PolarAir: A Compressed Sensing Scheme for Over-the-Air Federated Learning

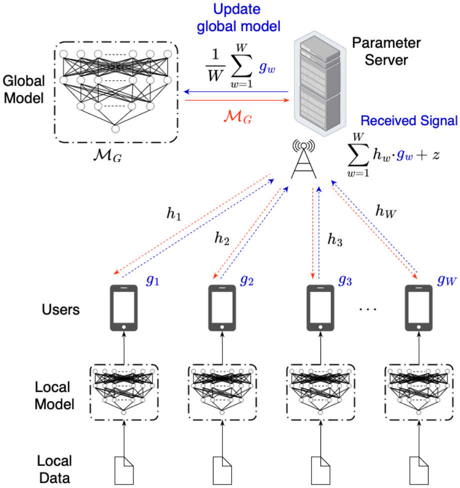
Michail Gkagkos, Krishna Narayanan,
Jean-Francois Chamberland, Costas N. Georghiades

Electrical and Computer Engineering
Texas A&M University

ITW 2023
Saint-Malo, France

Introduction: Over-the-Air Federated Learning

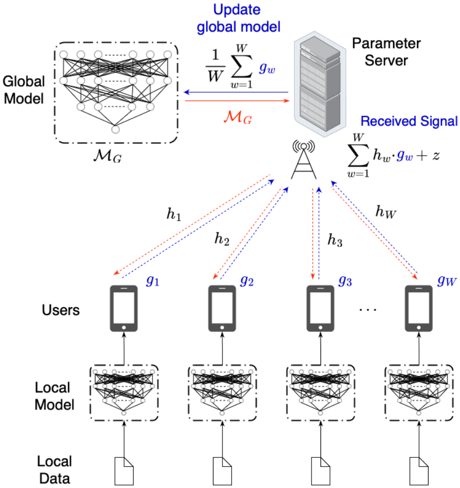
- Joint training of ML models with distributed data
- Exploit the additive nature of the wireless medium
- Uplink naturally computes sum of gradients



$$y = \sum_{w=1}^{W} h_w \, g_w + z$$

- No fading (AWGN) - $h_w = 1$
- SISO - precompensate for h_w

Introduction: Over-the-Air Federated Learning



$$y = \sum_{w=1}^{W} h_i \, g_w + z$$

- No fading (AWGN) - $h_i = 1$
- SISO - precompensate for h_i

- No. of channel uses = dimension of g_w = no. of parameters

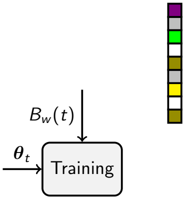
Problem Statement

- Train a ResNet Network over an AWGN channel
- Fix the Test Accuracy
- Minimize the number of channel uses needed to achieve it

Main Objective

- Design Compressed Sensing Algorithm
- Examine its behavior during the training

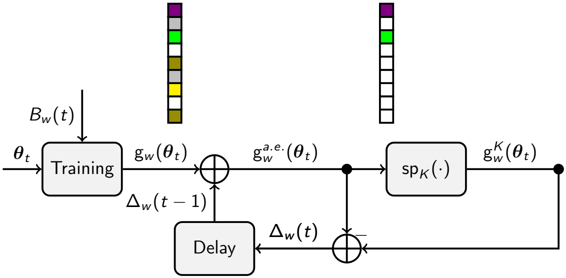
Framework for Over-the-air Federated Learning[1]



- The access point broadcasts the current model, $\boldsymbol{\theta}_t$, and workers use the available batch, $B_w(t)$ to compute the gradient $\mathrm{g}_w(\boldsymbol{\theta}_t)$
- Add the error from the previous sparsification step, $\mathrm{g}_w^{e.a.}(\boldsymbol{\theta}_t) = \mathrm{g}_w(\boldsymbol{\theta}_t) + \Delta_w(t-1)$
- Compute the error $\Delta_w(t) = \mathrm{g}_w^{a.e.}(\boldsymbol{\theta}_t) - \mathrm{g}_w^{K}(\boldsymbol{\theta}_t)$
- Sparsify $\mathrm{g}_w^{a.e.}(\boldsymbol{\theta}_t)$
- Compress the sparsified vector
- Scale the compressed version to satisfy the power constraint

