Deep Transfer Learning Based Downlink Channel Prediction for FDD Massive MIMO Systems

Yuwen Yang, Feifei Gao, Zhimeng Zhong, Bo Ai and Ahmed Alkhateeb

***Abstract*—Artificial intelligence (AI) based downlink channel state information (CSI) prediction for frequency division duplexing (FDD) massive multiple-input multiple-output (MIMO) systems has attracted growing attention recently. However, existing works focus on the downlink CSI prediction for the users under a given environment and is hard to adapt to users in new environment especially when labeled data is limited. To address this issue, we formulate the downlink channel prediction as a deep transfer learning (DTL) problem, where each learning task aims to predict the downlink CSI from the uplink CSI for one single environment. Specifically, we develop the direct-transfer algorithm based on the fully-connected neural network architecture, where the network is trained on the data from all previous environments in the manner of classical deep learning and is then fine-tuned for new environments. To further improve the transfer efficiency, we propose the meta-learning algorithm that trains the network by alternating inner-task and across-task updates and then adapts to a new environment with a small number of labeled data. Simulation results show that the direct-transfer algorithm achieves better performance than the deep learning algorithm, which implies that the transfer learning benefits the downlink channel prediction in new environments. Moreover, the meta-learning algorithm significantly outperforms the direct-transfer algorithm in terms of both prediction accuracy and stability, especially when the number of samples is very small, which validates its effectiveness and superiority.**

***Index Terms*—Deep transfer learning (DTL), meta-learning, few-sample learning, downlink CSI prediction, FDD, massive MIMO**

I. Introduction

The acquisition of downlink channel state information (CSI) is a very challenging task for frequency division duplexing (FDD) massive multiple-input multiple-output (MIMO) systems due to the prohibitively high overheads associated with downlink training and uplink feedback [1]–[3]. By exploiting the angular and delay reciprocities between the uplink and the downlink [4]–[6], conventional methods proposed to reduce the downlink training overhead by extracting frequency-independent information from the uplink CSI, or to reduce the uplink feedback overhead by using compressive sensing based algorithms [7]–[10]. Nevertheless, the conventional methods either assume that the propagation paths are distinguishable and limited or highly rely on the sparsity of channels.

Recently, artificial intelligence (AI, including machine learning and deep learning, etc.) has been recognized as a potential solution to deal with the high complexity and overheads of wireless communication system [11]–[25]. Great success has been achieved in various applications such as channel estimation [12], [13], data detection [14], [15], CSI feedback [16], beamforming [17], [18], and hybrid precoding [19], [20], etc. Furthermore, AI based downlink CSI prediction for FDD systems has also been studied in many works [22]–[25]. In [22], a convolutional neural network is proposed to predict the downlink CSI from the uplink CSI for single-antenna FDD systems. Based on the CSI correlations between different base station (BS) antennas, linear and supper vector based regression methods have been proposed to use the downlink CSI of partial BS antennas to predict the whole downlink CSI, which can reduce the overheads of both downlink pilots and uplink feedback [23]. In fact, [24] develops the channel mapping in space and frequent concept which proves that deep neural networks (DNNs) can learn not only the correlation between closely-located BS antennas but also between the base station arrays that are positioned at different locations or different frequency bands in the same environment. By exploiting the mapping relation between the uplink and downlink CSI in FDD massive MIMO systems, an efficient complex-valued based DNN is trained to predict the downlink CSI only from the uplink CSI, i.e., no downlink pilot is needed at all [25].

Compared with conventional methods (e.g. [7], [8]), AI aided downlink CSI prediction benefits from the excellent learning capability of DNNs and does not require accurate channel models or high computational operations. However, existing AI aided downlink CSI prediction methods [22]–[25] all focus on training models for users in a certain environment. Since users may experience different wireless transmission environments, data collection and training the models from scratch are required for the users in new environments [22]–[25]. Typically thousands of samples and training epochs are required for training one DNN, and thus training a new DNN for each user would face unacceptable time and data cost. Therefore, it is highly desirable to design a method that can adapt to new environments with a small amount of data.

Inspired by human's capability to transfer knowledge from previous experience, transfer learning, which aims to improve the performance of target tasks by exploiting the knowledge from source tasks, becomes a promising technology in machine learning area to solve similar tasks with limited labeled data. Studies on transfer learning date back from 1995 in

Y. Yang and F. Gao are with Institute for Artificial Intelligence Tsinghua University (THUAI), State Key Lab of Intelligent Technologies and Systems, Beijing National Research Center for Information Science and Technology (BNRist), Department of Automation, Tsinghua University, Beijing, 100084, P. R. China (email: yyw18@mails.tsinghua.edu.cn, feifeigao@ieee.org).

Z. Zhong is with the Huawei Technologies Ltd., Shanghai 210206, China (e-mail: zhongzhimeng@huawei.com).

B. Ai is with the State Key Laboratory of Rail Traffic Control and Safety, Beijing Jiaotong University, Beijing 100044, China (email: boai@bjtu.edu.cn).

A. Alkhateeb is with the School of Electrical, Computer and Energy Engineering at Arizona State University, Tempe, AZ 85287, USA (e-mail: alkhateeb@asu.edu).

different names, such as incremental/cumulative learning, lifelong learning, multi-task learning, and learning to learn, etc [26]. Generally, transfer learning algorithms acquire knowledge from source tasks by pre-training a model on a large-scale source dataset and then fine-tune the pre-trained model on a small-scale target dataset[1]. With the rapid development of deep learning, the concept of deep transfer learning (DTL) is proposed by combining transfer learning with deep learning [28]. The methodology in transfer learning can be easily generalized to DTL, including instance-transfer, feature-representation-transfer, and parameter-transfer, etc [26]. In particular, the model-agnostic meta-learning algorithm proposed in [29], a typical parameter-transfer learning algorithm, is known for its universal applicability and the state-of-the-art performance. Unlike prior meta-learning algorithms [30]–[32] that learn an update function and impose restrictions on the model architecture, the model-agnostic meta-learning algorithm learns a model initialization that can effectively adapt to a new task with a small amount of labeled data.

In this paper, we formulate the downlink channel prediction for FDD massive MIMO systems as a DTL problem, where each learning task aims to predict the downlink CSI from the uplink CSI for users in a certain environment. Based on the fully-connected neural network (FNN) architecture, we develop the *direct-transfer algorithm*, where the network is trained on the data from all previous environments in the manner of classical deep learning and is then fine-tuned with limited data from a new environment. To achieve more effective transfer, we further proposed the *meta-learning algorithm* that learns a model initialization by an alternating training procedure, consisting of the inner-task and the across-task updates. In the inner-task updates, the meta-learning algorithm learns task-specific parameters by performing gradient descents on task-specific loss functions. In the across-task updates, the meta-learning algorithm update the network parameters by optimizing the loss function associated with multiple tasks. The proposed meta-learning algorithm can adapt to any new environment with a few number of labeled data. We also propose the *no-transfer algorithm* as a baseline algorithm, where the network is trained in the manner of classical deep learning and is directly tested on the data from a new environment without parameter adaption. Simulation results show that the direct-transfer algorithm outperforms the no-transfer algorithm, which validates that transfer learning can effectively improve the performance of downlink channel prediction in new environments. Moreover, the meta-learning algorithm significantly outperforms the direct-transfer algorithm, especially when the number of samples is very small, which demonstrates its superiority over the direct-transfer algorithm.

The remainder of this paper is organized as follows. The system model for FDD massive MIMO systems is introduced in Section II. The DTL problem is formulated in Section III. The no-transfer and direct-transfer algorithms are presented in Section IV. The meta-learning algorithm is developed in Section V. Numerical results are given in Section VI. Our main conclusions are given in Section VII.

[1]In unsupervised learning [27], labeled data in target tasks are not required. Unsupervised learning algorithms require strong assumptions of data distributions and are hard to be generalized to different problems, and therefore will not be discussed here.

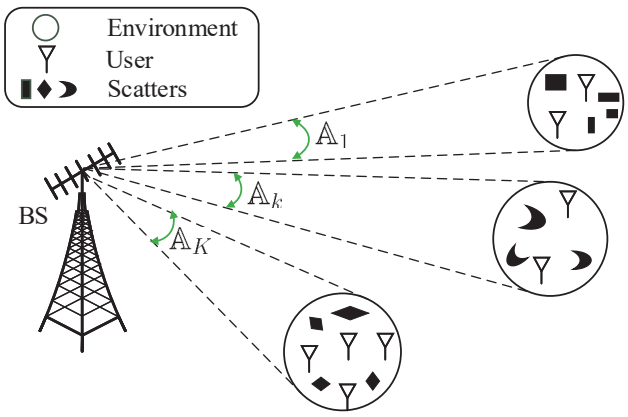

Fig. 1. Downlink CSI prediction for FDD Massive MIMO systems.

Notations: The bold and lowercase letters denote vectors while the bold and capital letters denote matrices. The notations $[\boldsymbol{z}]_p$ and $\text{len}(\boldsymbol{z})$ denote the p-th entry and the length of the vector $\boldsymbol{x}$, respectively. The notations $\Re[\cdot]$ and $\Im[\cdot]$, respectively, denote the real and imaginary parts of matrices, vectors or scales. The notation $\|\boldsymbol{x}\|_1$ denotes the L_1 norm of $\boldsymbol{x}$. The notation $(\cdot)^T$ denotes the transpose of a matrix or a vector. The notation $\mathbb{R}^{2M}$ represents the $2M$-dimensional real vector space. The notation $\circ$ represents the composite mapping operation. The notation $\mathcal{N}_C(\boldsymbol{0}, \boldsymbol{I})$ represents the standard complex Gaussian distribution. The notation $E[\cdot]$ represents the expectation with respect to all random variables within the brackets. The notation $\leftarrow$ represents the assignment operation while the notation $\rightarrow$ represents the direction of the trajectory.

II. System Model

We consider an FDD massive MIMO system, where BS is equipped with $M \gg 1$ antennas in the form of uniform linear array (ULA)[2] and serves multiple single-antenna users, as shown in Fig. 1. The users are randomly distributed in K regions and users in the same region share the same propagation environment. Denote $\mathbb{B}_k$ as the set of users in the k-th region. Then, the channel between the u-th ($u \in \mathbb{B}_k$) user and BS can be expressed as [8]:

$$\begin{aligned}\boldsymbol{h}_u(f) &= \int_{\theta\in\mathbb{A}_k} \alpha_u(\theta)\boldsymbol{a}(\theta)\, d\theta \\ &= \int_{\bar{\vartheta}_k^-}^{\bar{\vartheta}_k^+} |\alpha_u(\theta)|\, e^{-j2\pi f\tau_u(\theta)+j\phi_u(\theta)}\boldsymbol{a}(\theta)\, d\theta, \end{aligned} \quad (1)$$

where f is the carrier frequency of the u-th user, while $|\alpha_u(\theta)|$, $\phi_u(\theta)$ and $\tau_u(\theta)$ are the attenuation amplitude, the phase shift and the delay of the incident signal ray coming from direction of arrival (DOA) θ, respectively. Meanwhile,

[2]We adopt the ULA model here for simpler illustration, nevertheless, the proposed approach does not restrict to the specifical array shape, and therefore is applicable for array with arbitrary geometry.

