Interpretable Multi-Task Learning for Product Quality Prediction with Attention Mechanism

Cheng-Han Yeh
Department of Computer Science,
National Chiao Tung University,
Hsinchu, Taiwan
jerryyeh.cs05g@nctu.edu.tw

Yao-Chung Fan
Department of Computer Science,
National Chung Hsing University,
Taichung, Taiwan
yfan@nchu.edu.tw

Wen-Chih Peng
Department of Computer Science,
National Chiao Tung University,
Hsinchu, Taiwan
wcpeng@g2.nctu.edu.tw

***Abstract*—In this paper, we investigate the problem of mining multivariate time series data generated from sensors mounted on manufacturing stations for early product quality prediction. In addition to accurate quality prediction, another crucial requirement for industrial production scenarios is model interpretability, i.e., to understand the significance of an individual time series with respect to the final quality. Aiming at the goals, this paper proposes a multi-task learning model with an encoder-decoder architecture augmented by the matrix factorization technique and the attention mechanism. Our model design brings two major advantages. First, by jointly considering the input multivariate time series reconstruction task and the quality prediction in a multi-task learning model, the performance of the quality prediction task is boosted. Second, by incorporating the matrix factorization technique, we enable the proposed model to pay/learn attentions on the component of the multivariate time series rather than on the time axis. With the attention on components, the correlation between a sensor reading and a final quality measure can be quantized to improve the model interpretability. Comprehensive performance evaluation on real data sets is conducted. The experimental results validate that strengths of the proposed model on quality prediction and model interpretability.**

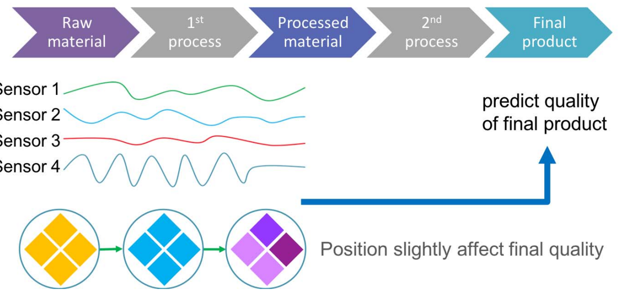


Fig. 1: A real photoelectric component production scenario where the entire production process takes a week to complete and there are two stages for production. The first stage takes a few hours while the second stage takes the rest of the production time. According to site work experiences, The first stage is crucial to the final product quality. Such observation brings the possibility of the early stage product quality prediction.

I. Introduction

Predicting product quality in the early stage is a pivotal task in industrial production scenarios. In an industrial production process, it often takes various manufacturing stages for production, and each stage has various impacts on the quality of a final product. Therefore, predicting product quality in an early stage is valuable to early damage control, promptly malfunction notification, and manufacturing cost reduction.

In this paper, we investigate the problem of mining multivariate time series data generated from sensors mounted on manufacturing stations for early product quality prediction. Specifically, we consider a real photoelectric component production scenario where the entire production process takes a week to complete and there are two stages for the component production, as illustrated in Figure 1.

The first stage takes a few hours to produce a set of wafers and the second stage takes the rest of the production time to further process the wafers to the final photoelectric component. According to site work experiences, the first stage dominates the final quality of the produced photoelectric components. Such indication motivates the possibility of early stage quality prediction if the data at the first stage can be properly analyzed. The available data at the first stage is the sensor readings of a manufacturing station. The sensors record the temperature, humidity, rotation, etc., over time, and therefore a multivariate time series data is formed at this stage. The multivariate time series data are highly correlated with the quality of the products, so discovering the correlation between the time series and the final quality should be highlighted. In addition to the sensor time series, also according to the working experiences, the position information of a wafer in a manufacturing station has also impacts on the final quality. Namely, the data from the first stage is a multivariate time series and one position code (the position information of a wafer), as illustrated in Figure 1. One goal of this work is to make use of these data to predict the final product quality for early-stage production quality prediction.

2375-026X/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDE.2019.00207

In addition to accurate quality prediction, another critical requirement for the industrial production scenario is model interpretability, i.e., to understand the importance of an individual sensor reading with respect to a final quality measure of the produced photoelectric components. According to company employees, the engineer can leverage such information to adjust the setting of the station to improve the wafer quality and use prediction results to decide if they shall terminate the process immediately.

Aiming at such targets, in this paper, we propose a multi-task learning model with an encoder-decoder architecture augmented by the matrix factorization technique. Our model design brings two major advantages. First, by jointly considering the input multivariate time series reconstruction task and the quality prediction in a multi-task learning model, the performance of the quality prediction task is boosted. Second, by incorporating the matrix factorization technique, we enable the proposed model to pay/learn attentions on the component of the multivariate time series rather than on the time axis. With the attention on components, the correlation between a sensor's reading (e.g. temperature) and a final quality measure (e.g. brightness of the final product) can be quantized to improve the model interpretability.

Specifically, our model is composed of an encoder and three decoders. The encoder consumes the input multivariate time series and transforms it into a learned latent representation. And three decoders share the latent representation as an input for their own tasks. One decoder is designed to generate an attention matrix to weight the correlation between a sensor's reading and a quality measure, while the other two decoders are designed to generate two low-rank representations of the input multivariate time series by incorporating matrix factorization techniques. The encoder and decoders are jointly trained and optimized to exploit commonalities and differences to guarantee the generation of low-rank representations. The outcomes of the proposed model are three folds. The first one is an attention matrix provides correlation information between a time series and a quality measure, and the second one is an accurate prediction model for predicting product qualities based on the input time series. And, the third one is an effective representation for the input time series and the position code, which extract important features as a latent vector. In summary, the proposed model produces three components: (1) an attention matrix providing the correlation information between individual sensor with respect to the product quality, (2) a model for predicting product quality based on the input multivariate time series, and (3) a compact representation which preserves the latent information of the raw input time series.

The contributions of this paper are summarized as follows:

- An analytical neural network design with a multi-task learning model utilizing attention mechanism and matrix factorization to provide interpretable and effective attentions between sensors and product qualities.
- A design of incorporating matrix factorization techniques to jointly guarantee the generation of low-rank representations of the input multivariate time series. Also, a serial of attention mechanism is designed to help the proposed model effectively learn attentions.
- The resultant latent representation of the input multivariate time series and the position code of the proposed model are validated to be able to improve the performance of the existing classification models (e.g., SVM, Linear regression, etc) on the quality prediction task. Such results show the effectiveness of the resultant representations.
- A comprehensive performance evaluation on real data set is conducted. The experimental results validate that the strengths of our model on quality prediction, model interpretability, and the effectiveness of the resultant time series embedding.

II. Related work

Our model adopts a multi-task learning framework, builds with an encoder-decoder architecture, and incorporates matrix factorization for decoder design to enable the model interpretability. In the section, we review the state-of-the-art of these techniques.

A. Multi-task learning

Multi-task learning (MTL) is a common strategy to generalize the deep learning model. MTL aims at improving not only the performance of each task but also the generalization of shared features. In addition, MTL acts like a regularizer that prevents the network from the over-fitting issue. For an MTL model that has to fulfill multiple tasks, MTL will find a feature representation that can fit into individual tasks. As a result, the MTL model avoids over-fitting any tasks.