[1] Proposed by M. M. Amiri and D. Gündüz's 2019 ISIT paper

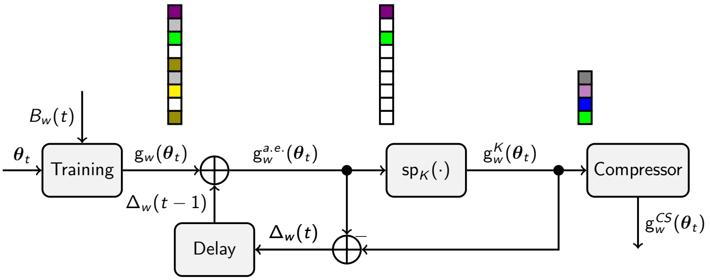
Framework for Over-the-air Federated Learning[1]



- The access point broadcasts the current model, θ_t, and workers use the available batch, $B_w(t)$ to compute the gradient $g_w(\theta_t)$
- Add the error from the previous sparsification step, $g_w^{e.a.}(\theta_t) = g_w(\theta_t) + \Delta_w(t-1)$
- Compute the error $\Delta_w(t) = g_w^{a.e.}(\theta_t) - g_w^K(\theta_t)$
- Sparsify $g_w^{a.e.}(\theta_t)$
- Compress the sparsified vector
- Scale the compressed version to satisfy the power constraint

[1] Proposed by M. M. Amiri and D. Gündüz's 2019 ISIT paper

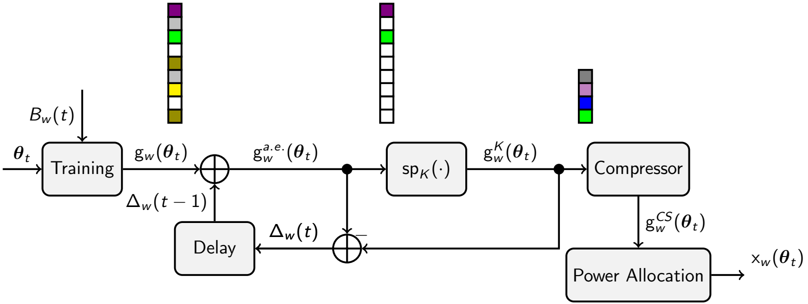
Framework for Over-the-air Federated Learning[1]



- The access point broadcasts the current model, θ_t, and workers use the available batch, $B_w(t)$ to compute the gradient $g_w(\theta_t)$
- Add the error from the previous sparsification step, $g_w^{e.a.}(\theta_t) = g_w(\theta_t) + \Delta_w(t-1)$
- Compute the error $\Delta_w(t) = g_w^{a.e.}(\theta_t) - g_w^K(\theta_t)$
- Sparsify $g_w^{a.e.}(\theta_t)$
- Compress the sparsified vector
- Scale the compressed version to satisfy the power constraint

[1] Proposed by M. M. Amiri and D. Gündüz's 2019 ISIT paper

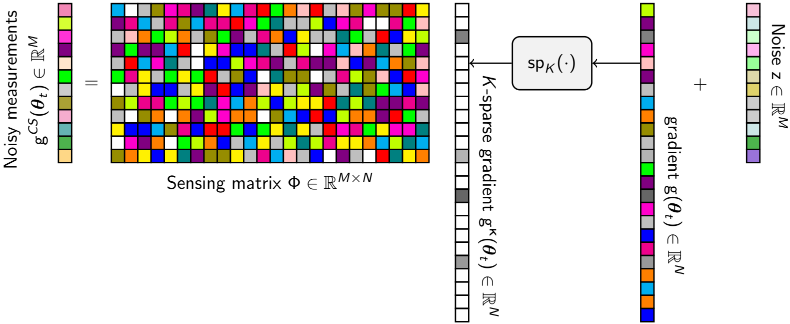
Framework for Over-the-air Federated Learning[1]