$\mathbb{A}_k \triangleq \left[\bar{\vartheta}_k^-, \bar{\vartheta}_k^+\right]$ is the incident angle spread (AS) of the users in the k-th region with $\bar{\vartheta}_k^-$ and $\bar{\vartheta}_k^+$ being the corresponding lower and upper bounds of AS. The incident AS is assumed to be limited in a narrow region due to the limited local scattering effects at the BS side [4]–[6]. Moreover, $\boldsymbol{a}(\theta)$ is the the array manifold vector defined as

$$\boldsymbol{a}(\theta) = \left[1, e^{-j\varpi \sin\theta}, \cdots, e^{-j\varpi(M-1)\sin\theta}\right]^T, \quad (2)$$

where $\varpi = 2\pi df/c$, d is the antenna spacing, and c represents the speed of light.

III. Formulation of DTL problem

In this section, we first prove the feasibility of applying deep learning to predict the downlink CSI from uplink CSI for a single user. Then, we formulate the downlink channel prediction as a DTL problem and analyze the related transfer learning algorithms.

A. Deep Learning for Uplink-to-Downlink Mapping

Since the downlink and the uplink of a certain user share common physical paths and similar spatial propagation characteristics, there exists a deterministic mapping between the downlink and the uplink channels when the position-to-channel mapping is bijective (see more details referring to [24], [25]). Denote $f_{\mathrm{U},u}$ and $f_{\mathrm{D},u}$ as the uplink and the downlink frequencies of the u-th user, respectively. Then, the uplink-to-downlink mapping function can be written as

$$\boldsymbol{\Psi}_{f_{\mathrm{U},u}\to f_{\mathrm{D},u}} : \{\boldsymbol{h}_u(f_{\mathrm{U},u})\} \to \{\boldsymbol{h}_u(f_{\mathrm{D},u})\}. \quad (3)$$

Although the mapping function $\boldsymbol{\Psi}_{f_{\mathrm{U},u}\to f_{\mathrm{D},u}}$ cannot be described by known mathematical tools, it can be approximated by deep neural networks as shown in the following.

Let us consider the simplest three-layers FNN with only one input layer, one hidden layer with N neurons, and one output layer. Denote $\boldsymbol{x}$ and $\boldsymbol{\Omega}$ as the input data and the network parameter of FNN, respectively. Then the corresponding output can be expressed as $\mathrm{NET}_N(\boldsymbol{x}, \boldsymbol{\Omega})$. Since deep learning algorithms work in real domain, we introduce the isomorphism between the complex and real domains as

$$\boldsymbol{\xi} : \boldsymbol{z} \to \left(\Re\left(\boldsymbol{z}^T\right), \Im\left(\boldsymbol{z}^T\right)\right)^T. \quad (4)$$

Denote the inverse mapping of $\boldsymbol{\xi}$ as $\boldsymbol{\xi}^{-1}$ that can be given as

$$\boldsymbol{\xi}^{-1} : \left(\Re\left(\boldsymbol{z}^T\right), \Im\left(\boldsymbol{z}^T\right)\right)^T \to \boldsymbol{z}. \quad (5)$$

Based on the universal approximation theorem [33], we obtain the following theorem:

Theorem 1: For any given error $\varepsilon > 0$, there exists a positive constant N large enough such that

$$\sup_{\boldsymbol{x}\in\mathbb{H}} \left\|\mathrm{NET}_N(\boldsymbol{x}, \boldsymbol{\Omega}) - \boldsymbol{\Psi}_{\boldsymbol{\xi},u,\mathrm{U}\to\mathrm{D}}(\boldsymbol{x})\right\| \le \varepsilon, \quad \mathbb{H} = \{\boldsymbol{\xi}\circ\boldsymbol{h}_u(f_{\mathrm{U},u})\} \subseteq \mathbb{R}^{2M}, \quad (6)$$

with $\boldsymbol{\Psi}_{\boldsymbol{\xi},u,\mathrm{U}\to\mathrm{D}}(\boldsymbol{x}) = \boldsymbol{\xi}\circ\boldsymbol{\Psi}_{f_{\mathrm{U},u}\to f_{\mathrm{D},u}}\circ\boldsymbol{\xi}^{-1}(\boldsymbol{x})$.

Proof. (i) Since $\forall \boldsymbol{x} \in \mathbb{H}$, $|\boldsymbol{x}| = |\boldsymbol{h}_u(f_{\mathrm{U},u})| \le \sqrt{M}\vartheta_u |\alpha_u(\theta)|$, we know $\boldsymbol{x}$ is bounded. Therefore, $\mathbb{H}$ is a compact set; (ii) Since $\boldsymbol{\Psi}_{f_{\mathrm{U},u}\to f_{\mathrm{D},u}}$ and $\boldsymbol{\xi}$ are continuous mapping and composition of continuous mappings is still a continuous mapping, we know $\boldsymbol{\Psi}_{\boldsymbol{\xi},u,\mathrm{U}\to\mathrm{D}}(\boldsymbol{x})$ is a continuous function for $\forall \boldsymbol{x} \in \mathbb{H}$. Therefore, the i-th element of $\boldsymbol{\Psi}_{\boldsymbol{\xi},u,\mathrm{U}\to\mathrm{D}}(\boldsymbol{x})$, denoted by $[\boldsymbol{\Psi}_{\boldsymbol{\xi},u,\mathrm{U}\to\mathrm{D}}(\boldsymbol{x})]_i$, is also a continuous function of $\boldsymbol{x}$. Based on (i), (ii) and universal approximation theorem [33, Theorem 1], we know for any $\varepsilon_i > 0$, there exists a positive constant N_i such that

$$\sup_{x\in\mathbb{H}} \left|\mathrm{NET}_{N_i}(\boldsymbol{x}, \boldsymbol{\Omega}_i) - [\boldsymbol{\Psi}_{\boldsymbol{\xi},u,\mathrm{U}\to\mathrm{D}}(\boldsymbol{x})]_i\right| \le \varepsilon_i, \quad (7)$$

where $\mathrm{NET}_{N_i}(\boldsymbol{x}, \boldsymbol{\Omega}_i)$ is the output of the network with only one neuron in the output layer, and $\boldsymbol{\Omega}_i$ denotes the corresponding network parameters. By stacking $2M$ of the above networks together, we can construct a larger network $\mathrm{NET}_N(\boldsymbol{x}, \boldsymbol{\Omega})$ with $N = \sum_{i=1}^{2M} N_i$, where $\boldsymbol{\Omega}$ denotes the corresponding network parameters. By choosing $\varepsilon_i = \varepsilon/\sqrt{2M}$, Eq. (6) can be proved. □

Theorem 1 reveals that the uplink-to-downlink mapping function can be approximated arbitrarily well by an FNN with a single hidden layer. It should be mentioned that although the network proposed in this work cannot obtain arbitrarily high prediction accuracy due to inadequate learning or insufficient number of hidden units [33], *Theorem 1* still provides theoretical foundation for the application of deep neural networks.

B. Definitions of DTL

For the users in the k-th environment (region), let $\mathcal{X}_k$ and $\mathcal{Y}_k$ denote the space[3] of the uplink and the downlink channels, respectively. Then, the definitions of the "domain" and the "task" are given in the following two definitions:

Definition 1: The "domain" $\mathcal{D}(k)$ is composed of the feature space $\mathcal{X}_k$ and the marginal probability distribution $P(\boldsymbol{h}_u(f_{\mathrm{U},u}))|_{u\in\mathbb{B}_k}$, i.e., $\mathcal{D}(k) = \{\mathcal{X}_k, P(\boldsymbol{h}_u(f_{\mathrm{U},u}))|_{u\in\mathbb{B}_k}\}$.

Definition 2: Define the "task" $\mathcal{T}(k)$ as the prediction of the downlink channels from the uplink channels for the users in the k-th environment. Given the specific domain $\mathcal{D}(k)$, the "task" $\mathcal{T}(k)$ is composed of the label space $\mathcal{Y}_k$ and the prediction function $\mathcal{F}_k$, i.e., $\mathcal{T}(k) = \{\mathcal{Y}_k, \mathcal{F}_k\}$.

Remark 1: The prediction function $\mathcal{F}_k$ can be learned from the training data of the k-th environment and then be used to predict the downlink channels for the users in the k-th environment. Based on the analysis in Section III-A, the prediction function $\mathcal{F}_k$ can be interpreted as the fitting function for all the uplink-to-downlink mapping functions defined in the k-th environment, i.e., $\left\{\boldsymbol{\Psi}_{f_{\mathrm{U},u}\to f_{\mathrm{D},u}}\right\}_{u\in\mathbb{B}_k}$. The prediction function $\mathcal{F}_k$ can also be interpreted as the conditional probability distribution $P(\boldsymbol{h}_u(f_{\mathrm{D},u})|\boldsymbol{h}_u(f_{\mathrm{U},u}))|_{u\in\mathbb{B}_k}$ from probabilistic view.

Classical transfer learning consists of two aspects, namely, the source domain transfer and the target domain adaption. Based on [26], the definition of transfer learning can be given as following:

[3] The space of the uplink channels is $\mathcal{X}$ means that any possible uplink channel vector belongs to $\mathcal{X}$, i.e. $\boldsymbol{h}_u(f_{\mathrm{U},u}) \in \mathcal{X}$ holds for any possible u or $f_{\mathrm{U},u}$.

Definition 3: Given the source task $\mathcal{T}_{\mathrm{S}}$, the source domain $\mathcal{D}_{\mathrm{S}}$, the target task $\mathcal{T}_{\mathrm{T}}$, and the target domain $\mathcal{D}_{\mathrm{T}}$, the aim of transfer learning is to improve the performance of the target task $\mathcal{T}_{\mathrm{T}}$ by using the knowledge from $\mathcal{T}_{\mathrm{S}}$ and $\mathcal{D}_{\mathrm{T}}$, where $\mathcal{D}_{\mathrm{T}} \neq \mathcal{D}_{\mathrm{S}}$ or $\mathcal{T}_{\mathrm{T}} \neq \mathcal{T}_{\mathrm{S}}$.

Here we extend the single-source domain transfer to the multi-source domain transfer. Then, a more generalized definition of transfer learning can be provided as follows:

Definition 4: Given the number of source tasks K_{s}, the source tasks $\{\mathcal{T}_{\mathrm{S}}(k)\}_{k=1}^{K_{\mathrm{s}}}$, the source domains $\{\mathcal{D}_{\mathrm{S}}(k)\}_{k=1}^{K_{\mathrm{s}}}$, the target task $\mathcal{T}_{\mathrm{T}}$, and the target domain $\mathcal{D}_{\mathrm{T}}$, the aim of transfer learning is to improve the performance of the target task $\mathcal{T}_{\mathrm{T}}$ by using the knowledge from $\{\mathcal{T}_{\mathrm{S}}(k)\}_{k=1}^{K_{\mathrm{s}}}$ and $\{\mathcal{D}_{\mathrm{S}}(k)\}_{k=1}^{K_{\mathrm{s}}}$, where $\mathcal{D}_{\mathrm{T}} \neq \mathcal{D}_{\mathrm{S}}(k)$ or $\mathcal{T}_{\mathrm{T}} \neq \mathcal{T}_{\mathrm{S}}(k)$ holds for $k = 1, \cdots, K$.