The method proposed in [1] shows that the prediction performance of MTL is better than that of a single task, which implies the shared representation is generalized to support different tasks. A naive approach to build an MTL model is to minimize the loss jointly and treats each task with the same importance in the loss function. Nonetheless, the performance of each task of the model will be highly affected by different importance/weight settings. Tuning weights for multi-task learning is a difficult and expensive task. Various settings are investigated [2][3]. For example, in [2], the authors propose to consider the homoscedastic uncertainty of each task. in [3], the weights are uniform and manually tuned. In this paper, we set the weights of the task uniformly, and the setting of weights is a future work to investigate.

Fast R-CNN [4] also adopts the concept of MTL in their loss function combining classification and bounding-box loss. In addition to computer vision, the work [5] in natural language processing (NLP) also takes an MTL approach. POS (part-of-speech) prediction is often used as features for other NLP tasks. Therefore, [5] consider a POS tagging task, a semantic roles discovery task, a semantically similar words identification task, etc, at the same time in a model. Those tasks are related to a shared lookup-table in the architecture and jointly trained.

B. Matrix Factorization

Matrix factorization (MF) has achieved great success in recommendation applications [6][7]. By factorizing the user-item matrix, one could approximate the original matrix with the multiplication of two low-rank matrices. Both user and item can be projected to a space with a lower dimension to have a compact representation of the item and user profile. MF can also be seen as a clustering method on the equivalence of K-means clustering. For $X \in \mathbb{R}^{d\times n}$,

$$X \approx Z \times H$$

$Z \in \mathbb{R}^{d\times k}$ can be construed as k centroid matrix, and $H \in \mathbb{R}^{k\times n}$ can be regarded as a cluster assignment matrix for n data.

However, MF is a linear method. This means that if there are non-linear relations in the task, a linear method fail to capture the relations. With the rise of deep learning, researches start to incorporate the deep learning techniques for matrix factorization [8][9]. [8] proposes a deep model that factorizes the original image into a serial of multiplication of many low-rank matrices. Each matrix captures part of the property of the original image in an unsupervised manner. In [9] the authors incorporate deep learning techniques with MF and combine multi-view clustering to cluster. By enforcing the representation of the last layer, it constrains the multi-view data to be projected in the same space. Multi-modality can also be captured through deep matrix factorization. The method in [10] uses similar constraints as proposed in [9] on the final layer with logistic loss or least squares to force the model to project different view of data into the same space and then perform the prediction of Alzheimer's disease. [11] proposed a model that utilizes the non-linearity of deep learning to project user and item vector from a user-item matrix into the same vector space, where the similarity between user and item can be calculated through dot product.

C. Encoder-Decoder

The autoencoder idea emerges with deep learning techniques. The Encoder-Decoder framework draws many research attention and was adopted in many deep learning applications. In a common setting, an encoder transforms a given input into a concise latent code, and a decoder uses the latent code to reconstruct the input. VAE [12] takes the same framework but with a different assumption. They assume that the latent code z has its own distribution $p(z)$, and use variational inference to approximate the distribution. Contractive autoencoder [13] adds Jacobian of latent space representation in their objective function, thus forcing the model to become less sensitive to input variations. DAEs [14] works by first corrupting input data, and with the goal of reconstructing the original intact data by deep learning model.

The encoder-decoder framework is also commonly adopted in machine translation tasks. Typically, in the machine translation tasks, the encoder takes a sequence as input, and then the decoder outputs another sequence. [15] and [16] built an encoder-decoder framework with LSTM cells for statistical machine translation. In [17], the authors propose to use CNN instead of RNN with an encoder-decoder architecture to perform language translation. Transformers [18], proposed by Google, used neither CNN nor RNN but fully connected layer and max-pooling to achieve state-of-the-art machine translation result and got rid of the disadvantages of both CNN and RNN.

Convolutional Auto-Encoder (CAE) is also a successful model that combines the merits of convolution and auto-encoder. CAE mainly differs from conventional auto-encoders by the idea that its weights are shared over all locations, so CAE preserves the spatial locality within an image. Some works stacks multiple auto-encoders together like [14], [19], which greedy layer-wise training was used. The model in [20] employs auto-encoder architecture with two inputs, two outputs, and a shared representation.

D. Attention

In natural language processing, attention mechanisms show significant influences on machine translation and language modeling tasks. Moreover, the attention mechanism is also used in different applications like image captioning, question answering, and image generation. Attention itself can be seen as a soft selection over the input. In this way, the model can focus on inputs that are important for emitting outputs. As a consequence, the model can process larger/longer inputs without decreasing performance.

In [21], the author first proposed an alignment model that learns to score the relevance between words around the input position and target position. The score is computed by Bi-LSTM encoder hidden states and a previous decoder state. In [22], the authors employ two kinds of attention mechanisms, global and local. The global attention took all input hidden states into account to compute attention score, and the local attention tries to find an aligned position of the current target word, and then hidden states around that position are used to derive attention score.

ConvS2S [17] introduced convolutional architecture and multi-step attention to tackling with machine translation; each decoder layer will derive attention scores and context vector with the last layer outputs of the encoder. Transformer [18] took another approach; the model is built in a feed-forward neural network which implements self-attention mechanism. Self-attention calculates attention of a position in an input sequence with all positions, and it differs from the alignment mechanism, which requires extra input (decoder) to compute attention score. Recently, SAGAN [23] employed this mechanism and achieved state-of-the-art image generation results.

In [24], [25], attention mechanism is employed to address image captioning tasks. The study in [24] used CNN as an encoder to extract image feature vectors. Next, LSTM is used as a caption decoder which consumes the hidden states, the previous words, and context vectors, computed by the attention function with image features and capturing image information. The study in [25] first used Faster R-CNN together with ResNet-101 CNN to get the features of an image, which

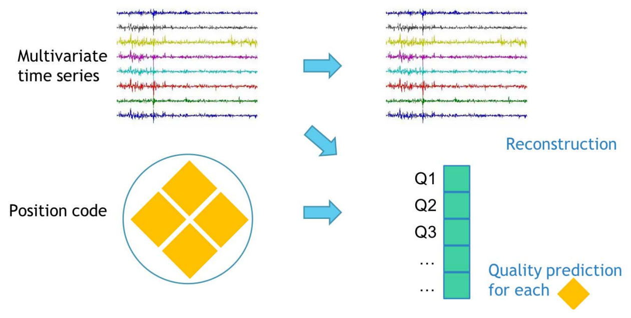


Fig. 2: Inputs are multivariate time series and a position code of a wafer.

they referred to as Bottom-Up attention model, and then the features and generated words is feed into two layers LSTM, in which the first layer LSTM is considered as a top-down visual attention model and the second layer is as a language model.