- The access point broadcasts the current model, θ_t, and workers use the available batch, $B_w(t)$ to compute the gradient $g_w(\theta_t)$
- Add the error from the previous sparsification step, $g_w^{e.a.}(\theta_t) = g_w(\theta_t) + \Delta_w(t-1)$
- Compute the error $\Delta_w(t) = g_w^{a.e.}(\theta_t) - g_w^{K}(\theta_t)$
- Sparsify $g_w^{a.e.}(\theta_t)$
- Compress the sparsified vector
- Scale the compressed version to satisfy the power constraint

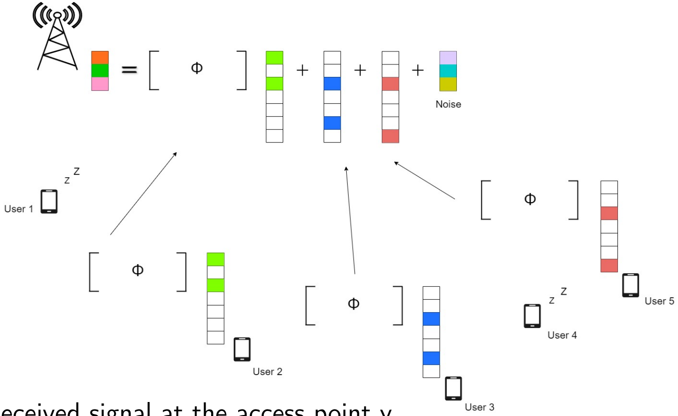
[1] Proposed by M. M. Amiri and D. Gündüz's 2019 ISIT paper

Compressed Sensing



- Gradient (model parameter updates) is K-sparse
- Each worker transmits $g^{CS}(\boldsymbol{\theta}_t) = \Phi g^K_w(\boldsymbol{\theta}_t)$

Over-the-air Federated Learning with Compressed Sensing



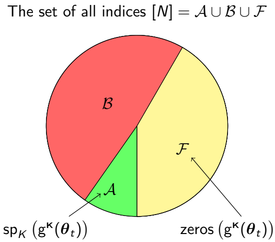
- Received signal at the access point y

$$\mathrm{y}(t) = \sum_{w=1}^{W} \Phi(t)\mathrm{g}_w^{\mathsf{K}}(\boldsymbol{\theta}_t) + \mathrm{z}(t) = \Phi(t)\sum_{w=1}^{W} \mathrm{g}_w^{\mathsf{K}}(\boldsymbol{\theta}_t) + \mathrm{z}(t)$$
$$= \Phi(t)\mathrm{g}^{\mathsf{K}}(\boldsymbol{\theta}_t) + \mathrm{z}(t)$$

- The channel "computes" the aggregated gradient

Recovery of aggregated gradient

- Sum of the W, K-sparse vectors has $K^{'} \in [K : KW]$ non-zero entries
- We choose to recover top-K entries from $\mathrm{y}(t) = \Phi(t)\mathrm{g}^{\mathsf{K}}(\boldsymbol{\theta}_t) + \mathrm{z}(t)$

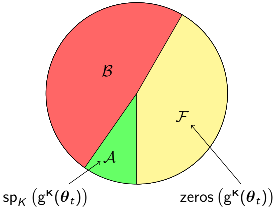


$\mathrm{supp}\left(\mathrm{g}^{\mathsf{K}}(\boldsymbol{\theta}_t)\right) = \mathcal{A} \cup \mathcal{B}$,
False Alarm Set $= \mathcal{B} \cup \mathcal{F}$

Recovery of aggregated gradient

- Sum of the W, K-sparse vectors has $K^{'} \in [K : KW]$ non-zero entries
- We choose to recover top-K entries from $\mathrm{y}(t) = \Phi(t)\mathrm{g}^{\mathsf{K}}(\boldsymbol{\theta}_t) + \mathrm{z}(t)$