In *Definition 4*, the condition $\mathcal{D}_{\mathrm{T}} \neq \mathcal{D}_{\mathrm{S}}(k)$ means that either the corresponding feature space $\mathcal{X}_{\mathrm{T}} \neq \mathcal{X}_{\mathrm{S}}(k)$ holds or the corresponding marginal probability distribution $P_{\mathrm{T}}\left(\boldsymbol{h}_u(f_{\mathrm{U},u})\right)|_{u\in\mathbb{B}_{\mathrm{T}}} \neq P_{\mathrm{S}(k)}\left(\boldsymbol{h}_u(f_{\mathrm{U},u})\right)|_{u\in\mathbb{B}_k}$ holds, where $\mathbb{B}_{\mathrm{T}}$ is the set of users in the target environment. The condition $\mathcal{T}_{\mathrm{T}} \neq \mathcal{T}_{\mathrm{S}}(k)$ means that either the corresponding label space $\mathcal{Y}_{\mathrm{T}} \neq \mathcal{Y}_{\mathrm{S}}(k)$ holds or the corresponding conditional probability distribution $P_{\mathrm{T}}\left(\boldsymbol{h}_u(f_{\mathrm{D},u}|\boldsymbol{h}_u(f_{\mathrm{U},u}))\right)|_{u\in\mathbb{B}_{\mathrm{T}}} \neq P_{\mathrm{S}(k)}\left(\boldsymbol{h}_u(f_{\mathrm{D},u})|\boldsymbol{h}_u(f_{\mathrm{U},u})\right)|_{u\in\mathbb{B}_k}$ holds. Since the conditional probability distributions for different prediction tasks are different, the condition $\mathcal{T}_{\mathrm{T}} \neq \mathcal{T}_{\mathrm{S}}(k)$ is satisfied. Therefore, the downlink channel prediction for massive MIMO systems can be formulated as a transfer learning problem.

DTL is to transfer knowledge by deep neural networks, which is defined as follows [28]:

Definition 5: Given a transfer learning task characterized by $\left\langle \{\mathcal{T}_{\mathrm{S}}(k)\}_{n=1}^{K_{\mathrm{s}}}, \{\mathcal{D}_{\mathrm{S}}(k)\}_{k=1}^{K_{\mathrm{s}}} \mathcal{T}_{\mathrm{T}}, \mathcal{D}_{\mathrm{T}} \right\rangle$, it is a DTL task when the prediction function $\mathcal{F}_{\mathrm{T}}$ of $\mathcal{T}_{\mathrm{T}}$ is a non-linear function that is approximated by a deep neural network.

Based on *Definition 4* and *Definition 5*, the downlink channel prediction for massive MIMO systems can be formulated as a typical DTL problem, where the k-th learning task is to predict the downlink channel $\boldsymbol{h}_u(f_{\mathrm{D},u})$ from the uplink channel $\boldsymbol{h}_u(f_{\mathrm{U},u})$ for the users in the k-th environment.

C. Motivation of Meta-learning

Before resorting to DTL methods, we should first consider the question "whether to transfer". In fact, if the source and target tasks are highly related, a deep neural network would perform well, without the need for fine-tuning the neural network based on the target environment, thanks to its remarkable generalization capability.

The second question is "how to transfer". One natural solution is to directly use all the labeled data in the source tasks to train the network, and then fine-tune the network with the labeled data in the target task. However, the direct transfer method tends to overfit when the number of labeled data in the target task is small, as will be verified by simulations in Section VI. To overcome this challenge, and motivated by the state-of-the-art performance of the model-agnostic meta-learning algorithm in few-sample learning [29], we will propose the meta-learning algorithm based downlink channel prediction in Section V.

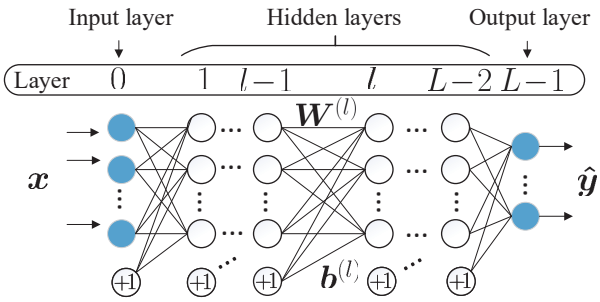

Fig. 2. The FNN architecture.

IV. No-Transfer and Direct-Transfer Algorithms

Based on the analysis in Section III-C, we propose the no-transfer algorithm based on classical deep learning algorithms to investigate the necessity of transfer learning. Then, the direct-transfer algorithm is proposed and used as the benchmark of the meta-learning algorithm. Both the no-transfer and the direct-transfer algorithms adopt the FNN architecture as described in the following subsection.

A. Network Architecture

The FNN architecture has L layers, including one input layer, $L-2$ hidden layers, and one output layer, as shown in Fig. 2. The input of FNN is $\boldsymbol{x} = \boldsymbol{\xi} \circ \boldsymbol{h}_u(f_{\mathrm{U},u})$. The output of the network is a cascade of nonlinear transformation of $\boldsymbol{x}$, i.e.,

$$\hat{\boldsymbol{y}} = \mathrm{Net}\left(\boldsymbol{x}, \boldsymbol{\Omega}\right) = \boldsymbol{f}^{(L-1)} \circ \boldsymbol{f}^{(L-2)} \circ \cdots \boldsymbol{f}^{(1)}\left(\boldsymbol{x}\right), \quad (8)$$

where $\boldsymbol{\Omega} \triangleq \left\{\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)}\right\}_{l=1}^{L-1}$ is the network parameters to be trained. Moreover, $\boldsymbol{f}^{(l)}$ is the nonlinear transformation function of the l-th layer and can be written as,

$$\boldsymbol{f}^{(l)}\left(\boldsymbol{x}\right) = \boldsymbol{r}^{(l)}\left(\boldsymbol{W}^{(l)}\boldsymbol{x} + \boldsymbol{b}^{(l)}\right), \ 1 \leq l \leq L-1, \quad (9)$$

where $\boldsymbol{W}^{(l)}$ is the weight matrix associated with the $(l-1)$-th and the l-th layers, while $\boldsymbol{b}^{(l)}$ and $\boldsymbol{r}^{(l)}$ are the bias vector and the activation function of the l-th layer, respectively. The activation function for the hidden layers is selected as the rectified linear unit (ReLU) function $\left[\boldsymbol{r}_{\mathrm{re}}(\boldsymbol{z})\right]_p = \max\{0, [\boldsymbol{z}]_p\}$ with $p = 1, 2, \cdots, \mathrm{len}(\boldsymbol{z})$. The activation function for the output layer is the linear function, i.e., $\boldsymbol{r}_{\mathrm{li}}(\boldsymbol{z}) = \boldsymbol{z}$. The loss function can be written as follow:

$$\mathrm{Loss}_{\mathbb{D}}\left(\boldsymbol{\Omega}\right) = \frac{1}{V}\sum_{v=0}^{V-1}\left\|\hat{\boldsymbol{y}}^{(v)} - \boldsymbol{y}^{(v)}\right\|_2^2, \quad (10)$$

where $\boldsymbol{y} = \boldsymbol{\xi} \circ \boldsymbol{h}_u(f_{\mathrm{D},u})$ is the supervise label, V is the batch size[4], the superscript (v) denotes the index of the v-th training sample, $\mathbb{D} = \left\{(\boldsymbol{x}^{(v)}, \boldsymbol{y}^{(v)})\right\}_{v=1}^{V}$ is the training dataset, and $\|\cdot\|_2$ denotes the L_2 norm.

[4]Batch size is the number of samples in one training batch.

B. Definitions and Generation of Datasets

As mentioned in *Definition 2*, each task represents the prediction of downlink channel from the uplink channel for the users in a certain environment. Denote Δf as the frequency difference between the uplink and the downlink. Then, each uplink frequency corresponds to a certain downlink frequency, i.e., $f_{\mathrm{D},u} = f_{\mathrm{U},u} + \Delta f$. Denote U as the number of users in the k-th environment that are involved in the generation of datasets. Denote the training dataset of the k-th source task as $\mathbb{D}_{\mathrm{Tr}}(k)$. To generate $\mathbb{D}_{\mathrm{Tr}}(k)$, N_{Tr} sample pairs $\{(\boldsymbol{\xi} \circ \boldsymbol{h}_u(f_{\mathrm{D},u}), \boldsymbol{\xi} \circ \boldsymbol{h}_u(f_{\mathrm{U},u}))\}$ are collected by randomly selecting N_{Tr}/U uplink frequencies for each user in the k-th environment.

Denote the adaption and testing datasets of the k-th target task as $\mathbb{D}_{\mathrm{Ad}}(k)$ and $\mathbb{D}_{\mathrm{Te}}(k)$, respectively. Similarly, the datasets $\mathbb{D}_{\mathrm{Ad}}(k)$ and $\mathbb{D}_{\mathrm{Te}}(k)$ can be obtained by separately collecting N_{Ad} and N_{Te} sample pairs for the k-th target task. Note that $\mathbb{D}_{\mathrm{Ad}}(k) \cap \mathbb{D}_{\mathrm{Ad}}(k) = \emptyset$ should be satisfied to ensure the testing dataset not known to the networks.

C. No-Transfer Algorithm

The no-transfer algorithm regards the DTL problem as one classical deep learning problem, including the training and the testing stages. The datasets $\{\mathbb{D}_{\mathrm{Tr}}(k)\}_{k=1}^{K_{\mathrm{S}}}$ for K_{S} source tasks are generated and are used to train the network. To avoid overfitting, we randomly shuffle all the samples in datasets $\{\mathbb{D}_{\mathrm{Tr}}(k)\}_{k=1}^{K_{\mathrm{S}}}$ and obtain the shuffled training dataset $\mathbb{D}_{\mathrm{STr}}$.