III. Preliminary

A. Application Scenario

As mentioned, we consider a real photoelectric component production scenario, where the entire production process takes a week to complete and there are two stages for production. The first stage is a chemical process. During the process, the engineers in the factory have to constantly monitor the process and adjust the settings in order to guarantee the quality. Thus, there are different quality between different runs. For understanding the factors, e.g., change of temperature, rotation speed, humidity, and others, of the working station are recorded as multivariate time series. In addition to the sensor reading, according to work site knowledge, the position of a wafer also has effects on final quality. In short, we take multivariate time series and position information in the first stage as our input data, based on which the quality prediction is made. According to site work experiences, the first stage dominates the final production quality. Our goal is to utilize those time series and position information to predict the qualities for a produced photoelectric component. Moreover, understanding individual importance between time series and qualities is also crucial. According to company employees, they can use the correlation to adjust the setting of the station to improve the wafer quality and use the prediction values to decide if they shall terminate the process immediately.

B. Problem Formulation

The inputs of our model are two folds. First, we are given a multivariate time series $\mathbf{D} = \{\mathbf{x_i} \mid i = 1, ..., T, \mathbf{x_i} \in \mathbb{R}^n\}$ generated from a station, where $\mathbf{x_i}$ is a vector of sensor values at time i, n is the number of sensors, and T is the length of the time series in $\mathbf{D}$. Second, we are also given a one-hot encoded position vector $\mathbf{pos}, \mathbf{pos} \in \mathbb{R}^p$, which indicates the position information of the wafer in the first stage operation.

Furthermore, the outputs of our model are three folds. The first output is a quality vector $\mathbf{q} \in \mathbb{R}^m$ indicates the predicted final qualities. The second output is a reconstructed multivariate time series $\hat{\mathbf{D}}$ with respect to the input time series. And, the third output is an attention matrix $\mathbf{A}$ to indicate the correlation between a sensor and a quality measure.

Table I: Notations

Sym	Definition	Dim
n	the number of sensors	$\mathbb{R}$
m	the number of qualities to be estimated	$\mathbb{R}$
T	the length of multivariate time series	$\mathbb{R}$
k	hyperparameter of matrix factorization	$\mathbb{R}$
d	the dimension of latent representation	$\mathbb{R}$
q_i	quality value of quality i	$\mathbb{R}$
λ	hyperparameter of regularization	$\mathbb{R}$
$\mathbf{x_i}$	time series values at time i	$\mathbb{R}^n$
$\mathbf{z}$	latent representation of multivariate time series	$\mathbb{R}^d$
$\mathbf{pos}$	one-hot encoding of position	$\mathbb{R}^p$
$\mathbf{c_i}$	context vector of quality i	$\mathbb{R}^k$
$\mathbf{q}$	quality vector of a wafer	$\mathbb{R}^m$
$\mathbf{D}$	input multivariate time series data	$\mathbb{R}^{n\times T}$
$\mathbf{V}$	embeddings of each input dimension	$\mathbb{R}^{n\times k}$
$\mathbf{H}$	embeddings of each input time point	$\mathbb{R}^{k\times T}$
$\mathbf{A}$	attentions between different dimensions and qualities	$\mathbb{R}^{m\times n}$
$\mathbf{A_z}$	attentions generated by multivariate time series data	$\mathbb{R}^{m\times n}$
$\mathbf{A_{off}}$	attention calibration generated by position code	$\mathbb{R}^{m\times n}$
$\mathbf{C}$	context vector of qualities	$\mathbb{R}^{m\times k}$

- **Input**:
  - a multivariate time series $\mathbf{D}$
  - a position code $\mathbf{pos}$
- **Output**:
  - a quality vector $\mathbf{q}$ for a wafer.
  - a reconstructed input time series $\hat{\mathbf{D}}$
  - an attention matrix $\mathbf{A}$

A complete list of notations is summarized in Table I.

IV. Method

A. Network Architecture

We propose to jointly address the tasks of predicting quality and reconstructing input time series by a multi-task learning model. Namely, our model is designed to conduct the input time series reconstruction and the product quality at the same time by a neural network. Our method resembles the network architecture [26], [27]. The architecture of our model is shown in Figure 3. Essentially, our model is an encoder-decoder neural network with one encoder and three decoders.

The encoder's goal is to compress the input time series $\mathbf{D}$ into a compact representation $\mathbf{z}$. In this work, we use multi-layer perceptron (MLP) to implement the encoder. A feed-forward network acts as a feature extractor to compresses the input $\mathbf{D}$ into a compact latent code $\mathbf{z}$. The latent code $\mathbf{z}$ is the output of the encoder and will be consumed by the decoders in our model.

The decoders' tasks are to use the latent vector $\mathbf{z}$ from the encoder and the position code to perform the input time series

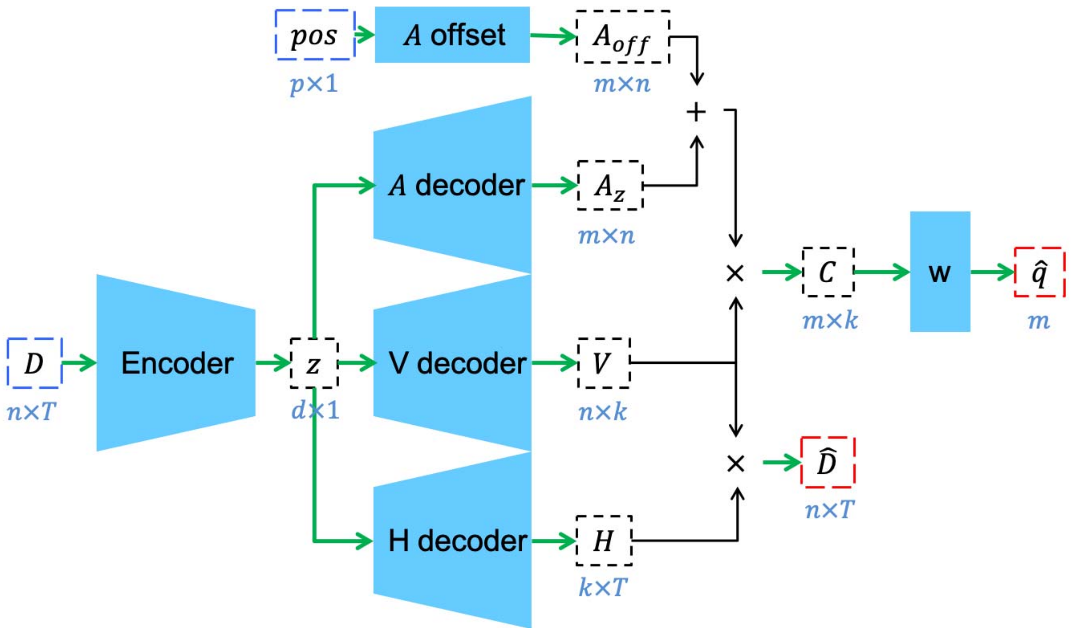


Fig. 3: Model architecture. The input data $\mathbf{D}$ will go through the encoder and be transformed into $\mathbf{z}$. Then, $\mathbf{z}$ will be used as input for three decoders: (1) Attention Decoder A, (2) Time Series Decoder V, and (3) Sensor Factor Decoder H. The decoded results of A and V is employed for quality prediction, and the result of V and H is employed for the time series reconstruction.

reconstruction and the final quality prediction. As shown in Figure 3, we have three decoders: (1) Attention Decoder A, (2) Time Series Decoder V, and (3) Sensor Factor Decoder H.