The set of all indices $[N] = \mathcal{A} \cup \mathcal{B} \cup \mathcal{F}$

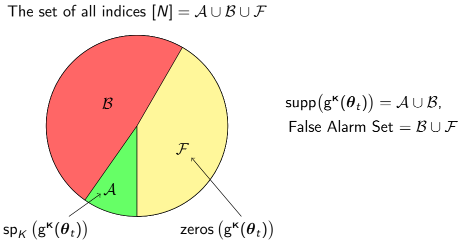


$\mathrm{supp}\left(\mathrm{g}^{\mathsf{K}}(\boldsymbol{\theta}_t)\right) = \mathcal{A} \cup \mathcal{B}$,
False Alarm Set $= \mathcal{B} \cup \mathcal{F}$

- Goal - estimate indices in $\mathcal{A}$ and corresponding values

Recovery of aggregated gradient

- Sum of the W, K-sparse vectors has $K^{'} \in [K : KW]$ non-zero entries
- We choose to recover top-K entries from $\mathrm{y}(t) = \Phi(t)\mathrm{g}^{\mathsf{K}}(\boldsymbol{\theta}_t) + \mathrm{z}(t)$

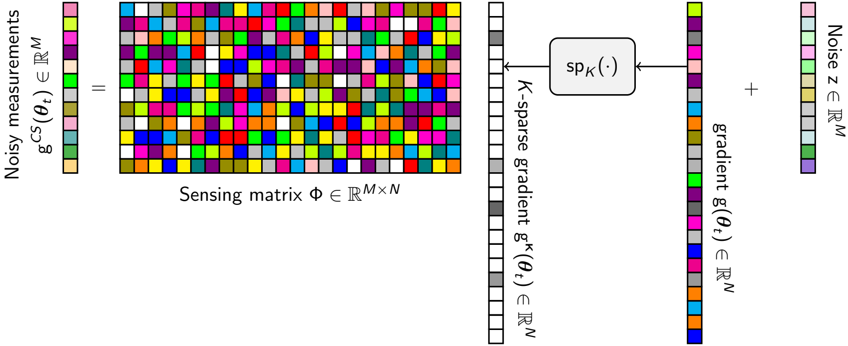


- Goal - estimate indices in $\mathcal{A}$ and corresponding values

$$\mathrm{y}(t) = \underbrace{\Phi(t)\mathrm{g}_{\mathcal{A}}}_{\text{top-}K\text{ entries}} + \underbrace{\Phi(t)\mathrm{g}_{\mathcal{B}}}_{\text{other non-zero}} + \mathrm{z}(t)$$

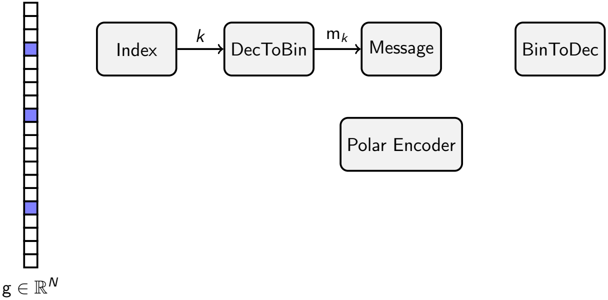
- What we want is task-specific compression

Limitations of current schemes



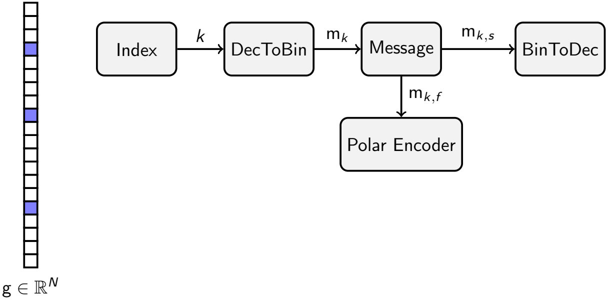
- Many go-to recovery algorithms have complexity $\mathcal{O}(MN)$
- State-of-the-art machine learning models can have very large N
- Our goal - construct a structured Φ and a recovery algorithm with lower complexity

