Replication of “A Joint Learning and Communications Framework for Federated Learning over Wireless Networks” Paper

Fault Tolerant System Design course

Sharif University of Technology

Name: Amir Arsalan Yavari

Std. No: 402203497

۱. روش پیشنهادی، نقاط قوت و ضعف

۱.۱ انتخاب کاربران با توجه به packet error rate و بهینه‌سازی FL

روشی که مقاله در نظر گرفته تا خطای یادگیری فدرال را در شبکه‌ی بی‌سیم واقعی کمینه کند به این صورت است که مانند هر الگوریتم یادگیری فدرال، تعداد از کاربران را انتخاب می‌کند که در فرآیند یادگیری مشارکت کنند اما تفاوت در اینجاست که در روش ارائه شده کاربران به صورت تصادفی انتخاب نمی‌شوند چرا که به دلیل ویژگی‌های شبکه‌ی بی‌سیم نظیر packet error، کیفیت یادگیری به خودی خود کاهش می‌یابد. بنابراین بر اساس فاصله‌ی کاربر از BS و یا همچنین تاثیر نویز که با عامل قبلی بی‌ارتباط هم نیست، کاربران به صورت بهینه انتخاب خواهند شد زیرا عوامل مرتبط با شبکه بی‌سیم مانند نرخ خطای بسته‌ها و محدودیت‌های منابع بی‌سیم بر کیفیت یادگیری تأثیرگذارند. همچنین بر همین اساس اختصاص منابع به کاربران صورت خواهد گرفت که هدف آن کاهش تابع هزینه یادگیری فدرال است. از سویی دیگر پارامترهایی مثل توان به صورتی قرار است تنظیم شود که کمترین خطا را داشته باشیم تا شرط یاد شده تحقق یابد.

این روش با سه روش دیگر نیز مقایسه شده است؛ روش اول روشی است که انتخاب کاربران به صورت بهینه صورت گرفته اما تخصیص منابع به صورت تصادفی انجام می‌شود. روش دوم همان یادگیری فدرال معمول است که انتخاب کاربران و تخصیص منابع هر دو به صورت تصادفی انجام می‌شوند و روش سوم روشی است که انتخاب کاربران و تخصیص منابع به صورت بهینه انجام می‌شوند تا میزان خطای بسته‌ها در شبکه‌ی وایرلس کمینه شود اما اختصاص وزن‌ها در شبکه‌ی یادگیری ما به صورت بهینه برای کمینه کردن خطا انجام نمی‌شوند. تنها تفاوت روش سوم با روش مذکور در همین امر است.

۲.۱ نقاط قوت و ضعف

نقاط قوت این روش این است که در یک شبکه‌ی بی‌سیم یادگیری می‌تواند به صورت توزیع شده انجام شود و همچنین حریم خصوصی افراد نیز حفظ خواهد شد. از سویی شبیه‌سازی صورت گرفته تمامی مشکلات و ویژگی‌های شبکه‌ی بی‌سیم وایرلس را نیز لحاظ کرده است. همچنین با تخصیص بهینه‌ی منابع به بالاترین دقت خواهد رسید و زمان convergence را نیز کم خواهد کرد. از جمله روش‌های ضعف شاید بتوان به نحوه‌ی تخصیص منابع اشاره کرد که از الگوریتم Hungorian برای حل مسئله **تخصیص بلوک‌های منابع (RB)** و **انتخاب کاربران بهینه** استفاده شده است که اردر آن O(n^3) می‌باشد که می‌تواند برای شبکه‌هایی با تعداد زیاد کاربران و منابع، از نظر محاسباتی سنگین باشد و وابسته به شرایط مسئله ممکن است راهکار بهتری مانند الگوریتم‌های ابتکاری (Heuristic) یا یادگیری تقویتی (Reinforcement Learning) وجود داشته باشد. مقاله فرض می‌کند که در صورت بروز خطای بسته (Packet Error)، مدل محلی آن کاربر نادیده گرفته می‌شود و فرآیند یادگیری ادامه پیدا می‌کند. اما اگر تعداد زیادی از کاربران در یک دور از یادگیری نتوانند مدل خود را ارسال کنند، این می‌تواند دقت نهایی مدل فدرال را کاهش دهد. همچنین می‌توان گفت روش‌هایی مانند کدگذاری خطا (Error Correction Coding) یا افزایش تکرار ارسال داده (Retransmission) می‌توانند عملکرد بهتری ایجاد کنند.