In each time step, V training samples are randomly selected as the training batch $\mathbb{D}_{\mathrm{STrB}}$. Then, we employ the adaptive moment estimation (ADAM) algorithm [34] to minimize the loss function on $\mathbb{D}_{\mathrm{STrB}}$, i.e., $\mathrm{Loss}_{\mathbb{D}_{\mathrm{STrB}}}(\boldsymbol{\Omega})$. Instead of using the gradient of the current step to guide the updates, ADAM adopts moments of the gradient to determine the direction of the optimization, which can accelerate the training, especially in the case of high curvature or noisy gradients [35, chapter 8]. The q-th moment of a random variable is defined as the expected value of the q-th power of the variable. To estimate the 1st and the 2nd moment vectors of the gradient, i.e., $\nabla_{\boldsymbol{\Omega}}\mathrm{Loss}_{\mathbb{D}_{\mathrm{STrB}}}(\boldsymbol{\Omega})$, ADAM computes the exponentially decaying moving averages of past gradients and elementwise squared gradients, i.e.,

$$\begin{cases} \boldsymbol{\mu}_1 & \leftarrow \rho_1 \boldsymbol{\mu}_1 + (1-\rho_1)\nabla_{\boldsymbol{\Omega}}\mathrm{Loss}_{\mathbb{D}_{\mathrm{STrB}}}(\boldsymbol{\Omega}), \\ \boldsymbol{\nu}_1 & \leftarrow \rho_2 \boldsymbol{\nu}_1 + (1-\rho_2)\left(\nabla_{\boldsymbol{\Omega}}\mathrm{Loss}_{\mathbb{D}_{\mathrm{STrB}}}(\boldsymbol{\Omega})\right)^2, \end{cases} \tag{11}$$

where $\left(\nabla_{\boldsymbol{\Omega}}\mathrm{Loss}_{\mathbb{D}_{\mathrm{STrB}}}(\boldsymbol{\Omega})\right)^2$ represents the elementwise square of the gradient, and the hyper-parameters $\rho_1, \rho_2 \in [0, 1)$ are the exponential decay rates of the moving averages. The vectors $\boldsymbol{\mu}_1$ and $\boldsymbol{\nu}_1$ are initialized with zeros and updated with moving averages, which results in biased moment estimates of the gradient. Following [34], unbiased 1st and 2nd moment vectors can be obtained by

$$\begin{cases} \bar{\boldsymbol{\mu}}_1 & \leftarrow \boldsymbol{\mu}_1/(1-\rho_1^t), \\ \bar{\boldsymbol{\nu}}_1 & \leftarrow \boldsymbol{\nu}_1/(1-\rho_2^t), \end{cases} \tag{12}$$

where t represents the time step. Then, the unbiased 1st and 2nd moment vectors, i.e., $\bar{\boldsymbol{\mu}}_1$ and $\bar{\boldsymbol{\nu}}_1$, are used to scale the learning rate individually for all elements in $\boldsymbol{\Omega}$, which can be expressed as

$$\boldsymbol{\Omega} \leftarrow \boldsymbol{\Omega} - \eta\bar{\boldsymbol{\mu}}_1/(\sqrt{\bar{\boldsymbol{\nu}}_1} + \varepsilon), \tag{13}$$

where η is the learning rate, ε is the disturbance factor, and the notation "/" is the elementwise division operation.

By iteratively executing the ADAM algorithm on the dataset $\mathbb{D}_{\mathrm{STr}}$ until $\mathrm{Loss}_{\mathbb{D}_{\mathrm{STrB}}}(\boldsymbol{\Omega})$ converges, we can obtain the trained network parameter $\boldsymbol{\Omega}_{\mathrm{Nt}}$. In the testing stage, the network parameter $\boldsymbol{\Omega}_{\mathrm{Nt}}$ is fixed. The testing datasets $\{\mathbb{D}_{\mathrm{Te}}(k)\}_{k=1}^{K_{\mathrm{T}}}$ of K_{T} target tasks are generated and are used to test the performance of the no-transfer algorithm. Denote $\hat{\boldsymbol{h}}_{\mathrm{D}} = \boldsymbol{\xi}^{-1} \circ \hat{\boldsymbol{y}}$ and $\boldsymbol{h}_{\mathrm{D}} = \boldsymbol{\xi}^{-1} \circ \boldsymbol{y}$ as the estimated and the true downlink channel, respectively. Normalized mean-squared-error (NMSE) is used to measure the prediction accuracy, which is defined as

$$\mathrm{NMSE} = E\left[\left\|\boldsymbol{h}_{\mathrm{D}} - \hat{\boldsymbol{h}}_{\mathrm{D}}\right\|_2^2 / \left\|\boldsymbol{h}_{\mathrm{D}}\right\|_2^2\right]. \tag{14}$$

Denote the prediction NMSE of the no-transfer algorithm as $\mathrm{NMSE}_{\mathrm{Nt}}$ that can be calculated by averaging the NMSEs of the no-transfer algorithm evaluated on K_{T} target environments. i.e., $\mathrm{NMSE}_{\mathrm{Nt}} = \sum_{\mathrm{k}=1}^{\mathrm{K_T}} \mathrm{NMSE}_{\mathrm{Nt}}(\mathrm{k})/\mathrm{K_T}$, where $\mathrm{NMSE}_{\mathrm{Nt}}(\mathrm{k})$ is the NMSE evaluated on the k-th target environment. The concrete steps of the no-transfer algorithm are given in the **Algorithm 1**.

D. Direct Transfer Algorithm

In the training stage, the direct-transfer algorithm trains the network via the ADAM algorithm on the training dataset $\mathbb{D}_{\mathrm{STr}}$ until $\mathrm{Loss}_{\mathbb{D}_{\mathrm{STrB}}}(\boldsymbol{\Omega})$ converges. During the training, the network tries to learn a generalized parameter to predict the downlink CSI for users in different environments. After the training finished, the network has learned the parameter $\boldsymbol{\Omega}_{\mathrm{Nt}}$ from the K_{S} source tasks. Then, the datasets $\{\mathbb{D}_{\mathrm{Ad}}(k)\}_{k-1}^{k=K_{\mathrm{T}}}$ and $\{\mathbb{D}_{\mathrm{Te}}(k)\}_{k-1}^{k=K_{\mathrm{T}}}$ of K_{T} target tasks are generated following Section IV-B. For the k-th target task, the direct-transfer algorithm initializes the target-task-specific parameter $\boldsymbol{\Omega}_{\mathrm{T},k}$ as the trained network parameter $\boldsymbol{\Omega}_{\mathrm{Nt}}$, then fine-tunes $\boldsymbol{\Omega}_{\mathrm{T},k}$ on the adaption dataset $\mathbb{D}_{\mathrm{Ad}}(k)$ via G_{Ad} steps of ADAM updates, and then tests the network on the testing dataset $\mathbb{D}_{\mathrm{Te}}(k)$. Specifically, ADAM aims to minimize $\mathrm{Loss}_{\mathbb{D}_{\mathrm{Ad}}(k)}(\boldsymbol{\Omega}_{\mathrm{T},k})$ by updating the parameter $\boldsymbol{\Omega}_{\mathrm{T},k}$, and unbiased moment vectors of the gradient $\nabla_{\boldsymbol{\Omega}_{\mathrm{T},k}}\mathrm{Loss}_{\mathbb{D}_{\mathrm{Ad}}(k)}(\boldsymbol{\Omega}_{\mathrm{T},k})$ are used to determine the direction of the update. Based on the derivations in Section IV-C, biased 1st and 2nd moment vectors are initialized with zeros and are then updated in each step following

$$\begin{cases} \boldsymbol{\mu}_2 & \leftarrow \rho_1 \boldsymbol{\mu}_2 + (1-\rho_1)\nabla_{\boldsymbol{\Omega}_{\mathrm{T},k}}\mathrm{Loss}_{\mathbb{D}_{\mathrm{Ad}}(k)}(\boldsymbol{\Omega}_{\mathrm{T},k}) \\ \boldsymbol{\nu}_2 & \leftarrow \rho_2 \boldsymbol{\nu}_2 + (1-\rho_2)\left(\nabla_{\boldsymbol{\Omega}_{\mathrm{T},k}}\mathrm{Loss}_{\mathbb{D}_{\mathrm{Ad}}(k)}(\boldsymbol{\Omega}_{\mathrm{T},k})\right)^2. \end{cases} \tag{15}$$

Next, we obtain the unbiased 1st and 2nd moment vectors by

$$\begin{cases} \bar{\boldsymbol{\mu}}_2 & \leftarrow \boldsymbol{\mu}_2/(1-\rho_1^g), \\ \bar{\boldsymbol{\nu}}_2 & \leftarrow \boldsymbol{\nu}_2/(1-\rho_2^g), \end{cases} \tag{16}$$

where g represents the step index. The update of $\boldsymbol{\Omega}_{\mathrm{T},k}$ can be written as

$$\boldsymbol{\Omega}_{\mathrm{T},k} \leftarrow \boldsymbol{\Omega}_{\mathrm{T},k} - \eta\bar{\boldsymbol{\mu}}_2/(\sqrt{\bar{\boldsymbol{\nu}}_2} + \varepsilon). \tag{17}$$

Algorithm 1: No-transfer algorithm for downlink CSI prediction

Input: Source tasks: $\{\mathcal{T}_{\mathrm{S}}(k)\}_{k=1}^{K_{\mathrm{S}}}$, Target tasks: $\{\mathcal{T}_{\mathrm{T}}(k)\}_{k=1}^{K_{\mathrm{T}}}$, learning rate: η, exponential decay rates: $\{\rho_1, \rho_2\}$, disturbance factor: ε, batch size: V

Output: Trained network parameter: $\mathbf{\Omega}_{\mathrm{Nt}} \leftarrow \mathbf{\Omega}$, predicted downlink CSI based on $\{\mathbb{D}_{\mathrm{Te}}(k)\}_{k=1}^{K_{\mathrm{T}}}$, NMSE of the no-transfer algorithm: $\mathrm{NMSE}_{\mathrm{Nt}}$

```
Training stage
```
Randomly initialize the network parameters $\mathbf{\Omega}$
Initialize the time step: $t \leftarrow 0$
Initialize the 1st and 2nd moment vectors: $\boldsymbol{\mu}_1, \boldsymbol{\nu}_1 \leftarrow \mathbf{0}$
Generate the training dataset $\mathbb{D}_{\mathrm{Tr}}(k)$ for each source task
Randomly shuffle training datasets $\{\mathbb{D}_{\mathrm{Tr}}(k)\}_{k=1}^{K_{\mathrm{S}}}$ and obtain the shuffled training datasets $\mathbb{D}_{\mathrm{STr}}$
while *not done* **do**
  Randomly select V training samples from $\mathbb{D}_{\mathrm{STr}}$ as the training batch $\mathbb{D}_{\mathrm{STrB}}$
  $t \leftarrow t + 1$
  Update biased 1st and 2nd moment vectors using Eq. (11)
  Compute unbiased 1st and 2nd moment vectors using Eq. (12)
  Update the parameter $\mathbf{\Omega}$ using unbiased 1st and 2nd moment vectors: $\mathbf{\Omega} \leftarrow \mathbf{\Omega} - \eta\bar{\boldsymbol{\mu}}_1/(\sqrt{\bar{\boldsymbol{\nu}}_1} + \varepsilon)$
end
```
Testing stage
```
Initialize NMSE: $\mathrm{NMSE}_{\mathrm{Nt}} \leftarrow 0$
for $k = 1, \cdots, K_{\mathrm{T}}$ **do**
  Generate the testing dataset $\mathbb{D}_{\mathrm{Te}}(k)$ for $\mathcal{T}_{\mathrm{T}}(k)$
  Predict the downlink CSI base on $\mathbb{D}_{\mathrm{Te}}(k)$ and $\mathbf{\Omega}$ using Eq. (8)
  Calculate $\mathrm{NMSE}_{\mathrm{Nt}}(\mathrm{k})$ using Eq. (14)
  $\mathrm{NMSE}_{\mathrm{Nt}} \leftarrow \mathrm{NMSE}_{\mathrm{Nt}} + \mathrm{NMSE}_{\mathrm{Nt}}(\mathrm{k})/\mathrm{K_T}$
end