PolarAir: Encoding the indices



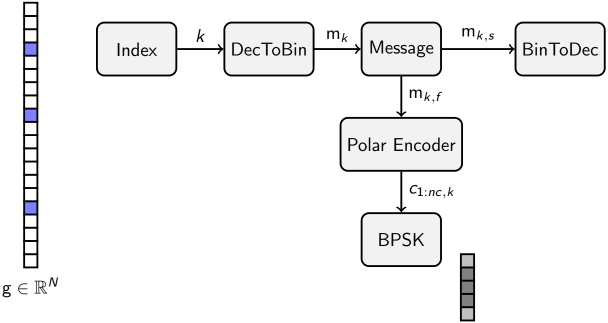
- Convert non-zero index k to a binary string m_k.
- Divide m_k into two parts: $\mathrm{m}_{k,f}, \mathrm{m}_{k,s}$.
- $\mathrm{m}_{k,s}$ is encoded and the coded bits are modulated using BPSK.
- Based on $\mathrm{m}_{k,s}$, a spreading sequence is chosen from the Master set A, where $\mathrm{A} = \{\mathrm{a}_1, \mathrm{a}_2, \ldots, \mathrm{a}_{2^{\mathrm{len}(\mathbf{m}_{k,s})}}\} \in \mathbb{R}^{L \times 2^{\mathrm{len}(\mathbf{m}_{k,s})}}$.
- The modulated bits are multiplied by spreading sequence $\mathrm{a}_j \in \mathrm{A}$.
- The active column ϕ_k is multiplied by g_k.

PolarAir: Encoding the indices



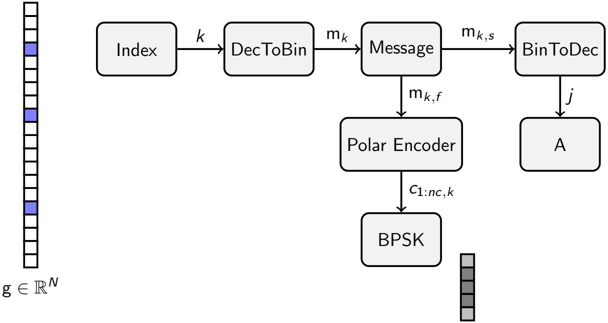
- Convert non-zero index k to a binary string m_k.
- Divide m_k into two parts: $m_{k,f}, m_{k,s}$.
- $m_{k,s}$ is encoded and the coded bits are modulated using BPSK.
- Based on $m_{k,s}$, a spreading sequence is chosen from the Master set A, where $A = \{a_1, a_2, \ldots, a_{2^{\mathrm{len}(\mathbf{m}_{k,s})}}\} \in \mathbb{R}^{L \times 2^{\mathrm{len}(\mathbf{m}_{k,s})}}$.
- The modulated bits are multiplied by spreading sequence $a_j \in A$.
- The active column ϕ_k is multiplied by g_k.

PolarAir: Encoding the indices



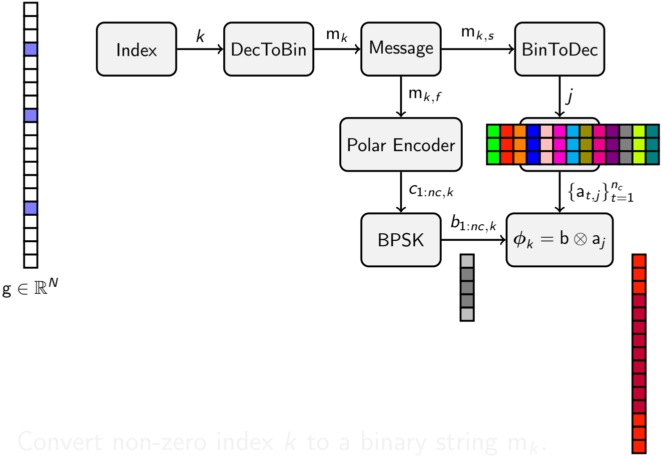
- Convert non-zero index k to a binary string m_k.
- Divide m_k into two parts: $\mathrm{m}_{k,f}, \mathrm{m}_{k,s}$.
- $\mathrm{m}_{k,s}$ is encoded and the coded bits are modulated using BPSK.
- Based on $\mathrm{m}_{k,s}$, a spreading sequence is chosen from the Master set A, where $\mathrm{A} = \{\mathrm{a}_1, \mathrm{a}_2, \ldots, \mathrm{a}_{2^{\mathrm{len}(\mathbf{m}_{k,s})}}\} \in \mathbb{R}^{L \times 2^{\mathrm{len}(\mathbf{m}_{k,s})}}$.
- The modulated bits are multiplied by spreading sequence $\mathrm{a}_j \in \mathrm{A}$.
- The active column ϕ_k is multiplied by g_k.

