XXXXXXX

First Author [#1], Second Author [*2], Third Author [#3]

[#] *First-Third Department, First-Third University*
Address Including Country Name
[1] first.author@first-third.edu
[3] third.author@first-third.edu

[*] *Second Company*
Address Including Country Name
[2] second.author@second.com

***Abstract*—For your paper to be published in the conference proceedings, you must use this document as both an instruction set and as a template into which you can type your own text. If your paper does not conform to the required format, you will be asked to fix it.**

I. Preliminary

Recent advances in deep learning methods based on artificial neural networks have led to breakthroughs in long-standing AI tasks such as speech, image, and text recognition, language translation, etc. Companies such as Google, Facebook, and Apple take advantage of the massive amounts of training data collected from their users and the vast computational power of GPU farms to deploy deep learning on a large scale. The unprecedented accuracy of the resulting models allows them to be used as the foundation of many new services and applications, including accurate speech recognition and image recognition that outperforms humans.

A. Multilayer Perceptrons

Multilayer perceptron (MLP), which is a type of simple feed-forward nerual networks, is the foundation of deep learning architectures, and it is the most common form of deep learning architectures. Figure 1 shows a typical neural network with one hidden layers. Each node in the network is a neutron. In multilayer perceptron, each neuron receives the output of the neurons in the previous layer plus a bias signal from a special neuron that emits 1. It then computes a weighted average of its inputs, referred to as the total input. The output of the neuron is computed by applying a nonlinear activation function to the total input value. The output vector of neurons in layer k is ak = f(Wk ak?1), where f is an activation function and Wk is the weight matrix that determines the contribution of each input signal. In this case, the output of each neuron j in the last layer is the relative score or probability that the input belongs to class j.

B. Convolutional Neural Networks

Convolutional neural network (CNN) is an variant of feed-forward neural networks, which has successfully been applied to analysis visual Image. The architecture of a typical CNN is structured as series of stages. The first few stages are composed of two types of layers: convolutional layers and pooling layers. Units in a convolutional layer are organized in feature maps, within which each unit is connected to local patches in the feature maps of the previous layer through a set of weights called a filter bank. The result of this local weighted sum is then passed through a non-linear activation function. All units in a feature map share the same filter bank. Different feature maps in a layer use different filter banks. The reason for this architecture are two folds. First, in array data such as images, local groups of values are often highly correlated, forming distinctive local motifs that are easily detected. Second, the local statistics of images and other signals are invariant to location. In other words, if a motif can appear in one part of the image, it could appear anywhere, hence the idea of units at different locations sharing the same weights and detecting the same pattern in different parts of the array. Mathematically, the filtering operation performed by a feature map is a discrete convolution, hence the name.

Although the role of the convolutional layer is to detect local conjunctions of features from the previous layer, the role of the pooling layer is to merge semantically similar features into one. Because the relative positions of the features forming a motif can vary somewhat, reliably detecting the motif can be done by coarse-graining the position of each feature. A typical pooling unit computes the maximum of a local patch of units in one feature map (or in a few feature maps). Neighbouring pooling units take input from patches that are shifted by more than one row or column, thereby reducing the dimension of the representation and creating an invariance to small shifts and distortions. Two or three stages of convolution, non-linearity and pooling are stacked, followed by more convolutional and fully-connected layers. Backpropagating gradients through a ConvNet is as simple as through a regular deep network, allowing all the weights in all the filter banks to be trained.

C. Recurrent Neural Network

Recurrent Neural Network (RNN) is another variant of neural networks, which aims at extracting sequential or temporal hidden feature vectors from original dataset. Unlike MLPs and CNNs which are feed-forward neural networks, RNNs has connections between units form a directed cycle. Figure 3 shows a typical structure of RNN. Given an input sequence x = (x1, . . . , xT) a standard recurrent neural network computes

the hidden vector sequence h = (h1, . . . , hT) and output vector sequence y = (y1, . . . , yT) by iterating the following equations from t = 1 to T:

$$\alpha = softmax\left(\mathbf{w}^T \tanh(\mathbf{W}_h \mathbf{H}_a)\right) \quad (1)$$

$$\mathbf{s} = \mathbf{H}\alpha^T \quad (2)$$

where the W terms denote weight matrices (e.g. Wih is the input-hidden weight matrix), the b terms denote bias vectors (e.g. bh is the hidden bias vector) and H is the hidden layer activation function, typically the logistic sigmoid function.