- The attention decoder generates a $m \times n$ matrix $\mathbf{A}$ to indicate the correlations among the m quality measures and the n sensor time series. Note that with $\mathbf{A}$ the importance between a sensor and a quality measure can be measured.
- The time series decoder V generates $n \times k$ matrix $\mathbf{V}$. $\mathbf{V}$ can be seen as a time series embedding as it reduces the dimensions of the original time series data $\mathbf{D}$ from $n \times T$ to $n \times k$.
- The sensor factor decoder H generates $k \times T$ matrix $\mathbf{H}$. $\mathbf{H}$ can be seen as a sensor factor embedding as it transforms the dimensions of the original time series $\mathbf{D}$ from $n \times T$ to $k \times T$.

With the decoders, our goal is to employ them in a multi-task learning model with two targets: the input time series reconstruction and the wafer quality prediction. As shown in Figure 3, for the input time series reconstruction (see Subsection IV-C for details), we approximate $\hat{\mathbf{D}}$ by $\mathbf{V}$ and $\mathbf{H}$, and for the quality prediction (see Subsection IV-B for details), we employ $\mathbf{A}$ and $\mathbf{V}$. The entire network is jointly optimized in a supervised learning manner, and the attention matrix $\mathbf{A}$, the latent code $\mathbf{z}$, and the embeddings (i.e., the sensor factor embedding and the time series embedding) are by-product results of the proposed multi-task learning model.

B. Quality Prediction

With the architecture introduced above, the task of quality prediction can be modeled as finding a function $\sigma_{q.predict}()$ to predict qualities of a wafer based on the decoder results $\mathbf{A}$ and $\mathbf{V}$. Namely,

$$\hat{\mathbf{q}} = \sigma_{q.predict}(\mathbf{AV})$$

As shown in the network architecture, we learn $\sigma_{q.predict}$ by a neural network. The neural network takes the input time series $\mathbf{D}$ and the position code $\mathbf{pos}$ as inputs with the goal of minimizing the prediction loss l_q, where

$$l_q = ||\mathbf{q} - \sigma_{q.predict}(\mathbf{AV})||_2$$

to produce $\mathbf{A}$ and $\mathbf{V}$. We use mean squared error to measure the difference between ground truth and predicted values.

C. Time Series Reconstruction

Likewise, the task of the time series reconstruction can be modeled as finding $\mathbf{V}$ and $\mathbf{H}$ to minimize the reconstruction loss l_r, where

$$l_r = ||\mathbf{D} - \mathbf{VH}||_2$$

We also use mean squared error to measure the difference between the ground truth and the construction.

The overall loss of our multi-task learning model is

$$L = l_r + l_q + \lambda\Omega(\theta)$$

, where we treat each task with same weight, and to prevent overfitting, we add *l2* regularization on model parameters θ.

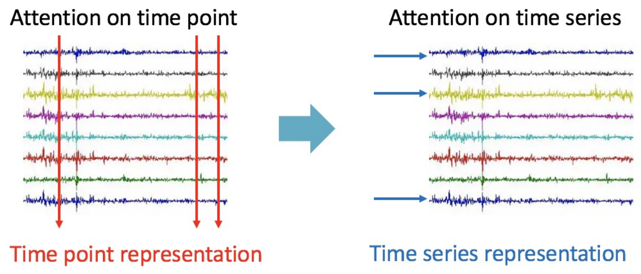


Fig. 4: The goal of the attention decoder is to learn to pay attention on time series rather on time point (as the existing attention mechanism did)

D. Attention Decoder

In this subsection, we introduce how the attention mechanism is designed. Given $\mathbf{z}$ and $\mathbf{pos}$, a naive idea is to concatenate $\mathbf{z}$ and $\mathbf{pos}$ as an input to the attention decoder for learning attention. However, doing so will cause the factor/influence by $\mathbf{pos}$ to be nearly ignored, as the dimensionality of $\mathbf{z}$ is far larger than the dimensionality of $\mathbf{pos}$. By considering this fact, two networks are designed. The first network takes $\mathbf{z}$ as the input and decodes it into $\mathbf{A_z}$, and the second network transforms $\mathbf{pos}$ into $\mathbf{A_{off}}$, which can be seen as an attention calibration for $\mathbf{A_z}$ to consider the position information. In short, we have

$$\mathbf{A} = \mathbf{A_z} + \mathbf{A_{off}}$$

With the attention decoder, we have an attention matrix $\mathbf{A}$ to indicate the correlation between a sensor and a quality measure, as shown in Figure 4. In our design, we do not force attentions between a quality i and all sensor factors to be sum up to 1. The reason is that the influence among the sensor factors should be treated as a multi-label classification problem, which means those influences are not mutual exclusive.

E. Context Vector

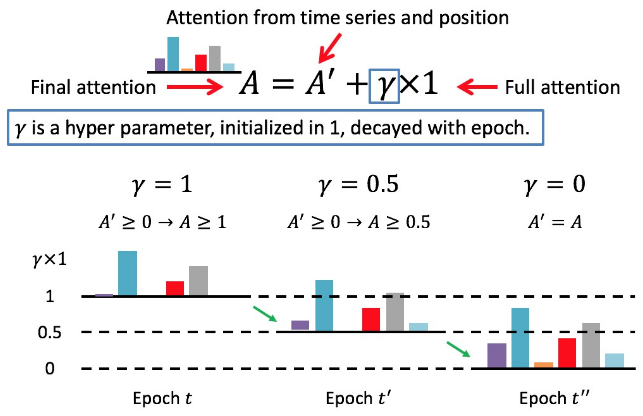
As introduced, in our model, we predict the quality vector $\mathbf{q}$ by

$$\hat{\mathbf{q}} = \sigma_{q.predict}(\mathbf{AV})$$

based on the results from the attention decoder and the time series decoder. For ease of discussion, we denote $\mathbf{AV}$ by $\mathbf{C}$ called *Context Matrix*. Note that the context matrix $\mathbf{C}$ is a $m \times k$ matrix, and the row vector $\mathbf{c_i}$ of $\mathbf{C}$ can be seen as a context vector with respect to a quality measure i. Specifically, $\mathbf{c_i}$ itself contains time series information about a run and a wafer position $\mathbf{pos}$; hence, $\mathbf{c_i}$ can act as a compact and low dimension input for other classification models, e.g., SVM, KNN, etc. In the experiment section, we show the context vector indeed boosts the performance of the conventional classification models.

F. Full attention decay

In SAGAN [23], the author proposes using the self-attention mechanism in a generation network. Interestingly, the output



Attention from time series and position
Final attention
$A = A' + \boxed{\gamma} \times 1$
Full attention
γ is a hyper parameter, initialized in 1, decayed with epoch.
$\gamma = 1$
$\gamma = 0.5$
$\gamma = 0$
$A' \geq 0 \rightarrow A \geq 1$
$A' \geq 0 \rightarrow A \geq 0.5$
$A' = A$
$\gamma \times 1$
1
0.5
0
Epoch t
Epoch t'
Epoch t''

Fig. 5: Full attention decay. The baseline of attention score decreases as the epoch increases.