PolarAir: Encoding the indices



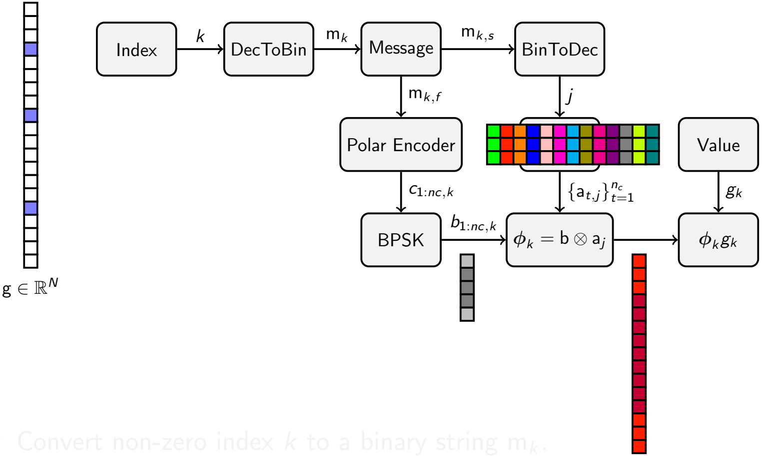
- Convert non-zero index k to a binary string m_k.
- Divide m_k into two parts: $m_{k,f}, m_{k,s}$.
- $m_{k,s}$ is encoded and the coded bits are modulated using BPSK.
- Based on $m_{k,s}$, a spreading sequence is chosen from the Master set A, where $A = \{a_1, a_2, \ldots, a_{2^{\mathrm{len}(\mathbf{m}_{k,s})}}\} \in \mathbb{R}^{L \times 2^{\mathrm{len}(\mathbf{m}_{k,s})}}$.
- The modulated bits are multiplied by spreading sequence $a_j \in A$.
- The active column ϕ_k is multiplied by g_k.

PolarAir: Encoding the indices



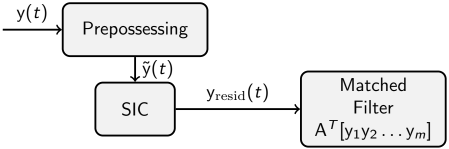
- Convert non-zero index k to a binary string m_k.
- Divide m_k into two parts: $m_{k,f}, m_{k,s}$.
- $m_{k,s}$ is encoded and the coded bits are modulated using BPSK.
- Based on $m_{k,s}$, a spreading sequence is chosen from the Master set A, where $A = \{a_1, a_2, \ldots, a_{2^{\mathrm{len}(m_{k,s})}}\} \in \mathbb{R}^{L \times 2^{\mathrm{len}(m_{k,s})}}$.
- The modulated bits are multiplied by spreading sequence $a_j \in A$.
- The active column ϕ_k is multiplied by g_k.

PolarAir: Encoding the indices



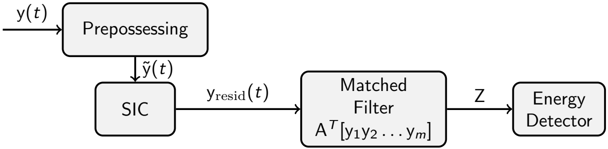
- Convert non-zero index k to a binary string m_k.
- Divide m_k into two parts: $\mathrm{m}_{k,f}, \mathrm{m}_{k,s}$.
- $\mathrm{m}_{k,s}$ is encoded and the coded bits are modulated using BPSK.
- Based on $\mathrm{m}_{k,s}$, a spreading sequence is chosen from the Master set A, where $\mathrm{A} = \{\mathrm{a}_1, \mathrm{a}_2, \ldots, \mathrm{a}_{2^{\mathrm{len}(\mathbf{m}_{k,s})}}\} \in \mathbb{R}^{L \times 2^{\mathrm{len}(\mathbf{m}_{k,s})}}$.
- The modulated bits are multiplied by spreading sequence $\mathrm{a}_j \in \mathrm{A}$.
- The active column ϕ_k is multiplied by g_k.

