Simplifying Meta-Beamforming Algorithms for Massive Wireless Networks

Cameron Kramr

***Abstract*—Machine learning in wireless communications has helped to solve many important issues related to next generation wireless systems. One major issue encountered by modern communication system is related to spatial multiplexing and reducing interference caused by neighboring users [13]. To reduce this interference, modern systems use electronic beamforming to direct the radiation pattern of each user away from every other user while maintaining contact with the base-station [2]. The optimal solution to this problem is an NP-hard Minimal Mean Squared Error (MMSE) optimization problem with a slow response time that needs to be recalculated as the users move and the channel properties changes [10]. To get around this, Deep Neural Networks (DNNs) have been deployed at various levels to either estimate the end-to-end problem, or to estimate certain difficult portions of the original computation [4]. This paper attempted to improve upon the architecture presented in [14] to reduce the computational complexity of calculating the optimal beam forming weights by using a GRU based architecture instead of an LSTM. While implementation details combined with an insufficiently documented source prevented further exploration, a great deal was learned about deploying neural networks in a reinforcement learning setting.**

I. Introduction

THE coming Industry 4.0 requires high-density wireless device connectivity with high-throughput [13]. High data-throughput requires large amounts of bandwidth, but bandwidth used by one device cannot be used by others. To mitigate this, traditional networks deploy time and frequency domain multiplexing [13]. These solutions allocate finite time and frequency between different users, and so through-put collapses as the number of users increases. To solve the problem of finite time and frequency, 5th generation networks apply spatial multiplexing which allows multiple users to share time and frequency and not interfere with each other [2], [13]. Spatial multiplexing is performed by electrically shaping the beam of an array of antennas toward its target and away from other users [2]. Beam-forming requires calculating weights and phase delays for a spatial array of antennas so that radiation goes mostly where desired and avoids undesired places [6], [4], [3].

The problem of optimizing antenna weights so that radiation reaches its target and avoids interfering with other users is NP-Hard [16]. Modern communications systems solve for the optimal weights using the Minimal Mean Squared Error method [16]. Because the MMSE method is iterative, the algorithm adapts slowly to changing environments which is unacceptable for mobile communications networks. To mitigate the slow response of the optimal solution, many sub-optimal approximations and other optimizations have been explored [12]. One optimization that shows promise is deploying Deep Neural Networks (DNNs) to approximate the antenna weights in dense networks [4]. State of the art beam-forming networks deploy DNNs either in end-to-end configurations, or to approximate difficult portions of the MMSE algorithm [4].

The paper [14] implemented an LSTM to decrease the computation required to find optimal antenna weights. The authors of [14] used an LSTM network to learn how to optimize the parameters of the optimization problem in a method that will be discussed later in the paper. The method of using LSTMs is the state of the art method for finding optimal antenna weights [4], [14]. This paper attempted to reduce the computation required to find antenna weights further using a simpler recurrent architecture, the Gated Recurrent Unit GRU and to document the process of training this architecture. This remains open work since this project was unable to properly replicate results of [2], [4], [10], [14]. The remainder of this paper will discuss in detail the exact optimization method attempted in the background and prior work sections, the experimentation and model section will detail the model attempted, followed by the results that were achieved in pursuit of this project. Finally, a conclusion discusses possible future work.

A. Background

Using multiple antenna to electrically shape antenna beams has been a staple of advanced wireless communication and radar systems since around the 1970's [**?**]. Beamforming works by exploiting the additive properties of electro-magnetic waves which increase intensity when added in phase and can completely level a wave when added out of phase. Controlling the phase of multiple antenna, it is possible to control where in space the interference pattern from all the antenna's radiation patterns add constructively and where the interference adds destructively [**?**].

The ability of antenna systems to beamform depends primarily on the physical arrangement of the antenna in relation to their wavelength. Since early mobile communication systems used relatively long wavelengths, beamforming systems would need to be tens of centimeters at their smallest [**?**]. This was acceptable because the long wavelength used in early cellular communication systems suffered far less attenuation than the frequencies of 5th (and soon 6th) generation wireless systems [**?**]. Besides physical concerns related to antenna, it is mathematically difficult to calculate how to design the phase-delay and attenuation of each antenna in an array for a system involving even a few users [**?**].