$\mathbf{o_i}$ of attention layers multiplied by a scalar γ is added back to input feature map $\mathbf{x_I}$, and the scalar γ is initialized with zero in the beginning and learned during training; according to [23], when γ is initialized as zero, the network will focus on clues in the local neighbors which is from pure input feature map $\mathbf{x_i}$, and then gradually the network learns how to assign more weights on output $\mathbf{o_i}$ of attention layer which reflects non-local evidence. The formula is as follow,

$$\mathbf{y_i} = \gamma \times \mathbf{o_i} + \mathbf{x_i}$$

Inspired by SAGAN [23], we apply the same technique to our attention with γ initialized as 1 and decayed with epochs.

$$\mathbf{A} = \mathbf{A}^{'} + \mathbf{1} \times \gamma$$

$\mathbf{A}^{'}$ means the output of $\mathbf{A}$ decoder, and $\mathbf{1}$ is a matrix filled with 1 with the same dimension as $\mathbf{A}^{'}$. During training, we set a baseline attention score as a condition for the network and assume all the sensors have the same effect on quality, which we refer as *full attention* from all sensors. As a result, we add *full attention* $\mathbf{1}$ multiplied with γ which is initially set to 1 to learn attention. In other words, the attention score is at least 1. When the epoch number increase, the γ decrease to gradually lower the baseline attention score, as shown in Figure 5. In each epoch, the network learns to give attention scores based on the baseline of that epoch. After a few epochs, γ will reduce to zero, and the network will learn to assign scores where the baseline now is 0, which is what we want the model to learn directly. For the rest of epochs, the network tunes the attention on condition that the baseline score equals 0. This helps the network converge to better results; the network will learn how to give attention step by step, from easy (baseline = 1) to hard (baseline = 0) condition.

V. Performance Evaluation

In this section, we present the performance evaluation results to demonstrate and validate the superiority and effectiveness of the proposed multi-task learning model. The goals of the performance evaluation are as follows.

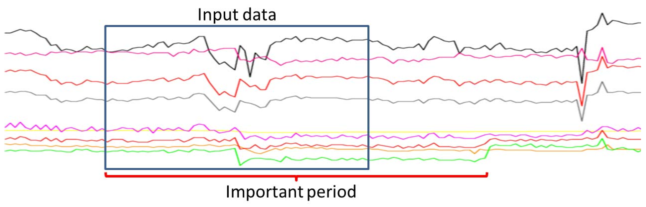


Fig. 6: We select the time series and time period which are crucial for final quality based on domain experts for each run

- Quality Prediction: the goal here is to compare the prediction accuracy between our method and the existing methods to see the effectiveness of the proposed model on the quality prediction task. (Subsection V-D1)
- Time Series Reconstruction: the goal of this experiment is to validate whether our model can reconstruct $\mathbf{D}$ by

$$\mathbf{D} \approx \mathbf{V} \times \mathbf{H}$$

  (Subsection V-D2)
- Model Interpretability: One of the outcomes of our model is the attention matrix $\mathbf{A}$, which equip our model the capability of understanding the correlation between a time series and a quality. (Subsection V-D3)
- Context Vector and Latent Vector: The by-products of our model is the context vector $\mathbf{c_i}$ and the latent vector $\mathbf{z}$. The experiment goal is to check the effectiveness of $\mathbf{c_i}$ and $\mathbf{z}$ as a low-dimension representation. We experimentally incorporate $\mathbf{c_i}$ and $\mathbf{z}$ as inputs for the off-shelf classifier models, e.g. SVM, Logistic Regression, etc., and compare the performance difference of the same model but with the original time series and the context/latent vector as input. (Subsection V-D4)
- Model Transferability: The dataset we obtained are a collection of multivariate time series from different manufacturing stations. We experimentally deploy our model in a cross-station scenario. The goal is to testify that our model possesses the capability to predict quality for the product generated from different stations. (Subsection V-E)
- Time Series Attention: The experiment goal is to testify that the learned attentions will outperform a naive average attention strategy. We also conduct experiments to show the proposed model will converge to the better result under the proposed full attention decay policy. (Subsection V-D3)

A. Dataset

The experimental datasets are from a photoelectric component production company. We have two datasets. The first dataset consists of the data of 1046 runs from a station, and the second dataset consists of the data of 3479 runs from another station. The two datasets are with the same time series number and the same time series length. In each run, sensors reading mounted on the station and the position of a wafer are recorded

Table II: Quality prediction

	LR	**SVR**	**Enet**	**KNNR**	**NN**	**Our**
fold 1 MSE	4.744	5.455	5.970	6.101	4.583	4.786
fold 2 MSE	4.439	5.698	6.127	3.874	4.429	3.751
fold 3 MSE	4.693	4.814	5.113	4.416	4.043	4.111
fold 4 MSE	4.842	6.266	7.054	6.090	5.050	4.523
fold 5 MSE	5.174	5.894	6.647	5.023	4.315	4.353
fold 6 MSE	4.123	4.036	4.338	4.129	4.490	4.298
fold 7 MSE	8.435	7.388	7.317	4.708	7.148	7.191
fold 8 MSE	14.518	7.384	8.019	5.610	6.571	6.331
fold 9 MSE	4.850	5.659	5.875	5.090	4.399	4.095
fold 10 MSE	4.412	6.019	6.729	4.410	4.935	4.307
MSE AVG	6.023	5.861	6.319	4.945	4.996	**4.774**

as the input time series and the position code. Also, each run is also associated with a set of post-examined quality measures on the photoelectric components.

We use interpolation to overcome missing data in a time series and standardize time series across various runs. To prevent the GPU memory allocation problem, we first remove the time series whose value is not changed over time. Second, we select the time series and time period which are crucial for final quality based on domain experts for each run, as shown in Fig. 6. In our data set, each run is with 36 by 9900 multivariate time series data. As a concern of the training time and GPU memory limitation, we downsample the time series over time axis to reduce the input scale. After the downsampling, we have a 36 by 309 multivariate time series data.

B. Implementation and Training

We use the multi-layer perceptron to build our encoder and decoders. The entire model is composed of one encoder and three decoders, which is inspired by [1] that has one encoder and two decoders. However, as introduced in Section IV, we replace the output layer of the decoder by a matrix factorization process. As a result, the output layer of the decoder will produce two matrices with linear activation function, and the product of those matrices has to be close to the input data. The attention decoder is also implemented by a multi-layer perceptron. The output layer dimension is $m \times n$ with sigmoid activation function. The latent vector's dimension d is set to 2500 such that $\mathbf{z}$ could capture apparent features and minor differences in data. The latent dimension of matrix factorization, i.e. k, is set to 20. We initialize weights of ReLU units with He initialization [28] and weights of linear and sigmoid units with Xavier initialization [29].