LSTM is very useful in learning long-term dependencies. It has been proposed to address the issue that standard RNN suffers from the severe gradients vanishing or exploding when the neural networks deal with long sequences. The mathematical description of the LSTM structure is as follows:

$$\mathbf{i}_t = \sigma\left(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{W}_{ic}\mathbf{c}_{t-1} + \mathbf{b}_i\right) \quad (3)$$

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{W}_{fc}\mathbf{c}_{t-1} + \mathbf{b}_f\right) \quad (4)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \phi\left(\mathbf{W}_{cx}\mathbf{x}_t + \mathbf{W}_{ch}\mathbf{h}_{t-1} + \mathbf{b}_c\right) \quad (5)$$

$$\mathbf{o}_t = \sigma\left(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{W}_{oc}\mathbf{c}_t + \mathbf{b}_o\right) \quad (6)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \phi\left(\mathbf{c}_t\right) \quad (7)$$

Formally, at time step t, each cell in LSTM is constructed by an input gate $\mathbf{i}_t$, a forget gate $\mathbf{f}_t$, an output gate $\mathbf{o}_t$, and a memory cell $\mathbf{c}_t$. These gates and memory cell have the same dimension in $\mathbb{R}^d$ as the hidden vector $\mathbf{h}_t$. The cell is activated by current inputs $\mathbf{x}_t(\mathbf{x}_t \in \mathbb{R}^e)$ and previous hidden state $\mathbf{h}_{t-1}$. $\mathbf{W}_{ix}$, $\mathbf{W}_{ih}$, $\mathbf{W}_{ic}$, $\mathbf{W}_{fx}$, $\mathbf{W}_{fh}$,$\mathbf{W}_{fc}$, $\mathbf{W}_{ox}$, $\mathbf{W}_{oh}$,$\mathbf{W}_{oc}$ $\in \mathbb{R}^{2d}$ are weighted matrices, and $\mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_c, \mathbf{b}_o \in \mathbb{R}^d$ are biases of LSTM, which are all learned during training process. σ is the sigmoid function and $\odot$ stands for element-wise multiplication. d denotes the number of the LSTM units and e is the input dimension.

II. Online CSI Prediction Scheme for 5G Wireless Communication

Our proposed scheme, online CSI prediction scheme for 5G wireless communications (OCEAN), is an online neural-based end-to-end system to predict CSI value in various of scenarios. The neural network architecture is shown in Figure. X. It contains several learned neural layers: a 2D convolutional layer, a 2D max-pooling layer, a flatten layer, a 1D convolutional layer, a 1D max-pooling layer, another flatten layer, a recurrent layer with LSTM units, and two fully connected layers. The 2D convolutional layer is several parallelized units, which are connected to a small region in the previous [?feature maps?] through a set of weights. These units, which are called filters, stride through the feature maps in two dimension, vertically and horizontally, and compute the convolutional results of the small region and filter weights. Then, the 2D max-pooling layer capture the max values of these convolutional results. After previous layers, the feature maps becomes a high dimension feature tensor, and the flatten layer is used to reduce the high dimension tensor into a one dimension vector by concatenating. The structure of the following 1D convolutional layer, 1D max-pooling layer and flatten layer is similar to previous 2D ones. The difference between them are that the operations in 2D ones are along two dimension, and along one dimension in 1D ones. The recurrent neural network is a type neural network where the nodes in the same layer connect to each other and form a loop, and it aims at extracting sequential or temporal hidden features. However, it suffers from severe gradients vanishing or exploding when dealing with long sequence. To address this issue, researchers proposed a special unit with memory network architecture called long-short term memory. Each unit is constructed by an input gate, a forget gate, an output gate, and a memory cell. The LSTM layer captures the sequential feature vectors, and connects to the final fully-connect layers. The fully-connected layers are the basic feed forward neural layers which connect to all the output of nodes in previous layer. The final connected-layer has the same size of the input data, and the output is the predicted value from the neural network. Comparing the difference of the predicted value and the ground-truth, we can obtain the systematical loss of the neural models. We can use the gradient descent methods to reduce the loss, and get a well-trained model.