An ideal multi-user beamforming system would be able to direct radiated energy at intended users while perfectly avoiding unintended users. The best known algorithm for finding parameters required to perform beamforming like this is called the Weighted Minimal Mean Squared Error (WMMSE) method and was first published by [?]. Since the WMMSE algorithm was first introduced, it was recognized as lacking many desirable qualities and several attempts have been made to improve the algorithm. Of specific interest was [?] who reformulated the original optimization problem in a way that permitted unrolling.

B. WMMSE for Beamforming and System Model

The problem formulation for the WMMSE in this analysis is presented by [?]. The system model is a multiple-input single-output, additive interference downlink channel. The downlink base-station has M independent transmit antenna and sends data to N single antenna users. All signals are considered to be additive, and so the received signal at user i is given by Equation 1.

$$\boldsymbol{y_i = h_i^H v_i x_i + \sum_{j=1, j\neq i}^{N} (h_i^H v_j x_j) + n_i} \tag{1}$$

The terms from this base equation are defined as:

- h_i: is the ith column of the complex channel matrix, H
- v_i: the complex antenna beamforming vector for user i
- x_i: the intended symbol to be sent to user i
- n_i: the circular Gaussian complex noise at user i

Each of these values can be complex to capture the magnitude and phase effects each variable. From this, notice that the first expression in Equation 1 represents the intended symbol received by user i. The second term is the interference received by user i caused by the base-station transmitting to other users. Vectors v_i and h_i have matrices V and H where each row represents a user and each column an antenna at the base-station. This way, the model captures the intended configurable weights of all transmit antenna (M) to send messages to each user (N) with V and the effect of the channel from each transmit antenna to each user. From this, [?] presents a global loss function to minimize the anticipated error rate of the system shown in Equation 2.

$$\mathcal{L} = \min_{u,w,V} \sum_{i=1}^{N} \alpha_i (w_i e_i - log_2 w_i) \tag{2}$$

Where e_i is the expectation of error quantified in Equation 3 as:

$$e_i = \mathbb{E}_{x,n_i}\{|\hat{x_i} - x_i|^2\} \tag{3}$$

Lastly, $\hat{x_i}$ is given as:

$$\hat{x_i} = u_i y_i \tag{4}$$

From these equations, the following new parameters are defined:

- α_i: A system design parameter describing importance of user i
- w_i: An optimized weight assigning importance to user i
- e_i: The expectation of error for user i
- u_i: User i's amplification

Looking deeper, w_i provides a way for the optimization algorithm to determine how important each node can be allowed to be, but prevents w_i from being too small. Furthermore, a receiver gain u_i is introduced. This parameter is controlled by users and allows each user to amplify received signals. The cost of doing so is increasing the amount of noise amplified by the receiver, so an optimization algorithm must find a way to balance that competing interest. Finally, there is also a weight α_i that can be assigned to each user by the system designer to describe how important each individual user is.

With the above equations, it is possible to define the fixed station update equation for each parameter as:

$$w_i = \frac{\sum_{j=1}^{N} |h_i^H v_j|^2 + \sigma^2}{\sum_{j=1, j\neq i}^{N} |h_i^H v_j|^2 + \sigma^2} \tag{5}$$

$$u_i = \frac{|h_i^H v_i|^2}{\sum_{j=1}^{N} |h_i^H v_j|^2 + \sigma^2} \tag{6}$$

$$v_i = \alpha_i u_i w_i h_i (A + \mu I)^{-1} \tag{7}$$

Notice how the raw update function for v_i in Equation 7 has a matrix inversion of $(A + \mu I)$. This is overcome by [?] through the use of Projected Gradient Descent (PGD) using the Lagrange multiplier method. Applying the PGD method transforms the error for e_i function from Equation 3 to be 8.

$$e_i = \sum_{j=1}^{N} |u_i h_i^H v_j|^2 - 2u_i h_i^H v_i + \sigma^2 |u_i|^2 + 1 \tag{8}$$

Next, the gradient of V is calculated explicitly as:

$$\nabla f(v_i) = -2\alpha_i w_i u_i h_i + 2A v_i \tag{9}$$

Finally, A is a matrix given in Equation 10

$$A = \sum_{i=1}^{N} \alpha_i w_i |u_i|^2 h_i h_i^H \tag{10}$$

The above equations were derived in more detail by [?] and the interested reader is directed there for additional details. The presented equations are necessary for understanding the problem statement and explaining the

This section will describe in detail the problem formulation which [14] used LSTMs to solve and which served as the basis of the work explored here and in [14]. The way the WMMSE algorithm works is to iteratively update each variable u_i, w_i, and V_i with the respective update functions shown in Equations 6, 5, and 7 respectively. The update procedure is repeated until an acceptable error is reached.