We train this model in an end-to-end manner. There is no dropout in the network, and the weight decay is set to 0.01 during the training phase. We use Adam [30] to optimize our networks jointly. The batch size is set to 16. The learning rate starts at $5e-5$, and decay 0.1 from Epoch 70 to Epoch 170 with a total of 250 epochs. In the first 50 epochs, we train the encoder and decoders to minimize the reconstruction error, and in the rest of epochs, the entire network starts the multi-task training to minimize the quality prediction error and the reconstruction error simultaneously. The idea of training the time series reconstruction first can be seen as a pre-train process; we first find weights that can minimize the reconstruction error, and then use those weights as the initial settings and fine-tune the entire model to minimize the total errors. γ is initially set to 1 and start decay from Epoch 50. When epoch reaches 150, γ becomes 0. We implemented this model using the Tensorflow framework. Our model is trained on an Nvidia 1080 Ti with Ubuntu 16.04. The total training time takes two hours.

C. Compared Methods

To evaluate the performance of our model, we implement and compare the following baseline method.

- **LR**: We implement a simple linear regression model by learning linear weights on the dimension of inputs.
- **SVR**: support vector regression uses the same principle used in support vector machine with the goal of minimizing error and maximizing the margin.
- **Enet**: elastic net uses the same loss formula as linear regression, but with different regularization which combines L1 and L2 penalty.
- **KNNR**: k nearest neighbor regression adopts the concept of the k-nearest neighbor classifier, where all available instances are stored and use k-nearest neighbors to predict new value.
- **NN**: We implement a neural network with multi-layer perceptron with ReLU activation function. The first three layers are the same as our multi-task encoder, and the rest of the layers are a fully connected layer with 1024 neurons.
- **Ours**: The multi-task learning model with the architecture and the attention mechanism as introduced in the previous sections.

D. Experiment Results

In this section, we show the comparison results and demonstrate the effectiveness of the learned attention weights through data visualization.

1) Quality Prediction: As stated, we compare our method with support vector regression **SVR**, linear regression **LR**, k nearest neighbor regression **KNNR**, and elastic net **Enet** in terms of the mean squared error (M.S.E.) measure. The compared baseline methods are implemented in sklearn Python library. Also, we built a network only for predicting the quality, and show that model gain benefits from reconstruction constrain and weighting mechanism. We flatten multivariate time series $\mathbf{D}$ and concatenate $\mathbf{pos}$ with it to form a vector as input for the baseline methods. The performance result of the compared methods over ten-fold cross-validation is shown in Table II.

We have two observations about this set of performance evaluation results. First, as expected, we can see the proposed method significantly outperform the compared baseline methods. Second, one can be observed that **KNNR** outperforms **SVR**, **LR**, and **Enet** as also shown in Table II. The potential reason is that **KNNR** predicts the quality of a wafer based on selecting a similar time series as a reference. And, in the production scenario, the wafers (with different position code) in a run have the same $\mathbf{D}$, and therefore **KNNR** may make prediction based on the result of the other wafers in the same run, as they have the same $\mathbf{D}$ (Note that the representation of the wafers in the same run differs only in position information), and use the result of the other wafers in the same run is a good heuristic, as the qualities of the wafers in a run are naturally highly correlated.

We also show the effect of varying the value of k on quality prediction task. As mention above, the k controls the embedding dimension of matrix factorization, and the embedding later will be aggregated into a context vector to produce quality value. Hence, k should have effects on quality prediction. The result is shown in Fig. 8a. Fig. 8a reflects the best performance when k is set to 20 or 25, but performance degrades when k is 30. We speculate that this due to the capacity of our decoders.

2) Time Series Reconstruction: In this subsection, we show the result on the time series reconstruction task in order to verify that our model not only predicts quality well but also successfully reconstruct the input time series. Note that when matrix factorization techniques are employed, the hyperparameter k dominates the reconstruction error; increasing k can reduce the difference between input data and the reconstructed data. We also observe this phenomenon in our model. As shown in Figure 8b, when we increase k to 30, we achieve minimal reconstruction error. When k is set to lower than 10, the model is still able to output a reasonable $\mathbf{V}$ and $\mathbf{H}$ so that their product can resemble the original data.

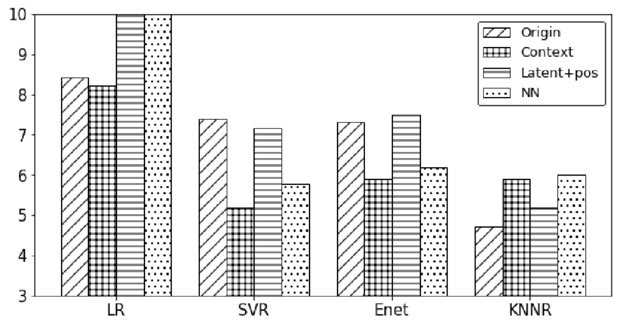
3) Attention: In this subsection, we show attention scores learned by our model. We show the attention score by a heap map visualization, where the darker the cell, the more important a sensor's readings. For scores close to 1, it means that a sensor's reading is highly correlated with a quality measure, vice versa. As illustrated in Figure 9, some of the scores learned by the model are close to zero, which means that the sensor readings did not have any influence on the resultant product quality.

According to our observation, even the run is different, the attention score that equal to zero is still in the same position. The only things that change are the value of non-zero scores. Moreover, we create this figure by feeding model with the same run and $\mathbf{pos}$ varying from position to position (one-hot encoding). It is obvious that $\mathbf{pos}$ will cause some attention change, but for the attention that is highly or not related to quality prediction, they do not affect by $\mathbf{pos}$.

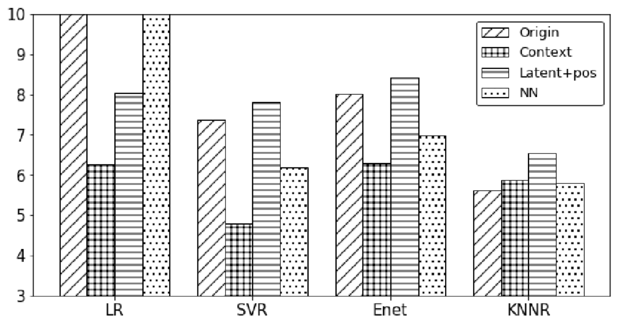
To show that attention learned by our model better fit the data and are effective, we compare our model with a model which has the same architecture as ours, but without $\mathbf{A}$ decoder. In this model, it only has a decoder with matrix factorization. To compute a *context vector* $\mathbf{c_i}$, the model

Table III: Learned attention v.s. Average attention effect on quality prediction

	Fold 1	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Fold 6**	**Fold 7**	**Fold 8**	**Fold 9**	**Fold 10**	**MSE AVG**
Learned	4.786	3.751	4.111	4.523	4.353	4.298	7.191	6.331	4.095	4.307	**4.774**
Average	5.920	4.880	4.797	6.037	4.736	4.337	9.060	7.682	5.063	6.173	5.869



(a) fold 7

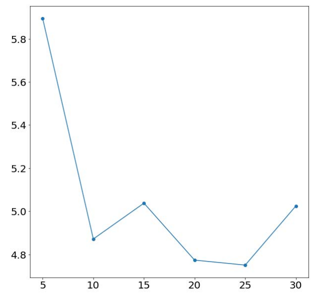


(b) fold 8

Fig. 7: Prediction performance comparison of different inputs. Measure in M.S.E.