Algorithm 2: Direct-transfer algorithm for downlink CSI prediction

Input: Source tasks: $\{\mathcal{T}_{\mathrm{S}}(k)\}_{k=1}^{K_{\mathrm{S}}}$, Target tasks: $\{\mathcal{T}_{\mathrm{T}}(k)\}_{k=1}^{K_{\mathrm{T}}}$, learning rate: η, exponential decay rates: $\{\rho_1, \rho_2\}$, disturbance factor: ε, batch size: V, number of gradsteps for adaption: G_{Ad}

Output: Trained network parameter: $\mathbf{\Omega}_{\mathrm{Nt}}$, predicted downlink CSI based on $\{\mathbb{D}_{\mathrm{Te}}(k)\}_{k=1}^{K_{\mathrm{T}}}$, NMSE of the direct-transfer algorithm: $\mathrm{NMSE}_{\mathrm{Dt}}$

```
Training stage
```
Implement the **Algorithm 1** and obtain the trained network parameter $\mathbf{\Omega}_{\mathrm{Nt}}$
```
Adaption and Testing for target tasks
```
Initialize NMSE: $\mathrm{NMSE}_{\mathrm{Dt}} \leftarrow 0$
for $k = 1, \cdots, K_{\mathrm{T}}$ **do**
  Generate the datasets $\mathbb{D}_{\mathrm{Ad}}(k)$ and $\mathbb{D}_{\mathrm{Te}}(k)$ for $\mathcal{T}_{\mathrm{T}}(k)$
  ```
  Adaption stage
  ```
  Load the network parameter $\mathbf{\Omega}_{\mathrm{T},k} \leftarrow \mathbf{\Omega}_{\mathrm{Nt}}$
  Initialize the 1st and 2nd moment vectors: $\boldsymbol{\mu}_2, \boldsymbol{\nu}_2 \leftarrow \mathbf{0}$
  for $g = 1, \cdots, G_{\mathrm{Ad}}$ **do**
    Update biased 1st and 2nd moment vectors using Eq. (15)
    Compute unbiased 1st and 2nd moment vectors using Eq. (16)
    Update the parameter $\mathbf{\Omega}_{\mathrm{T},k}$ using unbiased moment vectors: $\mathbf{\Omega}_{\mathrm{T},k} \leftarrow \mathbf{\Omega}_{\mathrm{T},k} - \eta\bar{\boldsymbol{\mu}}_2/(\sqrt{\bar{\boldsymbol{\nu}}_2} + \varepsilon)$
  end
  ```
  Testing stage
  ```
  Predict the downlink CSI base on $\mathbb{D}_{\mathrm{Te}}(k)$ and $\mathbf{\Omega}_{\mathrm{T},k}$ using Eq. (8)
  Calculate $\mathrm{NMSE}_{\mathrm{Dt}}(\mathrm{k})$ using Eq. (14)
  $\mathrm{NMSE}_{\mathrm{Dt}} \leftarrow \mathrm{NMSE}_{\mathrm{Dt}} + \mathrm{NMSE}_{\mathrm{Dt}}(\mathrm{k})/\mathrm{K_T}$
end

After the fine-tuning finished, the target-task-specific parameter $\mathbf{\Omega}_{\mathrm{T},k}$ will be fixed. Let $\mathrm{NMSE}_{\mathrm{Dt}}(\mathrm{k})$ represent the NMSE of the direct-transfer algorithm evaluated on the k-th target environment that can be obtained by testing the network on the dataset $\mathbb{D}_{\mathrm{Te}}(k)$ using Eqs. (8) and (14). Then, the prediction NMSE of the direct-transfer algorithm can be obtained by $\mathrm{NMSE}_{\mathrm{Dt}} = \sum_{\mathrm{k}=1}^{\mathrm{K_T}} \mathrm{NMSE}_{\mathrm{Dt}}(\mathrm{k})/\mathrm{K_T}$. The concrete steps of the direct-transfer algorithm are given in the **Algorithm 2**.

Notice that the direct-transfer algorithm utilizes the trained network parameter $\mathbf{\Omega}_{\mathrm{Nt}}$ as the initialization of the adaption stage for every target task. By contrast, another way is to utilize previous target-task-specific parameter $\mathbf{\Omega}_{\mathrm{T},k-1}$ as the initialization of the adaption stage for the k-th target task since $\mathbf{\Omega}_{\mathrm{T},k-1}$ appears to contain the information in all the source environments and $k-1$ target environments, and therefore could enhance the performance of the algorithm. However, the real situation is that a good initialization in DTL is not to train the network parameter with as much data as possible, but to find such a parameter that can easily adapt to any target task. In a way, a good initial parameter can be interpreted as a parameter point in the parameter space that is near (or easy to get) to the optimal parameter point for most target tasks. Fig. 3 illustrates the trajectory of parameter updating during the training and adaption stages of the direct-transfer algorithm. The black solid line represents the updating trajectory of the network parameter $\mathbf{\Omega}$ during the training stage. The blue dashed lines represent the updating trajectories of the target-task-specific parameters $\{\mathbf{\Omega}_{\mathrm{T},k}\}_{k=1}^{3}$ during the adaption stage following **Algorithm 2**. The red dash-dotted lines represent the parameter adaption process with previous target-task-specific parameter as the initial parameter. As shown in Fig. 3, the trajectory of red dash-dotted lines, i.e., $\mathbf{\Omega}_{\mathrm{Nt}} \rightarrow \mathbf{\Omega}_{\mathrm{T},1} \rightarrow \mathbf{\Omega}_{\mathrm{T},2} \rightarrow \mathbf{\Omega}_{\mathrm{T},3}$, is longer than the trajectory of blue dashed lines, i.e., $\{\mathbf{\Omega}_{\mathrm{Nt}} \rightarrow \mathbf{\Omega}_{\mathrm{T},k}\}_{k=1}^{3}$, which indicates $\mathbf{\Omega}_{\mathrm{Nt}}$ is a better initialization than $\mathbf{\Omega}_{\mathrm{T},k-1}$. The intrinsic reason is that $\mathbf{\Omega}_{\mathrm{Nt}}$ is obtained by training the network on a large-scale and shuffled dataset, i.e., $\mathbb{D}_{\mathrm{STr}}$, which has more stronger representativeness of different tasks than the small-scale adaption dataset $\mathbb{D}_{\mathrm{Ad}}(k)$. In fact, the fine-tuning on the adaption dataset $\mathbb{D}_{\mathrm{Ad}}(k)$ renders the network parameter away from the generalized parameter

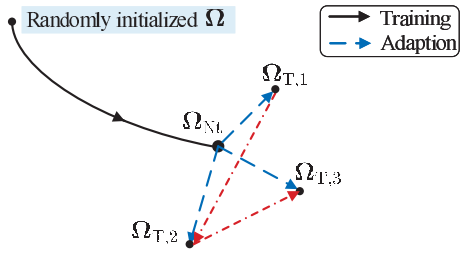

Fig. 3. Illustration of the direct-transfer algorithm, where $K_{\mathrm{T}} = 3$.

$\boldsymbol{\Omega}_{\mathrm{Nt}}$ and gets closed to the task-specific parameter $\boldsymbol{\Omega}_{\mathrm{T},k}$.

Remark 2: It should be emphasized that parameter initialization is one of the most crucial tasks in deep learning, especially in DTL. A high-quality initialization can effectively prevent the training from getting stuck in a low-performance local minimum and thus accelerates the convergence [36], [37].

V. Meta-learning Algorithm

The proposed meta-learning algorithm also adopts the FNN architecture in Section IV-A and has three stages, namely, the meta-training, the meta-adaption and the testing stages.

A. Definitions and Generation of Datasets

For the k-th source task, two datasets should be obtained for the meta-training stage, i.e., the training support dataset $\mathbb{D}_{\mathrm{TrSup}}(k)$ and the training query dataset $\mathbb{D}_{\mathrm{TrQue}}(k)$. Following Section IV-B, we can collect N_{Tr} sample pairs for each source task and then randomly divide the N_{Tr} sample pairs, i.e., $\{(\boldsymbol{\xi} \circ \boldsymbol{h}_u(f_{\mathrm{D},u}), \boldsymbol{\xi} \circ \boldsymbol{h}_u(f_{\mathrm{U},u}))\}$, into the training support dataset $\mathbb{D}_{\mathrm{TrSup}}(k)$ and the training query dataset $\mathbb{D}_{\mathrm{TrQue}}(k)$. Note that $\mathbb{D}_{\mathrm{TrSup}}(k) \cap \mathbb{D}_{\mathrm{TrQue}}(k) = \emptyset$ should be satisfied to improve the generalization capability of the network. For the k-th target task, the adaption dataset $\mathbb{D}_{\mathrm{Ad}}(k)$ and the testing dataset $\mathbb{D}_{\mathrm{Te}}(k)$ can be obtained following Section IV-B.

B. Meta-training Stage

In the meta-training stage, the meta-learning algorithm aims to learn a network initialization that can effectively adapt to a new task. The parameter of FNN, i.e., $\boldsymbol{\Omega}$, is randomly initialized and is then updated by two iterative processes, i.e., the inner-task and the across-task updates.

1) Inner-task update: Denote the source-task-specific parameter as $\boldsymbol{\Omega}_{\mathrm{S},k}$. The goal of the inner-task updates is to minimize the loss function on the training support dataset $\mathbb{D}_{\mathrm{TrSup}}(k)$, i.e., $\mathrm{Loss}_{\mathbb{D}_{\mathrm{TrSup}}(k)}(\boldsymbol{\Omega}_{\mathrm{S},k})$, by iteratively updating the parameter $\boldsymbol{\Omega}_{\mathrm{S},k}$. Specifically, $\boldsymbol{\Omega}_{\mathrm{S},k}$ is initialized as the current network parameter $\boldsymbol{\Omega}$, and is then updated with G_{Tr} steps of gradient descents. Since an overlarge gradient leads to the instability of gradient descents, we limit the values of the source-task-specific gradient $\nabla_{\boldsymbol{\Omega}_{\mathrm{S},k}} \mathrm{Loss}_{\mathbb{D}_{\mathrm{TrSup}}(k)}(\boldsymbol{\Omega}_{\mathrm{S},k})$ into a certain range and obtain the truncated source-task-specific gradient $\boldsymbol{\upsilon}_{\mathrm{S},k}$ as [38]

$$[\boldsymbol{\upsilon}_{\mathrm{S},k}]_p = \min\left\{\Upsilon, \left[\nabla_{\boldsymbol{\Omega}_{\mathrm{S},k}} \mathrm{Loss}_{\mathbb{D}_{\mathrm{TrSup}}(k)}(\boldsymbol{\Omega}_{\mathrm{S},k})\right]_p\right\}, \quad p = 1, \cdots, \mathrm{len}(\boldsymbol{\upsilon}_{\mathrm{S},k}), \tag{18}$$

where Υ is the upper threshold of the gradient. Generally, an overlarge Υ may lead to large fluctuations of the loss function, while a too small Υ distorts the direction of the update, resulting in too early convergence. By trails and adjustments, a proper Υ is selected such that the inner-task updates converge stably and efficiently. Notice that the gradient truncation is not required in ADAM since ADAM normalizes the size of update steps for each parameter by using the square root of the 2nd moment vector to divide the 1st moment vector, as shown in Eq. (17). Then, the parameter $\boldsymbol{\Omega}_{\mathrm{S},k}$ is updated based on the truncated gradient $\boldsymbol{\upsilon}_{\mathrm{S},k}$, i.e.,

$$\boldsymbol{\Omega}_{\mathrm{S},k} \leftarrow \boldsymbol{\Omega}_{\mathrm{S},k} - \beta \boldsymbol{\upsilon}_{\mathrm{S},k}, \tag{19}$$

where β is the inner-task learning rate. It should mentioned that in [29], the inner-task update only involves one step of gradient descent. Here we extend it to G_{Tr} steps for better performance.