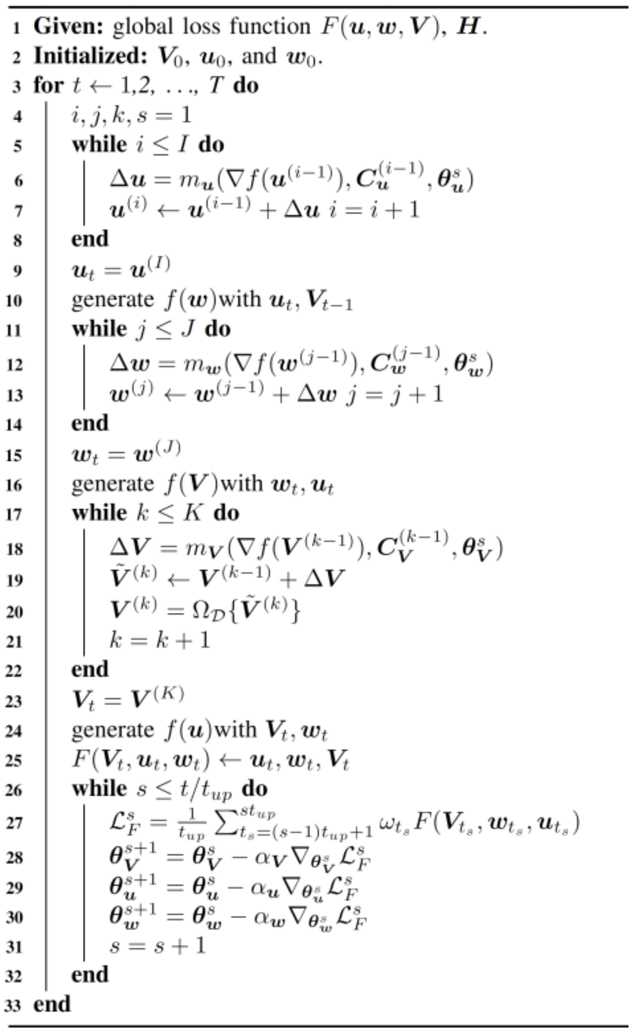
As was shown above, the original formulation for updating V_i involved a matrix inversion, so deep unfolding was implemented by [?] to circumvent the complications of calculating an inversion directly. Coincidentally, unfolding in this way creates a continuous path for gradients to flow within the entire problem which [14] used to replace the update functions with LSTM DNNs that learn how to optimize the variables u_i, w_i, and v_i for a given channel H.

C. Recurrent Neural Network for WMMSE

The authors in [14] built their architecture following the work of [?] who applied recent advancements in deploying DNNs in unfolded optimization problems. Specifically, [?] described the steps as in Table 3.

1) Map iterations of optimization algorithm to DNN
2) Fix number of iterations to compute availability
3) Optimize hyper-parameters within iteration allowance