The architecture of OCEAN system is structured as Figure X. The whole system is divided into four stages. We construct CSI information maps from raw CSI data and other side information. The standard convoluntional layers in first stage extract frequency representative (FR) vectors from CSI information maps of different frequency bands. The following 1D convolutional neural network compresses FRV of multiple bands into one state representative (SR) vector. In stage 3, the SR vectors pass through the LSTM layers to predict future SR vectors. At the output of the LSTM layers, we reconstruct CSI information maps from the SR vectors. To automatically adjust the system according to practical scenario, we equip our OCEAN system with online learning mechanism, which makes the system more efficient and stable in practical scenario. In this section, first we introduce CSI information maps, the training dataset used for OCEAN system. Then we describe the details of each stage of our model, like frequency representative vector extraction, state representative vector extraction, state vector prediction and reconstruction of predicted CSI information maps. Finally, we present the online learning mechanism used in OCEAN system.

A. The Proposed Learning Framework

Our proposed deep learning framework is used to conduct online CSI prediction for 5G wireless communications. In what follows, we describe the architecture of the learning framework and the data passing through the learning framework, respectively.

1) The architecture of the learning framework: The learning framework is developed to analyze the historic data for online CSI prediction. The architecture of this learning framework is shown in Fig. **??**, where this architecture consists of a 2D convolutional neural network, a 1D convolutional neural

network and a recurrent neural network with long-short term memory units (LSTM).

A 2D convolutional neural network (2D-CNN) are constructed by a 2D convolutional layer, a 2D max-pooling layer, and a flatten layer. The convolutional layer is several parallelized units, which are connected to a local patch of raw data image through a set of weights. These units, normally called filters, stride through the image along two dimension, vertically and horizontally. While striding, the filters compute the convolutional products. Then, the 2D max-pooling layers find the max values of the filters as their output, which reduce the size of the filters' outputs. After previous layers, the feature maps become high dimension feature tensors, and the flatten layer is used to reduce the high dimension tensors into one dimension vectors by concatenating.

The following network is 1D convolutional neural network (1D-CNN), and it contains 1D convolutional layer, 1D max-pooling layer and a flatten layer. These layers are similar to previous 2D ones. The main difference is that the operations in 2D ones are along two dimension. Meanwhile, the responding operations in 1D ones are along one dimension.

Then, we take advantage of recurrent neural network (RNN), which is excellent at extracting sequential or temporal hidden features of raw data. Generally, the node in RNN are connected to other nodes in the same layer and form a loop. However, the naive RNN suffers from severe gradients vanishing or exploding when dealing with long sequence. To address this issue, researchers proposed a special unit, which is equipped with memory network architecture called LSTM unite, to replace the normal neutron. Each LSTM unit is constructed by an input gate, a forget gate, an output gate, and a memory cell.

Finally, the LSTM layer captures the sequential feature vectors, and connects to the final fully-connect layers. The fully-connected layers are the basic feed forward neural layers which connect to all the output of nodes in previous layer. The final connected-layer has the same size of the input data, and the output is the predicted value from the neural network. Comparing the difference of the predicted value and the ground-truth, we can obtain the systematical loss of the neural models. We can use the gradient descent methods to reduce the loss, and get a well-trained model.

2) The data flow in the learning framework: The input data pass through the proposed learning framework that outputs the predicted CSI. Specifically, the raw data are first preprocessed and transfomred into CSI information images. Then, the CSI information images are fed into a CNN network to extract frequency representative vectors which are later transformed into a state representative matrix. Subsequently, this matrix is sent into another CNN network to compress the state representative matrix as a state representative vector that is afterwards fed into an LSTM network. The LSTM network outputs the predicted state vector that is finally reconstructed as the predicted CSI information images. In what follows, we describe CSI information image, frequency representative vector extraction, state representative vector extraction, state vector prediction, and reconstruction of predicted CSI information image, respectively.

B. CSI Information Maps

Our OCEAN system is an end-to-end prediction model, which we feed in CSI values of all frequency to the neural-based system, and correspondingly we obtain CSI values of all bands. To fit the requirements of the input, we build CSI information maps as training dataset, which combines all bands CSI information of all bands and the map location information together.

Inspired by representation of digital color image, we construct CSI information maps from raw CSI data and other side information. Color digital images are made of pixels, and pixels are made of combinations of primary colors represented by a series of code. A **pixel** is the smallest addressable element in the image. A **channel** is the grayscale image of the same size as a color image, made of just one of these primary colors. Similarly, we segment the map of a certain place into several small cells, which is "CSI pixel". For each frequency band, we collect CSI value and side information in these CSI pixels. The maps, created of CSI value data of same frequency, or same types of side information, are like "CSI information channels". The CSI information maps are the combination of CSI information channels.