2) Across-task update: In the t-th time step, we randomly select K_{B} source tasks out of K_{S} source tasks[5] and generate corresponding support datasets $\{\mathbb{D}_{\mathrm{TrSup}}(k)\}_{k=1}^{K_{\mathrm{B}}}$ and query datasets $\{\mathbb{D}_{\mathrm{TrQue}}(k)\}_{k=1}^{K_{\mathrm{B}}}$ following Section V-A. The optimization objective of the t-th update is the loss function associated with the K_{B} training query datasets $\{\mathbb{D}_{\mathrm{TrQue}}(k)\}_{k=1}^{K_{\mathrm{B}}}$ and the meta-trained source-task-specific parameter $\boldsymbol{\Omega}_{\mathrm{S},k}$, i.e.,

$$\sum_{k=1}^{K_{\mathrm{B}}} \mathrm{Loss}_{\mathbb{D}_{\mathrm{TrQue}}(k)}(\boldsymbol{\Omega}_{\mathrm{S},k}).$$

The network parameter $\boldsymbol{\Omega}$ is updated via the ADAM algorithm. Specifically, the biased 1st and 2nd moment vectors are initialized with zeros and are then updated in the t-th time step following

$$\begin{cases} \boldsymbol{\mu}_3 \leftarrow \rho_1 \boldsymbol{\mu}_3 + (1-\rho_1) \nabla_{\boldsymbol{\Omega}} \sum_{k=1}^{K_{\mathrm{B}}} \mathrm{Loss}_{\mathbb{D}_{\mathrm{TrQue}}(k)}(\boldsymbol{\Omega}_{\mathrm{S},k}), \\ \boldsymbol{\nu}_3 \leftarrow \rho_2 \boldsymbol{\nu}_3 + (1-\rho_2) \left(\nabla_{\boldsymbol{\Omega}} \sum_{k=1}^{K_{\mathrm{B}}} \mathrm{Loss}_{\mathbb{D}_{\mathrm{TrQue}}(k)}(\boldsymbol{\Omega}_{\mathrm{S},k})\right)^2. \end{cases} \tag{20}$$

Next, the unbiased 1st and 2nd moment vectors can be obtained by

$$\begin{cases} \bar{\boldsymbol{\mu}}_3 \leftarrow \boldsymbol{\mu}_3/(1-\rho_1^t), \\ \bar{\boldsymbol{\nu}}_3 \leftarrow \boldsymbol{\nu}_3/(1-\rho_2^t). \end{cases} \tag{21}$$

The update of network parameter $\boldsymbol{\Omega}$ can be written as follows:

$$\boldsymbol{\Omega} \leftarrow \boldsymbol{\Omega} - \eta \bar{\boldsymbol{\mu}}_3/(\sqrt{\bar{\boldsymbol{\nu}}_3} + \varepsilon). \tag{22}$$

After the across-task updates finished, the meta-trained network parameter will be obtained by $\boldsymbol{\Omega}_{\mathrm{Mt}} \leftarrow \boldsymbol{\Omega}$.

There are two important differences between the inner-task and across-task updates. (i) Each inner-task update is performed on the support dataset of one source task while each across-task update is performed on the query datasets of K_{B} source tasks; (ii) The aim of inner-task updates is to optimize the task-specific parameters $\{\boldsymbol{\Omega}_{\mathrm{T},k}\}_{k=1}^{K_{\mathrm{B}}}$ while the aim of across-task updates is to optimize the overall network parameters $\boldsymbol{\Omega}$.

[5]Typically, there is $K_{\mathrm{B}} \ll K_{\mathrm{S}}$.

Besides, the across-task learning rate γ is typically set to be smaller than the inner-task learning rate β. This allows the network to acquire task-specific knowledge rapidly and learn across-task knowledge slowly. By alternately implementing the inner-task and across-task updates until the loss converges, the network learns model initialization that can adapts to a new task using only a small number of samples.

It should be mentioned that in [29], the across-task update is preformed by stochastic gradient descent (SGD) like the inner-task update. Here we adopt the ADAM algorithm instead of SGD since the ADAM benefits from the moment technique and the parameter-specific learning rates, and thus significantly outperforms SGD in most situations [39].

C. Meta-adaption and Testing Stages

Following Section V-A, we generate the adaption datasets $\{\mathbb{D}_{\mathrm{Ad}}(k)\}_{k-1}^{k=K_{\mathrm{T}}}$ and the testing datasets $\{\mathbb{D}_{\mathrm{Te}}(k)\}_{k-1}^{k=K_{\mathrm{T}}}$ for the meta-adaption and testing stages, respectively. In the meta-adaption stage, the meta-learning algorithm aims to fine-tune the network on the adaption dataset $\mathbb{D}_{\mathrm{Ad}}(k)$ so that the network can predict the downlink CSI on $\mathbb{D}_{\mathrm{Te}}(k)$ as accurate as possible. For the k-th target task, the direct-transfer algorithm initializes the target-task-specific parameter $\mathbf{\Omega}_{\mathrm{T},k}$ as the meta-trained network parameter $\mathbf{\Omega}_{\mathrm{Mt}}$, and then fine-tunes $\mathbf{\Omega}_{\mathrm{T},k}$ on the adaption dataset $\mathbb{D}_{\mathrm{Ad}}(k)$ via G_{Ad} steps of gradient descents. The optimization objective is the loss function on the $\mathbb{D}_{\mathrm{Ad}}(k)$, i.e., $\mathrm{Loss}_{\mathbb{D}_{\mathrm{Ad}}(k)}\left(\mathbf{\Omega}_{\mathrm{T},k}\right)$. Then, the truncated target-task-specific gradient $\boldsymbol{\upsilon}_{\mathrm{T},k}$ can be obtained by

$$\left[\boldsymbol{\upsilon}_{\mathrm{T},k}\right]_p = \min\left\{\Upsilon, \left[\nabla_{\mathbf{\Omega}_{\mathrm{T},k}}\mathrm{Loss}_{\mathbb{D}_{\mathrm{Ad}}(k)}\left(\mathbf{\Omega}_{\mathrm{T},k}\right)\right]_p\right\}, \quad p = 1, \cdots, \mathrm{len}(\boldsymbol{\upsilon}_{\mathrm{T},k}). \tag{23}$$

Then, the parameter $\mathbf{\Omega}_{\mathrm{T},k}$ is updated based on the truncated gradient $\boldsymbol{\upsilon}_{\mathrm{T},k}$, i.e.,

$$\mathbf{\Omega}_{\mathrm{T},k} \leftarrow \mathbf{\Omega}_{\mathrm{T},k} - \beta\boldsymbol{\upsilon}_{\mathrm{T},k}. \tag{24}$$

After the fine-tuning finished, the target-task-specific parameter $\mathbf{\Omega}_{\mathrm{T},k}$ will be fixed. Let $\mathrm{NMSE}_{\mathrm{Mt}}(\mathrm{k})$ represent the NMSE of the meta-learning algorithm evaluated on the k-th target environment that can be obtained by testing the network on the dataset $\mathbb{D}_{\mathrm{Te}}(k)$ using Eqs. (8) and (14). Then, the prediction NMSE of the meta-learning algorithm can be obtained by $\mathrm{NMSE}_{\mathrm{Mt}} = \sum_{\mathrm{k}=1}^{\mathrm{K_T}} \mathrm{NMSE}_{\mathrm{Mt}}(\mathrm{k})/\mathrm{K_T}$. The concrete steps of the meta-learning algorithm are given in **Algorithm 3**.

Fig. 4 illustrates the trajectories of parameter updating during the meta-training and meta-adaption stages of the meta-learning algorithm. The black solid line represents the updating trajectory of the network parameter $\mathbf{\Omega}$ during the meta-training stage. The blue dashed lines represent the updating trajectories of the target-task-specific parameters $\{\mathbf{\Omega}_{\mathrm{T},k}\}_{k=1}^{3}$ during the meta-adaption stage. The green dotted lines represent the trajectories of the source-task-specific parameters $\{\mathbf{\Omega}_{\mathrm{S},k}\}_{k=1}^{6}$ during the inner-task updates. As shown in Fig. 4, K_{B} red dash-dotted lines point to the next position of parameter $\mathbf{\Omega}$, which implies that K_{B} trained source-task-specific parameters are involved in the across-task update to determine the updating direction of $\mathbf{\Omega}$. As depicted in Fig. 3 and Fig. 4, the

Algorithm 3: Meta-learning algorithm for downlink CSI prediction

Input: Source tasks: $\{\mathcal{T}_{\mathrm{S}}(k)\}_{k=1}^{K_{\mathrm{S}}}$, Target tasks: $\{\mathcal{T}_{\mathrm{T}}(k)\}_{k=1}^{K_{\mathrm{T}}}$, inner-task learning rate: β, across-task learning rate: γ, exponential decay rates for across-task learning: $\{\rho_1, \rho_2\}$, disturbance factor: ε, number of gradsteps for inner-task training: T_{Tr}, number of tasks in each time index: K_{B}

Output: Meta-trained network parameter: $\mathbf{\Omega}_{\mathrm{Mt}}$, Predicted downlink CSI based on $\{\mathbb{D}_{\mathrm{Te}}(k)\}_{k=1}^{K_{\mathrm{T}}}$, NMSE of the meta-learning algorithm: $\mathrm{NMSE}_{\mathrm{Mt}}$

```
Meta-training stage
Randomly initialize the network parameters Ω
Initialize the time step: t ← 0
Initialize the 1st and 2nd moment vectors: μ, ν ← 0
while not done do
    Randomly select K_B tasks from {T_S(k)}_{k=1}^{K_S}
    Generate corresponding support dataset {D_TrSup(k)}_{k=1}^{K_B} and query dataset {D_TrQue(k)}_{k=1}^{K_B}
    for k = 1, ..., K_B do
        Initialize the parameter Ω_k ← Ω
        for g = 1, ..., G_Tr do
            Obtain truncated gradient υ_{S,k} using Eq. (18)
            Update the parameter Ω_{S,k} with gradient descent: Ω_{S,k} ← Ω_{S,k} − βυ_{S,k}
        end
    end
    t ← t + 1
    Update biased 1st and 2nd moment vectors using Eq. (20)
    Compute unbiased 1st and 2nd moment vectors using Eq. (21)
    Update the parameter Ω unbiased 1st and 2nd moment vectors: Ω ← Ω − ημ̄_3/(√ν̄_3 + ε)
end
Meta-trained network parameter: Ω_Mt ← Ω
Meta-adaption and Testing
Initialize NMSE: NMSE_Mt ← 0
for k = 1, ..., K_T do
    Generate the datasets D_Ad(k) and D_Te(k) for T_T(k)
    Meta-adaption stage
    Load the network parameter Ω_{T,k} ← Ω_Mt
    for g = 1, ..., G_Ad do
        Obtain truncated gradient υ_{T,k} using Eq. (23)
        Update the parameter Ω_{T,k} with gradient descent: Ω_{T,k} ← Ω_{T,k} − βυ_{T,k}
    end
    Testing stage
    Predict the downlink CSI base on D_Te(k) and Ω_{T,k} using Eq. (8)
    Calculate NMSE_Mt(k) using Eq. (14)
    NMSE_Mt ← NMSE_Mt + NMSE_Mt(k)/K_T
end
```

