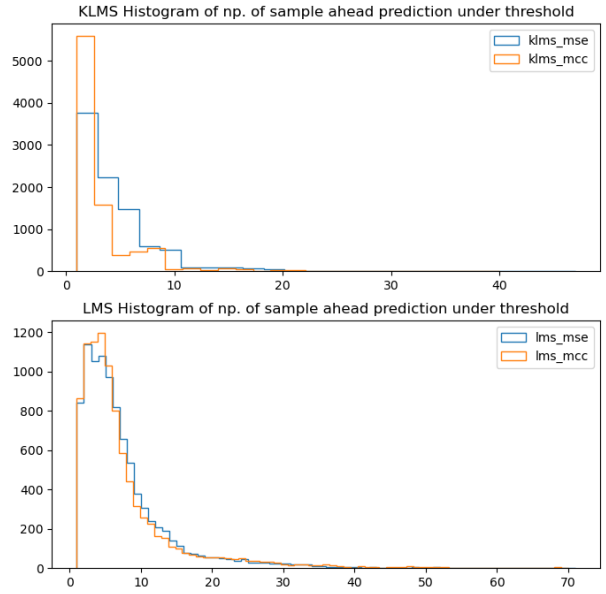

Fig. 11. Average sample ahead prediction for KLMS and LMS below 0.2 error threshold

For further rolling time series analysis, we generated the trajectory of the adaptive algorithms until 0.2 * standard deviation $\approx$ 0.2 of the dataset. To evaluate this , across the whole dataset , we took multiple random sequence of 8 contiguous samples and tried to predict the future samples until the error surpassed threshold. Later, we plotted their histogram to compare their performance. It was shown the lms averaged at 7 samples while the KLMS averaged around 3 samples. Furthermore, we trained algorithms to predict 10 time steps ahead by using a 10 time step delay. However, every algorithm (LMS and KLMS) sampled values that lagged behind by the desired values by 17 for the validation set. Thus, their analysis of next 10 or 50 samples was dropped. Although KLMS was able to fit the training data, it showed poor results on the validation data.

V. Conclusion

In conclusion, both linear and non-linear filters can be used for short-term next-step forecasting for oil temperature. In this case, LMS outperformed KLMS. Moreover, LMS-MCC achieved significantly better results compared to the rest of the models. The effect of MCC kernel size on training of the algorithm was also observed. While the models succeeded in one-time-step-ahead predictions, a robust model should be able to predict a large number of multiple-time steps with high accuracy for better generalization, where these models underperformed. To tackle this issue, different methods of training or non-linear algorithms must be addressed.