The training dataset is the collection of CSI information maps. At each time slot, we collect required data and build the CSI maps Xi. As time goes by, we obtain a CSI information maps sequence(X1,X2,Xi).

C. Frequency Representative Vector Extraction

The CSI information maps contains details of raw information of CSI values and other side information data. For classic machine learning system, we should manually extract features from these data, and select suitable features, which will deeply influence future CSI values. However, the feature extraction and selection processes are heavily depends on expertise on CSI. In addition, the manually extraction and selection are effort-consuming tasks. To address these issues, we utilize the advance of neural-based learning system to extract representative vectors from the CSI information maps.

As show in stage 1 of Figure X, the convolutional neural networks (CNNs) extract frequency representative vectors from CSI information maps. (Here, the vectors obtained by CNN are not only frequency representative vectors. In fact, several vectors represent the side environment information, such as *, *. We ignore the differences between these two types of vectors, and call all of them Frequency Representative Vectors) First, we decompose a CSI information map into multiple CSI information channel

$$\alpha = softmax\left(\mathbf{w}^T \tanh(\mathbf{W}_h \mathbf{H}_a)\right) \tag{8}$$

$$\tag{9}$$

For normal CNNs, we use a number of convolutional filters to process these channels, but for clarity we will explain the CNN networks with one filter. Let wR be a convolutional filter which

we apply to a window of pixels in the channel to generate a feature. A feature C for a window of pixels (P1,P2) is given as follows:

$$\alpha = softmax\left(\mathbf{w}^T \tanh(\mathbf{W}_h \mathbf{H}_a)\right) \tag{10}$$

$$\tag{11}$$

where bv is a bias term and X is the activation function. A feature map C is a collection of features computed from all windows of pixels:

$$\alpha = softmax\left(\mathbf{w}^T \tanh(\mathbf{W}_h \mathbf{H}_a)\right) \tag{12}$$

$$\tag{13}$$

To capture the most salient features in c, we apply a max-over-time pooling operation (Collobert et al., 2011), yielding a scalar:

$$\alpha = softmax\left(\mathbf{w}^T \tanh(\mathbf{W}_h \mathbf{H}_a)\right) \tag{14}$$

$$\tag{15}$$

In the end of the stage 1, we combine all outputs from convolutional filters together, and flatten them into a vector, which is the extracted Frequency Representative Vector.

D. State Representative Vector Extraction

After Stage 1, we obtains FR vectors of all frequency bands and side information. Concatenating these FR vectors together, we build a state representative matrix of a certain time slot. We can use the matrix sequence as input to pass it through recurrent neural networks to capture the temporal features of the sequence. However, sequentially concatenating vectors cannot imply the inference among frequency bands and side information. Meanwhile, the size of dimension of inputs determines the number of parameters required to optimize. If the input size is too large, there are a huge size of parameters need to train, which will spend an unacceptable time, and the system may not converge for lacking enough training samples.

To solve these problems, we use CNNs to compress the state representative matrix into a state representative (SR) vector. The process is similar to FR vectors extraction, but for SR vector extraction, the convolutional computation is only along the one dimension as shown in Figure X1. The output of this CNN is a state representative vector:

$$\alpha = softmax\left(\mathbf{w}^T \tanh(\mathbf{W}_h \mathbf{H}_a)\right) \tag{16}$$

$$\tag{17}$$

E. State Vector Prediction

For OCEAN system, we aims at predicting the future CSI state of the whole space based on previous and current CSI state. From previous stage, We accumulate the captured state representative vectors to form a state vector sequence. Due to the superior performance of recurrent neural networks in different sequence learning tasks, we adopt recurrent neural network to predict future state vector, specifically we choose LSTM-based RNN.

Base on the Equation X, the output of LSTM network layers can be written as:

$$\alpha = softmax\left(\mathbf{w}^T \tanh(\mathbf{W}_h \mathbf{H}_a)\right) \tag{18}$$

$$\tag{19}$$

Then, a softmax layer is followed to transform XX to predicted state vector.

$$\alpha = softmax\left(\mathbf{w}^T \tanh(\mathbf{W}_h \mathbf{H}_a)\right) \tag{20}$$

$$\tag{21}$$

F. Reconstruction of Predicted CSI Information Maps

The output of previous stage is the predicted state vector. However, the goal of our design is to predict the real multi-spectrum CSI status of a specific space, which is represented by CSI information maps. Only obtaining status vector is far from our objective.