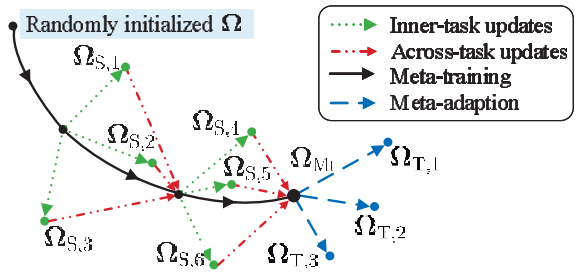

Fig. 4. Illustration of the meta-learning algorithm, where $K_{\mathrm{B}} = K_{\mathrm{T}} = 3$.

meta-learning algorithm takes the unique features of different tasks into account and exploits the joint guidance of multiple source tasks while the direct-transfer algorithm simply regards all the source tasks as one source task and ignores the structure information in different tasks.

VI. Simulation Results

Unless otherwise specified, the system parameters are set as follows: The BS is equipped with 64 antennas. To generate the training dataset, the uplink frequency is randomly selected from $[1, 3]$ GHz. The frequency difference between the uplink and the downlink is 120 MHz. We approximate the channel in Eq. (1) by 200 discrete propagation paths. The incident AS is 20°. For different environments, the power of attenuations and AS are different. For different users, the attenuation of each path follows Rayleigh distribution. The phase and delay of each path follow uniform distribution over $[-\pi, \pi)$ and $[0, 10^{-4}]s$. For the same user, the delays and DOAs of the uplink and downlink channels are the same.

The meta-learning, the no-transfer and the direct-transfer algorithms are all implemented on one computer with one Nvidia GeForce GTX 1080 Ti GPU. TensorFlow 1.4.0 is employed as the deep learning framework. The parameters of the no-transfer, the direct-transfer, and the meta-learning algorithms are given in Tab. I. The numbers of neurons in the input and the output layers are consistent with the lengths of input and output data vectors, respectively. The trainable parameters of FNN are randomly initialized as truncated normal variables[6] with normalized variance[7]. To be fair, the number of training samples in $\mathbb{D}_{\mathrm{STr}}$ are equal to the number of training samples used in the meta-learning algorithm, i.e., the total number of samples in the training set $K_{\mathrm{S}} \times N_{\mathrm{Tr}} = 8000$.

Fig. 5 depicts the NMSE performance of the no-transfer, the direct-transfer, and the meta-learning algorithms (i.e., $\mathrm{NMSE}_{\mathrm{Nt}}$, $\mathrm{NMSE}_{\mathrm{Dt}}$ and $\mathrm{NMSE}_{\mathrm{Mt}}$) versus the number of gradsteps G_{Ad}. The NMSE performance is the average accuracy of channel estimation from 100 target tasks, i.e., obtained by repetitively (fine-tuning and then) testing 100 different target environments. Since the network parameters of the no-transfer algorithm do not fine-tune based on the target environment, the accuracy curve of the no-transfer algorithm is a horizontal line. As shown in Fig. 5, the prediction accuracy

[6]The truncated normal distribution is a normal distribution bounded by two standard deviations from the mean.

[7]The weights of neurons in the l-th layer are initialized as truncated normal variables with variance $1/n_l$.

TABLE I
Default Parameters for the No-Transfer, the Direct-Transfer, and the Meta-Learning Algorithms

Parameter	Value
Number of neurons in hidden layers	(128, 128)
Learning rates: (γ, β, η),	(5e-4, 1e-3, 1e-3)
Exponential decay rates for ADAM: (ρ_1, ρ_2)	(0.9, 0.999)
Disturbance factor for ADAM: ε	1e-08
Number of source tasks: K_{S}	400
Number of tasks in each training index: K_{B}	10
Number of samples in each source task: N_{Tr}	20
Number of samples in the target task: $(N_{\mathrm{Ad}}, N_{\mathrm{Te}})$	(20, 20)
Number of users in each task: U	2
Number of gradsteps: $(T_{\mathrm{Tr}}, G_{\mathrm{Ad}})$	(10, 5e3)
Batch size: V	128

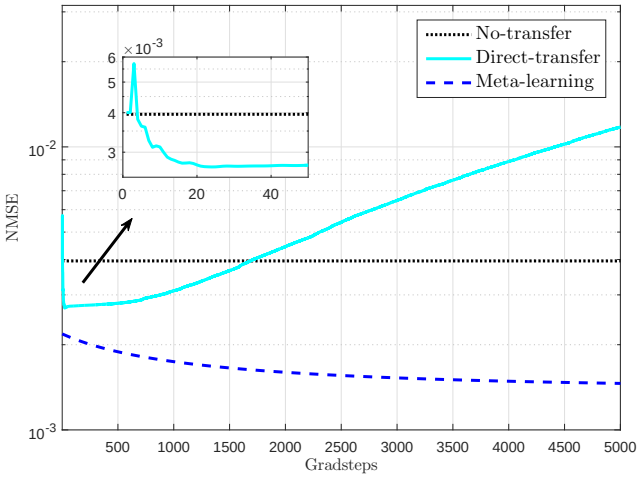

Fig. 5. The NMSE performance of the no-transfer, the direct-transfer, and the meta-learning algorithms versus the number of gradsteps G_{Ad}, where the number of samples in $\mathbb{D}_{\mathrm{AdSup}}$ is 20.

of the direct-transfer algorithm first improves as G_{Ad} increases, then degrades rapidly as G_{Ad} increases, and is even worse than the no-transfer algorithm when G_{Ad} is larger than 1700. This implies that the direct-transfer algorithm severely overfits as G_{Ad} increases. As shown in the partial enlarged picture, the optimal performance of the direct-transfer algorithm is achieved when G_{Ad} is equal to 25. To circumvent the problem of overfitting, the direct-transfer algorithm can stop optimization when the prediction accuracy no longer improves. The optimal prediction accuracy of the direct-transfer algorithm is achieved with different values of G_{Ad} in different simulation conditions. In the following simulations, we use the optimal performance of the direct-transfer algorithm as the benchmark. Furthermore, the optimal performance of the direct-transfer algorithm is significantly better than the no-transfer algorithm, which indicates that additional samples for the target task and appropriate fine-tuning can improve the prediction accuracy, i.e., transfer learning is more suitable than classical deep learning in downlink channel prediction. It also can be noticed that the performance of the meta-learning algorithm without meta-adaption stage is significantly better than the no-transfer learning algorithm, which demonstrates that the meta-learning algorithm can find a better initialization for fast adaption than the no-transfer learning algorithm. Moreover, the performance

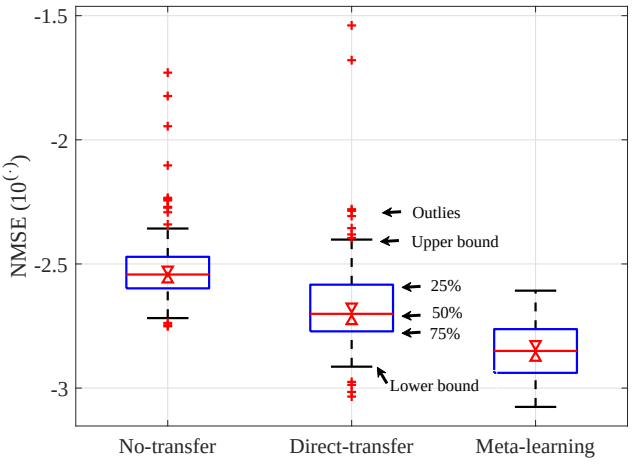

Fig. 6. The stability comparisons of the no-transfer, the direct-transfer, and the meta-learning algorithms.

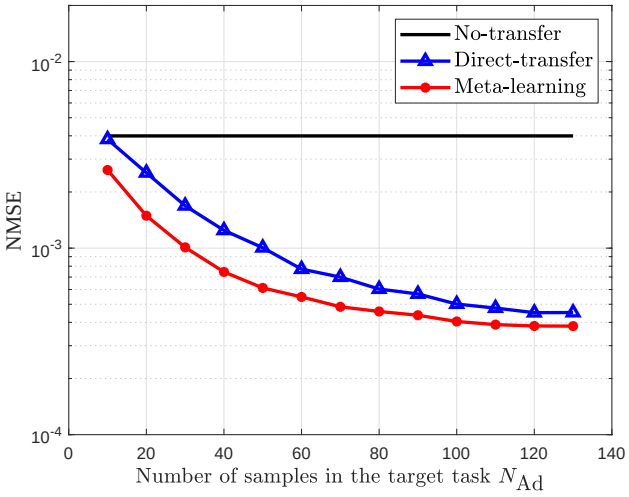

Fig. 7. The NMSE performance of the no-transfer, the direct-transfer, and the meta-learning algorithms versus the number of samples in $\mathbb{D}_{\text{AdSup}}$.

of the meta-learning algorithm tends to improve as G_{Ad} increases while the direct-transfer algorithm tends to overfit as G_{Ad} increases, which also indicates that a better initialization can avoid overfitting effectively.

Fig. 6 compares the accuracy stabilities of the no-transfer, the direct-transfer, and the meta-learning algorithms, where the robustness over different environments are considered. By repetitively (fine-tuning and then) testing 100 new environments, we can obtain the optimal prediction accuracies of the meta-learning and direct-transfer algorithms for each new environment. As shown in Fig. 6, the boxplot displays the distribution of prediction accuracies with respect to 100 environments with six indexes, i.e., "outlies", "upper bound", "first quartile", "median", "third quartile", and "lower bound". The prediction accuracies of 50 environments range between the "lower bound" and "upper bound", i.e., within the range of the blue box. As shown in Fig. 6, the red "+" denotes the "outlies". The algorithm with more outlies exhibits more unstable performance. In the no-transfer algorithm, the prediction NMSEs of users in 11 environments is higher than the corresponding upper bound NMSE. In the direct-transfer algorithm, the prediction NMSEs of users in 8 environments is higher than the corresponding upper bound NMSE. There is no outlies in the meta-learning algorithm, which demonstrate its remarkable robustness and stability over different environments.