Following the above algorithm, the authors in [14] used Tensorflow to create three LSTM networks, one for learning to update each variable u_i, w_i, and v_i based on the global loss function. This preserved the original structure of the unfolded WMMSE problem and creates the following algorithm as published by [14]:



```
Given: global loss function $F(\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{V})$, $\boldsymbol{H}$.
Initialized: $\boldsymbol{V}_0$, $\boldsymbol{u}_0$, and $\boldsymbol{w}_0$.
for $t \leftarrow 1, 2, \ldots, T$ do
    $i, j, k, s = 1$
    while $i \leq I$ do
        $\Delta \boldsymbol{u} = m_{\boldsymbol{u}}(\nabla f(\boldsymbol{u}^{(i-1)}), \boldsymbol{C}_{\boldsymbol{u}}^{(i-1)}, \boldsymbol{\theta}_{\boldsymbol{u}}^{s})$
        $\boldsymbol{u}^{(i)} \leftarrow \boldsymbol{u}^{(i-1)} + \Delta \boldsymbol{u}$ $i = i + 1$
    end
    $\boldsymbol{u}_t = \boldsymbol{u}^{(I)}$
    generate $f(\boldsymbol{w})$with $\boldsymbol{u}_t, \boldsymbol{V}_{t-1}$
    while $j \leq J$ do
        $\Delta \boldsymbol{w} = m_{\boldsymbol{w}}(\nabla f(\boldsymbol{w}^{(j-1)}), \boldsymbol{C}_{\boldsymbol{w}}^{(j-1)}, \boldsymbol{\theta}_{\boldsymbol{w}}^{s})$
        $\boldsymbol{w}^{(j)} \leftarrow \boldsymbol{w}^{(j-1)} + \Delta \boldsymbol{w}$ $j = j + 1$
    end
    $\boldsymbol{w}_t = \boldsymbol{w}^{(J)}$
    generate $f(\boldsymbol{V})$with $\boldsymbol{w}_t, \boldsymbol{u}_t$
    while $k \leq K$ do
        $\Delta \boldsymbol{V} = m_{\boldsymbol{V}}(\nabla f(\boldsymbol{V}^{(k-1)}), \boldsymbol{C}_{\boldsymbol{V}}^{(k-1)}, \boldsymbol{\theta}_{\boldsymbol{V}}^{s})$
        $\tilde{\boldsymbol{V}}^{(k)} \leftarrow \boldsymbol{V}^{(k-1)} + \Delta \boldsymbol{V}$
        $\boldsymbol{V}^{(k)} = \Omega_{\mathcal{D}}\{\tilde{\boldsymbol{V}}^{(k)}\}$
        $k = k + 1$
    end
    $\boldsymbol{V}_t = \boldsymbol{V}^{(K)}$
    generate $f(\boldsymbol{u})$with $\boldsymbol{V}_t, \boldsymbol{w}_t$
    $F(\boldsymbol{V}_t, \boldsymbol{u}_t, \boldsymbol{w}_t) \leftarrow \boldsymbol{u}_t, \boldsymbol{w}_t, \boldsymbol{V}_t$
    while $s \leq t/t_{up}$ do
        $\mathcal{L}_F^s = \frac{1}{t_{up}} \sum_{t_s=(s-1)t_{up}+1}^{s t_{up}} \omega_{t_s} F(\boldsymbol{V}_{t_s}, \boldsymbol{w}_{t_s}, \boldsymbol{u}_{t_s})$
        $\boldsymbol{\theta}_{\boldsymbol{V}}^{s+1} = \boldsymbol{\theta}_{\boldsymbol{V}}^{s} - \alpha_{\boldsymbol{V}} \nabla_{\boldsymbol{\theta}_{\boldsymbol{V}}^{s}} \mathcal{L}_F^s$
        $\boldsymbol{\theta}_{\boldsymbol{u}}^{s+1} = \boldsymbol{\theta}_{\boldsymbol{u}}^{s} - \alpha_{\boldsymbol{u}} \nabla_{\boldsymbol{\theta}_{\boldsymbol{u}}^{s}} \mathcal{L}_F^s$
        $\boldsymbol{\theta}_{\boldsymbol{w}}^{s+1} = \boldsymbol{\theta}_{\boldsymbol{w}}^{s} - \alpha_{\boldsymbol{w}} \nabla_{\boldsymbol{\theta}_{\boldsymbol{w}}^{s}} \mathcal{L}_F^s$
        $s = s + 1$
    end
end
```

Fig. 1. Algorithm from [14] for implementing the unfolded WMMSE algorithm using DNNs. Note that m_u, m_w, m_V represent the neural networks for updating each variable respectively. Furthermore, $C_{u,w,V}$ represent the internal state of each cell for the respective parameter

The algorithm in figure 5 shows the complete algorithm implemented by [14] that was to be replicated and improved in this paper. Further discussion for the model will take place in the following section.

II. Experimentation, Model, and Results

The key stages for the planned experimentation was to first get a working comparison using the traditional WMMSE algorithm then to implement the design in [14], and then to exchange the LSTM modules with GRU modules and to do various testing of the hyper-parameters to observe the effects of using a different number of hidden layers. All of this was experimentation that [14] did not perform and which would have revealed interesting new knowledge about applying DNNs in unsupervised learning optimization settings. Specifically, how much computation is actually required for the LSTM WMMSE algorithm to solve for beam-weights in wireless MISO systems. Several difficulties prevented this which will be discussed in some detail.