**We segment the state vector into n part. The length of segments should equal to the size of the channel in the CSI information map. **

G. Online Learning Mechanism

Although we consider plenty of side information, like temperature, humidity, obstacles etc. in our system design, there still are some factors which may lead to abrupt change of CSI status. To address this issue, we equip our system with online learning mechanism, which can capture any inference to CSI status and make the system more stable in real scenario.

Set Threshold

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

2) Frequency Representative Vector Extraction: After receiving CSI information images, the learning framework first needs to extract features from such images. In learning frameworks based on classic machine learning techniques, features are extracted from datasets manually. However, feature extraction and selection heavily depend on the expertise in CSI analysis. Thus, we propose to explore a deep neural network for performing feature extraction and selection to obtain representative vectors from CSI information images.

In particular, we propose to utilize a CNN network to perform feature extraction and selection for extracting frequency representative vectors from CSI information images. To find such vectors, a CSI information image *IMG* is decomposed into multiple CSI information channels as follows:

$$[C_1, C_2, \cdots, C_l] = \textit{IMG} \tag{22}$$

where $\textit{IMG} \in \mathbb{R}^{h \times w \times l}$, $C_i \in \mathbb{R}^{h \times w}, i \in [1, l]$, w, h, and l are the width, height, and channel number of the image, respectively.

Generally, a normal CNN network has a number of convolutional filters to process these channels. For simple presentation, we describe the CNN network with one filter in this paper. Let w be a convolutional filter that is applied to a window of pixels belonged to a frequency information channel. Specifically,

after a window of pixels x passes a filter, the corresponding feature pixels in the feature map x^* can be obtained as follows:

$$x^* = f\left(x \otimes w + b\right) \tag{23}$$

where b is a bias term and f is the activation function.

Particularly, to capture the most salient features in x^*, a max-pooling operation is applied to yield a scalar.

$$\curvearrowleft^* = max(x^*) \tag{24}$$

Then the filters stride over each channel to obtain corresponding feature maps. As a result, a feature image IMG^*, a collection of features computed from all windows of pixels, can be expressed as follows:

$$IMG^* = [C^*_{11}, C^*_{12}, \cdots, C^*_{1n}, \cdots, C^*_{l1}, C^*_{l2}, \cdots, C^*_{ln}] \tag{25}$$

where n is the number of filters.

At last, the output results from all convolutional filters are combined together and all of them are flattened into all a vector, i.e., called frequency representative vectors.

$$V_{Fi} = concat(C^*_{i1}, C^*_{i2}, \cdots, C^*_{ln}) \tag{26}$$

This process can be briefly summarized as follows:

$$[V_{F1}, V_{F2}, \cdots, V_{Fl}] = conv2D(IMG) \tag{27}$$

3) State Representative Vector Extraction: After obtaining frequency representative vectors of all frequency bands, we need to construct a state representative matrix for a certain time slot by concatenating the obtained frequency representative vectors. Traditionally, the matrix needs to be fed into a RNN network like LSTM to extract temporal features within the data. However, sequentially concatenating vectors is unable to extract the relationship among frequency bands and other features. Moreover, the high dimension of input data results in a great number of parameters to be trained, which may incur the infeasible training time. Besides, the training process may not converge until a large size of training samples are fed into the network.

To solve this problem, we propose a 1D-CNN network to compress the state representative matrix into a state representative vector. The compression process is similar to the process of extracting frequency representative vectors, except that the convolutional computation is only along the one dimension. The state representative vector of time t can be expressed as follows:

$$S_t = conv1D(V_{F1t}, V_{F2t}, \cdots, V_{Flt}) \tag{28}$$

4) State Vector Prediction: Our proposed OCEAN aims at predicting the current CSI of all frequency bands based on historic input data and current side information. Considering that the LSTM network can be excellent in sequential tasks learning, we adopt a LSTM network to predict the current state vector.