Table IV: Downsample effect on quality prediction

	fold 1	**fold 2**	**fold 3**	**fold 4**	**fold 5**	**fold 6**	**fold 7**	**fold 8**	**fold 9**	**fold 10**	**MSE AVG**
309 input	4.786	3.751	4.111	4.523	4.353	4.298	7.191	6.331	4.095	4.307	4.774
618 input	4.977	3.850	4.379	4.407	3.802	4.416	7.611	5.733	4.654	4.427	4.826


(a) k v.s. quality prediction error

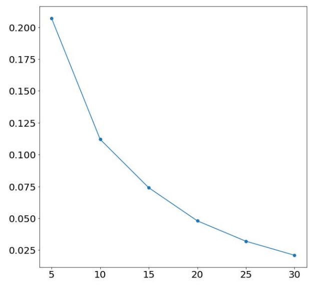

(b) k v.s. reconstruction error

Fig. 8: The effects of varying the matrix factorization hyper-parameter k, where the x axis is values of k and the y axis is the mean square error with respect to various k values.

average overall $\mathbf{r_j}$.

$$\mathbf{c_i} = \frac{1}{n}\sum_{j=1}^{n}\mathbf{r_j}, i = 1...m$$

It means that we treat each $\mathbf{r_j}$ the same importance; in other words, all $\mathbf{r_j}$ contribute the same to the quality prediction task. The result is shown in Table III.

4) Context Vector and Latent Code: The by-products of our model are context vectors $\mathbf{c_i}$ and latent codes $\mathbf{z}$. To confirm that the model indeed produces an effective context vector, we conduct an experiment to validate this fact. We treat our model as a pre-train model so that we can get $\mathbf{c_i}$ for every wafer. $\mathbf{c_i}$ itself contains time series information about a run and wafer position $\mathbf{pos}$; hence, $\mathbf{c_i}$ can act as a compact and low dimension input for **LR**, **SVR**, **Enet**, and **KNNR**. Besides, the latent code $\mathbf{z}$ can also be input for other models, but we still need to concatenate $\mathbf{pos}$ to indicate the difference between wafer.

In Table II, most of the model shows bad result in **fold 7, 8**; our model exhibits relatively better performance. We want to know whether our model learns to produce compact representation for input data $\mathbf{D}$ and $\mathbf{pos}$ to perform better. As a result, we examine $\mathbf{c_i}$ and $\mathbf{pos}$ respectively as input to find if those can boost the performance of other models. In contrast, feature in the neural network merely optimized for quality prediction might yield better features. This makes us extract features before regression layers of the neural network which is used in Table II. The result is in Fig. 7.

It is obvious that our context vector reduces MSE in **LR**, **SVR**, and **Enet**; on the other hand, the latent vector concatenated $\mathbf{pos}$ gives limited improvement, and the performance of **LR** is actually worse than other two kinds of input. Representations from neural network also shows promising improvements on **SVR** and **Enet**. By visualizing context vector (Figure 10) using t-SNE, it can be seen that the context vector which corresponds to the same position code tends to stay together as a cluster, and the brighter the color, the higher the ground truth quality value. This shows the effect of $\mathbf{A_{off}}$ decoder. Moreover, within each cluster, high-quality value tends to stay at a side or the center of a cluster, and low-quality

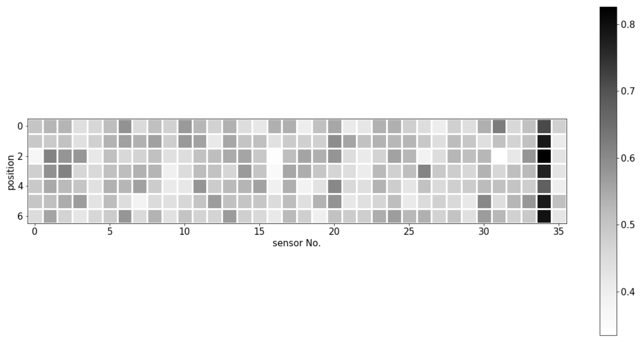
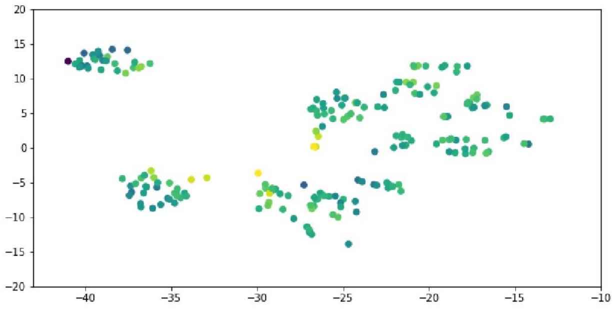
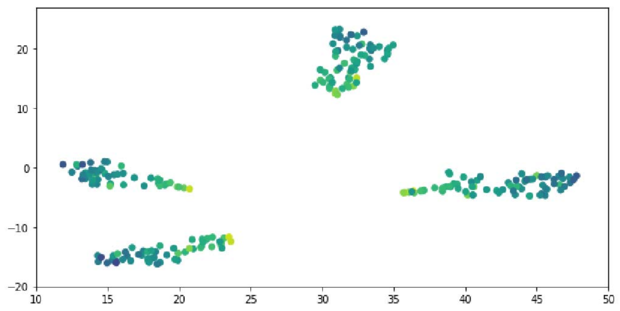
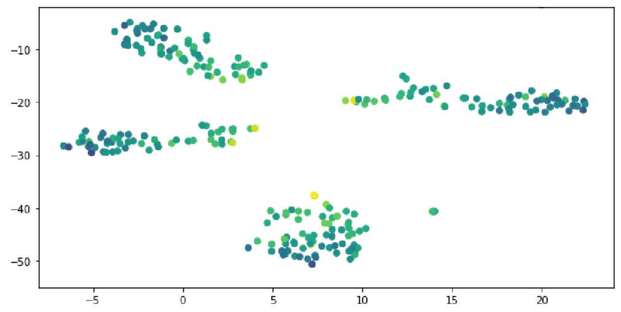


Fig. 9: Visualization on Attentions. The x-axis of this image is 36 different sensors. We could not show the name of each sensor due to a non-disclosure agreement. The y-axis is the position code which starts from index 0 to 7 to indicate the position information of wafers. Each small block represents the attention score for a sensor with respect to a quality measure in consideration.