A. PyTorch Implementation

Using the formulation of the problem in 5, the following graph shows the attempted implementation:

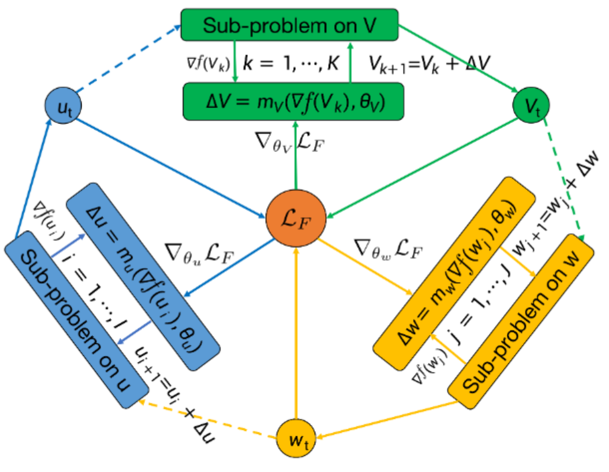

Sub-problem on V
$\nabla f(V_k)$
$k = 1, \cdots, K$
$V_{k+1} = V_k + \Delta V$
$\Delta V = m_V(\nabla f(V_k), \theta_V)$
u_t
V_t
$\nabla_{\theta_V} \mathcal{L}_F$
$\mathcal{L}_F$
$\nabla_{\theta_u} \mathcal{L}_F$
$\nabla_{\theta_w} \mathcal{L}_F$
$\nabla f(u_i)$
$\Delta u = m_u(\nabla f(u_i), \theta_u)$
$i = 1, \cdots, I$
Sub-problem on u
$u_{i+1} = u_i + \Delta u$
$\Delta w = m_w(\nabla f(w_j), \theta_w)$
$j = 1, \cdots, J$
$\nabla f(w_j)$
$w_{j+1} = w_j + \Delta w$
Sub-problem on w
w_t

Fig. 2. The graph showing WMMSE implemented using RNNs for updating the values.

The unfolded LSTM WMMSE algorithm was implemented using a single class to contain all of the relevant parts so that everything would be self-contained in-order to facilitate future experimentation on various configurations. There were three individual LSTM networks, supporting variables, and an Adam optimizer for tuning the LSTM weights along with all the necessary helper functions to calculate the portions of the algorithm. More detail can be found in the code submission portion of this assignment which is thoroughly documented.

B. WMMSE Deep Unfolding Replication

The authors in [?] made their code for the unfolded WMMSE algorithm public on GitHub which served as both a source of confusion and invaluable reference. The code provided by [?] would have served as a benchmark to compare the performance of any simpler RNN on the beamforming problem. To this end, the results of the WMMSE algorithm were replicated and can be seen in Figure ??. Since much of the other code from [?] used Tensorflow–which was unfamiliar–that part was not used. Unfortunately, since the results from

[14] could not be replicated with the details provided in their paper, no comparison was able to be performed.

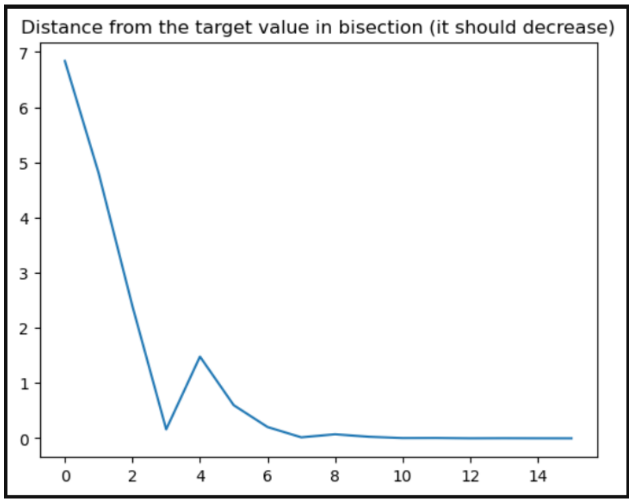


Fig. 3. WMMSE algorithm results implemented by [?] showing the algorithm successfully converging to a solution. The code that generated this figure was copied from [?] in order to be used as a basis of comparison.