Fig. 7 depicts the NMSE performance of the no-transfer, the direct-transfer, and the meta-learning algorithms versus the number of samples in $\mathbb{D}_{\text{AdSup}}$. For each sample size, the NMSE performance is the average accuracy of channel estimation from 100 environments. Since the network parameters of the no-transfer algorithm do not use the dataset $\mathbb{D}_{\text{AdSup}}$, the accuracy curve of the no-transfer algorithm is a horizontal line. It cam be seen from Fig. 7 that the prediction accuracies of the direct-transfer, and the meta-learning algorithms both improves as the sample size increases. Compared with the no-transfer algorithm, the improvement of direct-transfer algorithm is negligible when the sample size $\mathbb{D}_{\text{AdSup}}$ is 10. However, when the sample size is larger than 110, the performance of the direct-transfer, and the meta-learning algorithms both become saturated, which indicates that the sample size is large enough to fine-tune the network. Moreover, the meta-learning algorithm outperforms the direct-transfer algorithm, especially when the sample size is small, which implies its superiority in few-sample learning.

Fig. 8 shows the NMSE performance of the no-transfer, the direct-transfer, and the meta-learning algorithms versus the frequency difference Δf. For each frequency difference point, the NMSE performance is the average accuracy of channel estimation from 100 environments. The prediction accuracies of the three algorithms all degrades as the frequency difference increases. This is because the channel correlation between the uplink and the downlink tends to vanish as the frequency difference increases. The meta-learning algorithm outperforms the no-transfer and the direct-transfer algorithms, and the performance gaps becomes narrower as the frequency difference increases. When the frequency difference is small than 400 MHz, the prediction accuracies of both the meta-learning and the direct-transfer algorithms are higher than 10^{-2}, which validates the effectiveness of the meta-learning and the direct-transfer algorithms.

VII. Conclusion

In this paper, we formulated the downlink channel prediction for FDD massive MIMO systems as a deep transfer learning problem, where each learning task represents the downlink CSI prediction from the uplink CSI for a certain environment. Then, we proposed the no-transfer, direct-transfer and meta-learning algorithms based on the fully-connected neural network architecture. The no-transfer algorithm trains the network in the classical deep learning manner. The direct-transfer algorithm fine-tunes the network based on the initialization of the no-transfer algorithm. The meta-learning algorithm learns a model initialization that can effectively adapt to a new environment with a small amount of labeled data. Simulation results have shown that the meta-learning

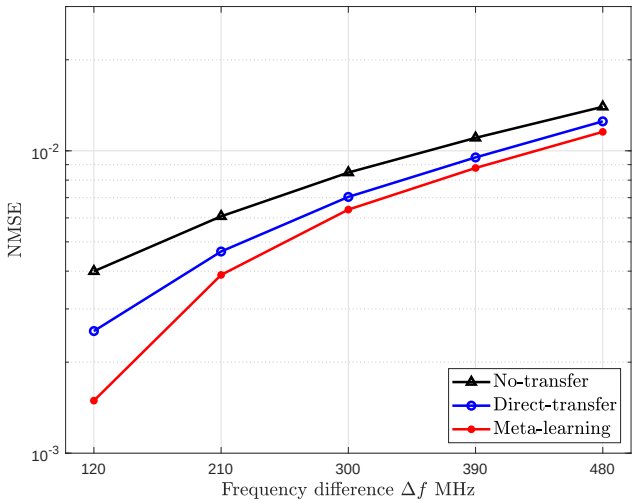

Fig. 8. The NMSE performance of the no-transfer, the direct-transfer, and the meta-learning algorithms versus the frequency difference Δf.

algorithm significantly outperforms the direct-transfer and the no-transfer algorithms in terms of both accuracy and stability, especially when the number of samples are very small, which demonstrates its effectiveness and superiority.

REFERENCES

[1] B. Wang, F. Gao, S. Jin, H. Lin, and G. Y. Li, "Spatial- and frequency-wideband effects in millimeter-wave massive MIMO systems," *IEEE Trans. Signal Process.*, vol. 66, no. 13, pp. 3393–3406, Jul. 2018.

[2] H. Xie, F. Gao, S. Zhang, and S. Jin, "A unified transmission strategy for TDD/FDD massive MIMO systems with spatial basis expansion model," *IEEE Trans. Vehicular Technol.*, vol. 66, no. 4, pp. 3170–3184, Apr. 2017.

[3] B. Wang, F. Gao, S. Jin, H. Lin, G. Y. Li, S. Sun, and T. S. Rappaport, "Spatial-wideband effect in massive MIMO with application in mmwave systems," *IEEE Commun. Mag.*, vol. 56, no. 12, pp. 134–141, Dec. 2018.

[4] J. Hoydis, C. Hoek, T. Wild, and S. ten Brink, "Channel measurements for large antenna arrays," in *Proc. Int. Symp. Wireless Commun. Systems (ISWCS)*, Paris, France, Aug. 2012, pp. 811–815.

[5] Y. Zhou, M. Herdin, A. Sayeed, and E. Bonek, "Experimental study of MIMO channel statistics and capacity via the virtual channel representation," *Univ. Wisconsin-Madison, Madison, WI, USA, Tech. Rep*, vol. 5, pp. 10–15, 2007.

[6] K. Hugl, K. Kalliola, and J. Laurila, "Spatial reciprocity of uplink and downlink radio channels in FDD systems," in *Proc. COST*. Citeseer, 2002, vol. 273, p. 066.

[7] Y. Han, Q. Liu, C. Wen, S. Jin, and K. Wong, "FDD massive MIMO based on efficient downlink channel reconstruction," *IEEE Trans. Commun.*, pp. 1–1, 2019.

[8] H. Xie, F. Gao, S. Jin, J. Fang, and Y. Liang, "Channel estimation for TDD/FDD massive MIMO systems with channel covariance computing," *IEEE Trans. Wireless Commun.*, vol. 17, no. 6, pp. 4206–4218, Jun. 2018.

[9] X. Rao and V. K. N. Lau, "Distributed compressive CSIT estimation and feedback for FDD multi-user massive MIMO systems," *IEEE Trans. Signal Process.*, vol. 62, no. 12, pp. 3261–3271, Jun. 2014.

[10] M. E. Eltayeb, T. Y. Al-Naffouri, and H. R. Bahrami, "Compressive sensing for feedback reduction in MIMO broadcast channels," *IEEE Trans. Commun.*, vol. 62, no. 9, pp. 3209–3222, Sep. 2014.

[11] Z. Qin, H. Ye, G. Y. Li, and B. F. Juang, "Deep learning in physical layer communications," *IEEE Wireless Commun.*, vol. 26, no. 2, pp. 93–99, Apr. 2019.

[12] Y. Yang, F. Gao, X. Ma, and S. Zhang, "Deep learning-based channel estimation for doubly selective fading channels," *IEEE Access*, vol. 7, pp. 36579–36589, Mar. 2019.

[13] H. He, C. Wen, S. Jin, and G. Y. Li, "Deep learning-based channel estimation for beamspace mmwave massive MIMO systems," *IEEE Wireless Commun. Lett.*, vol. 7, no. 5, pp. 852–855, Oct. 2018.

[14] Z. Jia, W. Cheng, and H. Zhang, "A partial learning based detection scheme for massive MIMO," *IEEE Wireless Commun. Lett.*, pp. 1–1, Apr. 2019.

[15] H. Ye, G. Y. Li, and B. Juang, "Power of deep learning for channel estimation and signal detection in OFDM systems," *IEEE Wireless Commun. Lett.*, vol. 7, no. 1, pp. 114–117, Feb. 2018.

[16] C. Wen, W. Shih, and S. Jin, "Deep learning for massive MIMO CSI feedback," *IEEE Wireless Commun. Lett.*, vol. 7, no. 5, pp. 748–751, Oct. 2018.

[17] S. Jin W. Xu, F. Gao and A. Alkhateeb, "3D scene based beam selection for mmwave communications," *arXiv preprint arXiv:1911.08409*, 2019.

[18] M. Alrabeiah, A. Hredzak, Z. Liu, and A. Alkhateeb, "ViWi: A deep learning dataset framework for vision-aided wireless communications," *arXiv preprint arXiv:1911.06257*, 2019.

[19] A. Alkhateeb, O. El Ayach, G. Leus, and R. W. Heath, "Channel estimation and hybrid precoding for millimeter wave cellular systems," *IEEE J. Sel. Topics Signal Process.*, vol. 8, no. 5, pp. 831–846, Oct. 2014.

[20] X. Li and A. Alkhateeb, "Deep learning for direct hybrid precoding in millimeter wave massive MIMO systems," *in Proc. Asilomars, arXiv e-prints*, p. arXiv:1905.13212, May 2019.

[21] Y. Yang, F. Gao, C. Qian, and G. Liao, "Model-aided deep neural network for source number detection," *arXiv preprint arXiv:1909.13273*, 2019.

[22] M. Safari and V., "Deep UL2DL: Channel knowledge transfer from uplink to downlink," *arXiv preprint arXiv:1812.07518*, 2018.

[23] P. Dong, H. Zhang, and G. Y. Li, "Machine learning prediction based CSI acquisition for FDD massive MIMO downlink," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Abu Dhabi, United Arab Emirates, Dec. 2018, pp. 1–6.

[24] M. Alrabeiah and A. Alkhateeb, "Deep learning for TDD and FDD massive MIMO: Mapping channels in space and frequency," *arXiv preprint arXiv:1905.03761*, 2019.

[25] Y. Yang, F. Gao, G. Y. Li, and M. Jian, "Deep learning based downlink channel prediction for FDD massive MIMO system," *IEEE Commun. Lett.*, pp. 1–1, 2019.

[26] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[27] J. Lin, Y. Wang, Y. Xia, T. He, and Z. Chen, "Learning to transfer: Unsupervised meta domain translation," *arXiv preprint arXiv:1906.00181*, 2019.

[28] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *Proc. Int. Conf. Artificial Neural Networks*. Springer, 2018, pp. 270–279.

[29] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc.34th Int. Conf. Machine Learning*. JMLR. org, 2017, vol. 70, pp. 1126–1135.

[30] M. Andrychowicz, M. Denil, S. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas, "Learning to learn by gradient descent by gradient descent," in *Proc. 30th Int. Conf. Neural Inf. Process. Systems*, Barcelona, Spain, 2016, pp. 3988–3996.

[31] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *Proc. Int. Conf. machine learning (ICML)*, 2016, pp. 1842–1850.

[32] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *Proc. Int. Conf. Learning Representations (ICLR)*, Toulon, France, 2017.

[33] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural netw.*, vol. 2, no. 5, pp. 359–366, 1989.

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

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016, http://www.deeplearningbook.org.

[36] S. Liang, R. Sun, Y. Li, and R. Srikant, "Understanding the loss surface of neural networks for binary classification," *arXiv preprint arXiv:1803.00909*, 2018.

[37] l. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proc. Int. Conf. Machine Learning (ICML)*, Atlanta, USA, 2013, pp. 1139–1147.

[38] J. Langford, L. Li, and T. Zhang, "Sparse online learning via truncated gradient," *J. Machine Learning Research*, vol. 10, no. Mar., pp. 777–801, 2009.

[39] J. Chen and A. Kyrillidis, "Decaying momentum helps neural network training," *arXiv preprint arXiv:1910.04952*, 2019.