PolarAir: Recovery Algorithm



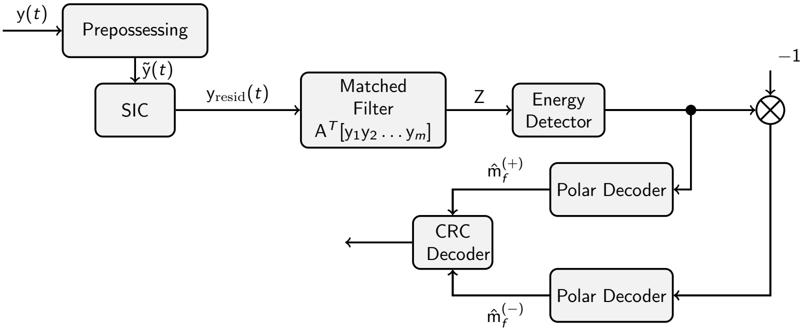
- Matched Filter: Estimate the symbols
- Energy Detector: Recover active sequence
- Unknown sign of gradient, use two Polar Decoders
- Encoder: Construct the Active Columns of Φ
- Least Squares: Estimate the values of the gradient

PolarAir: Recovery Algorithm



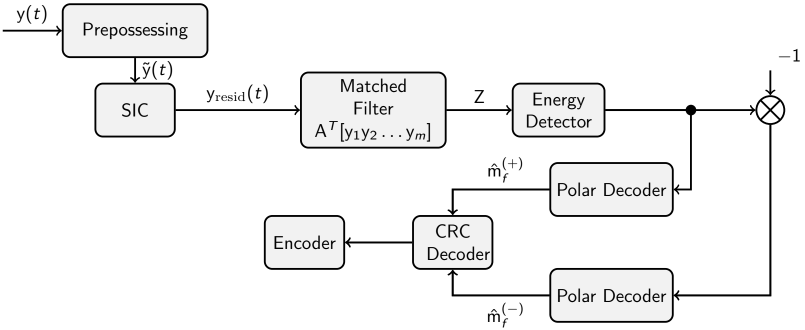
- Matched Filter: Estimate the symbols
- Energy Detector: Recover active sequence
- Unknown sign of gradient, use two Polar Decoders
- Encoder: Construct the Active Columns of Φ
- Least Squares: Estimate the values of the gradient

PolarAir: Recovery Algorithm



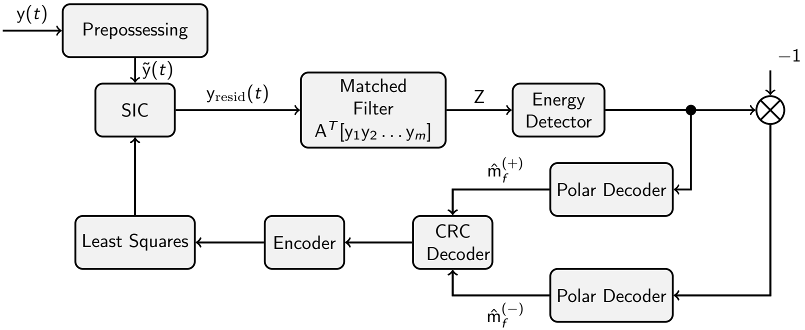
- Matched Filter: Estimate the symbols
- Energy Detector: Recover active sequence
- Unknown sign of gradient, use two Polar Decoders
- Encoder: Construct the Active Columns of Φ
- Least Squares: Estimate the values of the gradient

PolarAir: Recovery Algorithm



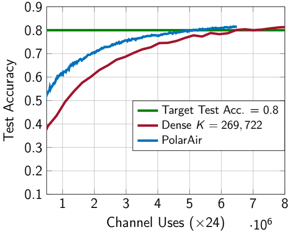
- Matched Filter: Estimate the symbols
- Energy Detector: Recover active sequence
- Unknown sign of gradient, use two Polar Decoders
- Encoder: Construct the Active Columns of Φ
- Least Squares: Estimate the values of the gradient