(a) Training data of Fold 2


(b) Training data of Fold 3


(c) Training data of Fold 7

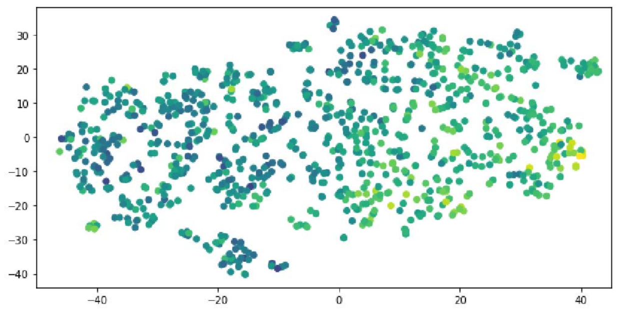

(d) Training data of Fold 8

Fig. 10: Context vector visualization. Those are only a part of visualization.

Table V: Full attention initial value effect on quality prediction

init value	**0**	**0.5**	**1**	**1.5**	**2**
MSE	4.668	3.831	3.930	4.009	4.321

value stands at the opposite place. This means that there are some rules for our model to do the quality prediction within each cluster given position code **pos**.

5) Downsample Effect: In this experiment, we focus on verifying the influence of downsampling multivariate time series data. Downsampling will make the size of a data instance become smaller, but it also means that this process removes information from the data. To show that our model still can predict well even losing some information, we evaluate two models that take different scales of data as input by mean squared error. The first model that takes 36 by 309 as input is the one we used in previous experiments. The second model takes 36 by 618 as input. The comparison results are summarized in Table IV.

It shows that even if we increase twice data points in a run, the effect of downsampling is relatively small in each fold. The model still can take downsample data as input to predict regression without harming performance. This helps to save memory in GPU, and training and inference time also reduce

Table VI: Cross station quality prediction

	LR	SVR	Enet	KNNR	NN	Our
MSE AVG	127.69	6.973	7.797	6.131	6.236	6.459

by downsampling.

6) Full attention decay: In full attention decay formula, we set every sensor with full attention $\mathbf{1}$ as an initial value at the beginning of the training. However, we could alter the value. To be specific, the attention score could be larger than 1 or 0. As a result, we try a different initial value of full attention to see if it does affect the performance of quality prediction. The decay epochs are the same for all initial value, and MSE is calculated as the average of last 20 epochs of Fold 2. The result is shown in Table. V.

For initial value equals 0, it means that we do not use full attention decay in training; namely, the model learns attention directly. Comparing initial value 1 with initial value 0, the result shows that this trick does help the model to converge better. However, when full attention is getting larger, the model fails to learn correct score assignment. We hypothesize that it is due to the larger difference of baseline between previous and current epoch. When baseline decreases to dramatically, the baseline could not effectively constrain the model to learn giving attention scores gradually.

E. Cross station quality prediction

The final experiment is to predict the quality of products from different stations. This task needs extra information to indicate the model. Hence, we use some categorical data suggested by domain experts, for example, type of stations, material. We transform those categorical data into one hot encoding and concatenate them with position code $\mathbf{pos}$. This code will be sent to A_{off} decoder to generate $\mathbf{A_{off}}$. We also enlarge the number of neurons in encoder and decoder to keep more information. The dimension d of latent code becomes 3000, and k is set to 13. Where other conditions remain the same, such as data preprocessing, environment, training procedure, etc.

In VI, we could see that every model perform worse than a single station experiment. To further check this problem, we found that among 348 testing data, there are about 11 examples contribute over 30 MSE to total MSE, which is about 3% in testing data. After having the discussion with domain experts, we found out that our model performs badly on certain condition. In our task, we use a multivariate time series as input to predict output quality scalar. The target quality is an average value over many quality testing samples on a wafer. Hence, if input data has anomalous events that cause a wafer has uneven qualities on it, the mean value can not reflect uneven problem, which the mean quality value is normal. This might make the model biased during training, and cause it to converge to a bad local minimum.

VI. Conclusion

In this paper, we propose a neural network model to predict quality in the early stage of industrial manufacture product based on multivariate time series. The model employs multi-task learning, encoder-decoder architecture, matrix factorization, and attention mechanism. We carry out a series of experiments to validate the performance of the proposed framework. The experiment results validate that (1) our model incorporating multi-task learning indeed outperform baseline models (the models consider only single-task learning), (2) the model interpretability, by validating by domain experts, the proposed attention mechanism indeed provides interpretability between sensors' reading and product quality measure, (3) the produced context vector and latent vector can serve an effective feature representation for the input time-series, (4) the model transferability, the model learned from one station can also effectively predict the quality of the product from another station.

Acknowledgment

This work is supported in part by the Ministry of Science and Technology, Taiwan, under grant No: 107-2218-E-009-053, 106-2628-E-009-008-MY3, and 107-2221-E-005-064-MY2.

References

[1] N. Srivastava, E. Mansimov, and R. Salakhudinov, "Unsupervised learning of video representations using lstms," in *International conference on machine learning*, 2015, pp. 843–852.

[2] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[3] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2650–2658.

[4] R. Girshick, "Fast r-cnn," in *Computer Vision (ICCV), 2015 IEEE International Conference on*. IEEE, 2015, pp. 1440–1448.

[5] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 160–167.

[6] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, no. 8, pp. 30–37, 2009.

[7] H. Ma, H. Yang, M. R. Lyu, and I. King, "Sorec: social recommendation using probabilistic matrix factorization," in *Proceedings of the 17th ACM conference on Information and knowledge management*. ACM, 2008, pp. 931–940.

[8] G. Trigeorgis, K. Bousmalis, S. Zafeiriou, and B. W. Schuller, "A deep matrix factorization method for learning attribute representations," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 3, pp. 417–429, 2017.

[9] H. Zhao, Z. Ding, and Y. Fu, "Multi-view clustering via deep matrix factorization." in *AAAI*, 2017, pp. 2921–2927.

[10] Q. Wang, M. Sun, L. Zhan, P. Thompson, S. Ji, and J. Zhou, "Multi-modality disease modeling via collective deep matrix factorization," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 1155–1164.

[11] H.-J. Xue, X.-Y. Dai, J. Zhang, S. Huang, and J. Chen, "Deep matrix factorization models for recommender systems," *static. ijcai. org*, 2017.

[12] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[13] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*. Omnipress, 2011, pp. 833–840.

[14] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 1096–1103.

[15] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[16] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1724–1734.

[17] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," in *International Conference on Machine Learning*, 2017, pp. 1243–1252.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 6000–6010.

[19] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction," in *International Conference on Artificial Neural Networks*. Springer, 2011, pp. 52–59.

[20] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 689–696.

[21] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[22] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[23] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," *arXiv preprint arXiv:1805.08318*, 2018.

[24] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, 2015, pp. 2048–2057.

[25] P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, and L. Zhang, "Bottom-up and top-down attention for image captioning and vqa," *arXiv preprint arXiv:1707.07998*, 2017.

[26] Z. Lin, M. Feng, C. N. d. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, "A structured self-attentive sentence embedding," *arXiv preprint arXiv:1703.03130*, 2017.

[27] S. Zhai, K.-h. Chang, R. Zhang, and Z. M. Zhang, "Deepintent: Learning attentions for online advertising with recurrent neural networks," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM, 2016, pp. 1295–1304.

[28] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.