LSTM is very useful in learning long-term dependencies. It has been proposed to address the issue that standard RNN suffers from the severe gradients vanishing or exploding when the neural networks deal with long sequences. The mathematical description of the LSTM structure is as follows:

$$\mathbf{i}_t = \sigma\left(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{W}_{ic}\mathbf{c}_{t-1} + \mathbf{b}_i\right) \tag{29}$$

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{W}_{fc}\mathbf{c}_{t-1} + \mathbf{b}_f\right) \tag{30}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \phi\left(\mathbf{W}_{cx}\mathbf{x}_t + \mathbf{W}_{ch}\mathbf{h}_{t-1} + \mathbf{b}_c\right) \tag{31}$$

$$\mathbf{o}_t = \sigma\left(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{W}_{oc}\mathbf{c}_t + \mathbf{b}_o\right) \tag{32}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \phi\left(\mathbf{c}_t\right) \tag{33}$$

Formally, at time step t, each cell in LSTM is constructed by an input gate $\mathbf{i}_t$, a forget gate $\mathbf{f}_t$, an output gate $\mathbf{o}_t$, and a memory cell $\mathbf{c}_t$. These gates and memory cell have the same dimension in $\mathbb{R}^d$ as the hidden vector $\mathbf{h}_t$. The cell is activated by current inputs $\mathbf{x}_t (\mathbf{x}_t \in \mathbb{R}^e)$ and previous hidden state $\mathbf{h}_{t-1}$. $\mathbf{W}_{ix}$, $\mathbf{W}_{ih}$, $\mathbf{W}_{ic}$, $\mathbf{W}_{fx}$, $\mathbf{W}_{fh}$,$\mathbf{W}_{fc}$, $\mathbf{W}_{ox}$, $\mathbf{W}_{oh}$,$\mathbf{W}_{oc}$ $\in \mathbb{R}^{2d}$ are weighted matrices, and $\mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_c, \mathbf{b}_o \in \mathbb{R}^d$ are biases of LSTM, which are all learned during training process. σ is the sigmoid function and $\odot$ stands for element-wise multiplication. d denotes the number of the LSTM units and e is the input dimension.

The update of each LSTM unit can be briefly summarized as follows:

$$\mathbf{h}_t = LSTM(\mathbf{h}_{t-1}, \mathbf{x}_t, \theta). \tag{34}$$

Function $LSTM$ is a combination of Eq. (3-7), and θ represents all the parameters in the LSTM network.

We take the advantage of LSTM network to capture the sequential feature of state representation vector:

$$\mathbf{h}_{St} = LSTM(\mathbf{h}_{St-1}, \mathbf{S}_t, \theta). \tag{35}$$

Then, the fully-connected softmax layers are followed to generate a probability distribution of predicted CSI image, as follows:

$$\widehat{IMG_t} = softmax(\mathbf{W}_h\mathbf{h}_{St} + b) \tag{36}$$

Thus, the prediction loss of our model is:

$$loss = \sum_{t=0}^{T} \frac{1}{2}\left(IMG_t - \widehat{IMG_t}\right)^2 \tag{37}$$

1) Online XXXXXX Mechanism: To train our model precisely and stably, we apply a online and offline two-step training process to guide the updates of weights in our model.

As show in Figure. X, we first conduct the offline training process in our model, and this process is to capture the sequential pattern of CSI series along the time based on huge volume of historical data. Reducing the system loss by back propagation based stochastic gradient descent algorithm, we obtain a well-trained neural network model, where the weights of our model is represented by $\mathbf{W_h}$.

However, although we consider plenty of side information, like temperature, humidity, obstacles etc. in our prediction system, there are always some other factors may lead to abrupt of our CSI prediction when we implement our model to a specific area. To address this issue, we apply an online training process to strengthen the stability of our model. For

online training process, we keep records of our predictied CSI images. Meanwhile, we measure the related informations in the specific environment to build the real CSI images. Combining predicted and real CSI image together as a pair, we build a temporal dataset. For each time period t, we randomly choose a sequence of data in the temporal dataset, and calculate the gradients $\nabla \mathbf{W}$ of our model based on this sequence. Then, we updates the model weights as follow:

$$\mathbf{W}_c = \mathbf{W}_h - \beta \nabla \mathbf{W} \tag{38}$$

H. Parameter Settings for The Learning Framework

We set up 6 nodes in each case we mentioned in previous section as pixel of CSI image. In addition, for each CSI image, it contains 7 channels, which are CSI value in 2.4 GHz, CSI value in 5 GHz, location, time, temperature, humidity and whether.

Based on the size of CSI image, in our experiments, for 2D CNN, we utilize one 2×2 filter, and for 1D CNN, we utilize one 3×1 filter. The following LSTM network is a single LSTM layer with 64 nodes. The output of the LSTM network connected to two fully connected layers.

In offline training process, the values of the initial weights are set as random values between $[-0.1, 0.1]$, and we update them with RMSprop gradient descent algorithm. For online training process, we set the update time period as 5 minutes, and the $\beta = 0.1$.