Figure 3 shows the WMMSE implementation by [?] successfully solving for the beamweights in a 4 user 4 transmit antenna system. This was

C. *Implementation Difficulties*

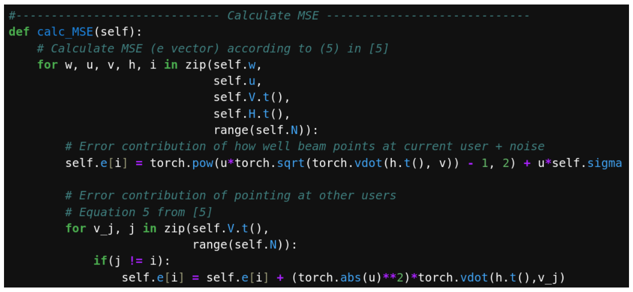
There were a number of substantial difficulties encountered when attempting to implement the algorithm described in [14]. Ultimately, these difficulties prevented any additional analysis, but this section will explore them in detail in-place of novel results.

The first major difficulty encountered was related to the novel model type used in this project. All of the assignments this semester used supervised learning, so the challenges inherent to creating unsupervised models were novel and substantial. Learning how to make the PyTorch AutoGrad function "happy" was by far the most challenging. A particularly painful lesson was learning to detach old intermediate tensors after stepping the optimizer function.

Implementing many of the update functions was difficult since many of them required "in-place" operations since many of the vectors were significantly dependent on position. For example, the error expectation vector, e implemented as follows:

I am still not sure that the gradient properly flows through this function, but did not have time to properly test it. I suspect that it does not, or that there are other errors in my implementation since the gradient diverges.

The second difficulty was massive uncertainty in implementation details of the algorithm. This was because the [14] did not publish their code and were not very specific when describing which functions were used. This was further complicated by mistaking the code of [?] for the LSTM module early on. The confusion this caused created major delays in development. After realizing this mistake, I did not have time to properly debug my algorithm implementation.



```
#---------------------------- Calculate MSE ----------------------------
def calc_MSE(self):
    # Calculate MSE (e vector) according to (5) in [5]
    for w, u, v, h, i in zip(self.w,
                             self.u,
                             self.V.t(),
                             self.H.t(),
                             range(self.N)):
        # Error contribution of how well beam points at current user + noise
        self.e[i] = torch.pow(u*torch.sqrt(torch.vdot(h.t(), v)) - 1, 2) + u*self.sigma

        # Error contribution of pointing at other users
        # Equation 5 from [5]
        for v_j, j in zip(self.V.t(),
                          range(self.N)):
            if(j != i):
                self.e[i] = self.e[i] + (torch.abs(u)**2)*torch.vdot(h.t(),v_j)
```

Fig. 4. Calculating the Mean Squared Error vector for the neural network. Notice that many of the operations are "in-place". This is according to how MSE should be calculated to [14], but doing so has troublesome effects on the Autograd in PyTorch due to the in-place operations

D. *Results*

After getting to a somewhat functional state, the LSTM network would always diverge. Many techniques were attempted to mitigate this from changing the initialization, to normalizing the inputs to the LSTMs, but nothing worked.

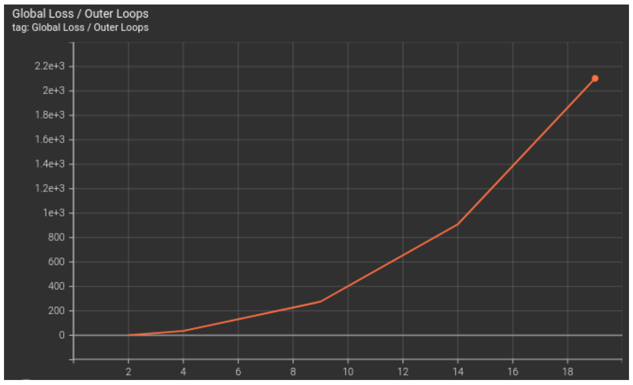


Fig. 5. Diverging loss function of LSTM version of WMMSE

I then instrumented every part of the gradient to attempt to find the issue, but despite all my efforts, the gradient would not remain contained.