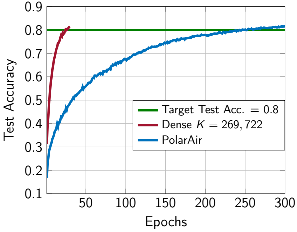
PolarAir: Recovery Algorithm



- Matched Filter: Estimate the symbols
- Energy Detector: Recover active sequence
- Unknown sign of gradient, use two Polar Decoders
- Encoder: Construct the Active Columns of Φ
- Least Squares: Estimate the values of the gradient

Results - Test Accuracy





- CIFAR-10 dataset, classification task, ResNet model
- $N = 269722$, we sparsified to $K = 270$, $W = 8$ users
- $\approx$ 30% less channel uses
- Much better compared to Dense (Genie)

Results - Probability of Missed Detection & False Alarm

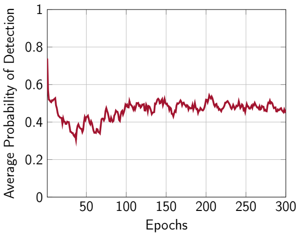


Figure: Average probability of Detection.

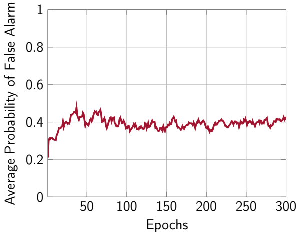


Figure: Average probability of False Alarm.

Results - Probability of Missed Detection & False Alarm

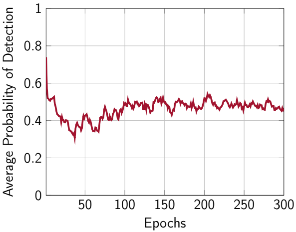


Figure: Average probability of Detection.

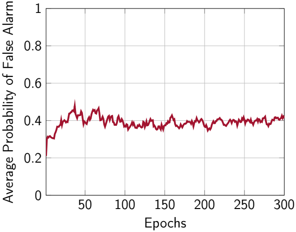
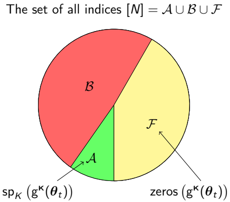


Figure: Average probability of False Alarm.



$\mathrm{supp}\big(\mathrm{g}^{\kappa}(\boldsymbol{\theta}_t)\big) = \mathcal{A} \cup \mathcal{B}$,
False Alarm Set $= \mathcal{B} \cup \mathcal{F}$

Results - behavior of the sum of sparse gradients

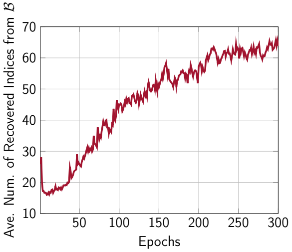


Figure: Average Number of Recovered Indices from $\mathcal{B}$.

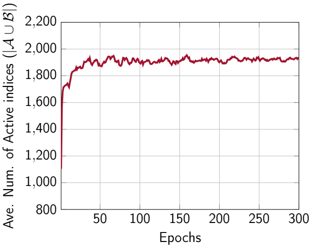
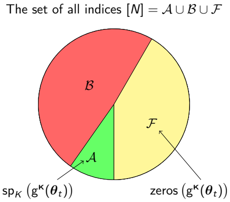


Figure: Average number of active indices as a function of epochs.



$\mathrm{supp}\big(\mathrm{g}^{\kappa}(\boldsymbol{\theta}_t)\big) = \mathcal{A} \cup \mathcal{B}$,
False Alarm Set $= \mathcal{B} \cup \mathcal{F}$

Conclusion

- PolarAir is a method to compress Deep Learning Models for OTAFL
- The complexity is $\mathcal{O}(K^3 + K^2 \log N)$
- $\approx$ 30% less channel uses compared to naive approach

Figure: Group's WebPage

Figure: Presentation

Figure: Source Code