III. Conclusion

Ultimately, I learned a large amount about implementing unsupervised learning algorithms using PyTorch. Specifically, how to make sure that the auto-gradients assemble properly. Unfortunately, because the implementation details were not specific enough, I was not able to successfully replicate the work of [14]. This likely relates to my lack of training in optimization which likely would have made many of my questions self-evident. Eventually, I figured out an algorithm that I thought would be similar enough to the correct algorithm, but which clearly did not work. After I finally got the PyTorch autogradient to function, I was hit with a wall of diverging gradients, and despite attempting various initialization techniques, normalizing the gradient inputs to the LSTMs, I was not able to train a network to solve the optimization problem. Future work would focus on getting a working network to solve the algorithm, and re-working the helper functions to be transparent to the gradient function.

IV. Roles

Cameron Kramr was the sole member of this group and performed all roles.

V. Code Availability

https://github.com/CameronKramr/ELEC_576_CKK5_Final_Project

References

[1] Mohamed Akrout, Amal Feriani, Faouzi Bellili, Amine Mezghani, and Ekram Hossain. Domain generalization in machine learning models for wireless communications: Concepts, state-of-the-art, and open issues. *IEEE Communications Surveys and Tutorials*, pages 1–1, 2023.

[2] Olakunle Elijah, Sharul Kamal Abdul Rahim, Wee Kiat New, Chee Yen Leow, Kanapathippillai Cumanan, and Tan Kim Geok. Intelligent massive mimo systems for beyond 5g networks: An overview and future trends. *IEEE Access*, 10:102532–102563, 2022.

[3] Jakob Hoydis, Stephan ten Brink, and Merouane Debbah. Massive mimo in the ul/dl of cellular networks: How many antennas do we need? *IEEE Journal on Selected Areas in Communications*, 31(2):160–171, 2013.

[4] Haya Al Kassir, Zaharias D. Zaharis, Pavlos I. Lazaridis, Nikolaos V. Kantartzis, Traianos V. Yioultsis, and Thomas D. Xenos. A review of the state of the art and future challenges of deep learning-based beamforming. *IEEE Access*, 10:80869–80882, 2022.

[5] Ricardo Lovato and Xun Gong. Phased antenna array beamforming using convolutional neural networks. In *2019 IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting*, pages 1247–1248, 2019.

[6] Siyuan Lu, Shengjie Zhao, and Qingjiang Shi. Learning-based massive beamforming, 2020.

[7] Taras Maksymyuk, Juraj Gazda, Oleh Yaremko, and Denys Nevinskiy. Deep learning based massive mimo beamforming for 5g mobile network. In *2018 IEEE 4th International Symposium on Wireless Systems within the International Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS)*, pages 241–244, 2018.

[8] Fangzhou Mu. maml in pytorch - re-implementation and beyond. https://github.com/fmu2/PyTorch-MAML, 2020.

[9] Lissy Pellaco, Mats Bengtsson, and Joakim Jaldén. Deep unfolding of the weighted mmse beamforming algorithm, 2020.

[10] Lissy Pellaco and Joakim Jaldén. A matrix-inverse-free implementation of the mu-mimo wmmse beamforming algorithm. *IEEE Transactions on Signal Processing*, 70:6360–6375, 2022.

[11] David A. Schmidt, Changxin Shi, Randall A. Berry, Michael L. Honig, and Wolfgang Utschick. Minimum mean squared error interference alignment. In *2009 Conference Record of the Forty-Third Asilomar Conference on Signals, Systems and Computers*, pages 1106–1110, 2009.

[12] Qingjiang Shi, Meisam Razaviyayn, Zhi-Quan Luo, and Chen He. An iteratively weighted mmse approach to distributed sum-utility maximization for a mimo interfering broadcast channel. *IEEE Transactions on Signal Processing*, 59(9):4331–4340, 2011.

[13] Danfeng Sun, Yanlong Xi, Abdullah Yaqot, Horst Hellbrück, and Huifeng Wu. Throughput maximization using deep complex networks for industrial internet of things. *Sensors*, 23(2):951, Jan 2023.

[14] Jingyuan Xia and Deniz Gunduz. Meta-learning based beamforming design for miso downlink. In *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE, jul 2021.

[15] Jingyuan Xia, Shengxi Li, Jun-Jie Huang, Imad Jaimoukha, and Deniz Gunduz. Meta-learning based alternating minimization algorithm for non-convex optimization, 2022.

[16] Jie Zhou and Yunmin Zhu. The linear minimum mean-square error estimation with constraints and its applications. In *2006 International Conference on Computational Intelligence and Security*, volume 2, pages 1801–1804, 2006.