۲. نحوه‌ی راه‌اندازی کدها

۱.۲ کد متلب

نویسندگان مقاله کد متلب مقاله‌ی مذکور را در آدرس https://github.com/mzchen0/Wireless-FL قرار داده‌اند که برای اجرای این کد تنها کافی است متلب را بر روی سیستم خود داشته، سپس مجموعه داده‌ی MNIST را نیز از آدرس https://yann.lecun.com/exdb/mnist بارگیری و از فرمت فشرده در آورده و در کنار فایل‌ها قرار دهید. و در پایان کد **FLMIN.m** را اجرا نمایید.

مشکل اساسی این است که این کد فقط الگوریتم‌ها را اجرا کرده و هیچ رسم نموداری ندارد. علاوه بر این موضوع تنها کد MNIST را داراست که برای مثال Linear Regression کدی وجود ندارد. در ادامه ما از پیاده‌سازی‌های موجود در این کد بهره می‌گیرم تا یک کد پایتون پیاده‌سازی کنم.

۲.۲ کد پایتون

مطابق با توضیحات یاد شده در مقاله و همچنین تنظیمات موجود در کد متلب یاد شده، کد پایتون معادل آن را نوشته‌ام. ساختار کدها به صورت زیر است:

```
codes
├── train_LR_plot_1.py
├── train_LR_plot_2.py
├── train_LR_plot_3.py
├── train_LR.py
├── train_mnist_plot_1.py
├── train_mnist_plot_2.py
├── train_mnist_plot_3.py
└── train_mnist.py
```

فایل train_mnist.py همان فایل معادل کد متلب است اما فایل‌های plot_1، plot_2 و plot_3 به ترتیب نمودارهای بدست آمده در مقاله را قرار است برای ما منطبق با تنظیمات و توضیحات داده شده رسم کنند. همچنین برای شبیه‌سازی Linear Regression نیز به همین ترتیب یک فایل train_LR.py دارم که کد به صورت کلی است و فایل‌های plot_1، plot_2 و plot_3 به ترتیب نمودارهای مقاله را قرار است رسم کند. در ادامه، در قسمت نتایج به بررسی نتایج بدست آمده و مقایه‌ی آنها با نتایج گزارش شده در مقاله خواهیم پرداخت.

برای اجرای فایل‌های مذکور توصیه می‌شود cuda بر روی سیستم راه‌اندازی شده، سپس فایل requirements.txt را با استفاده از pip نصب نموده و پس از آن می‌توان تک تک این فایل‌ها را با python اجرا نمود. برای این منظور می‌توانید به تابع main فایل‌ها توجه فرمایید تا آرگومان‌های اجرایی را نیز در صورت نیاز هنگام اجرای کد در نظر بگیرید. در پایان برای مشاهده‌ی نتایج لازم است دستور زیر را در خط فرمان وارد نمایید.

```
tensorboard --logdir=./runs
```

۳. توضیح کد

کدها به این صورت نوشته شده‌اند که منطقی یکسانی دارند من به عنوان مثال train_mnist را توضیح میدهم و کدهای train_mnist_plot_1..3.py همان کدها هستند اما برای بدست آورد نمودارهای مقاله در شرایط با user متفاوت، با تعداد نمونه متفاوت، با RB متفاوت و ... اجرا شده‌اند. همچنین کدهای train_LR هم مانند همین کد هستند تنها در شبکه‌ی عصبی و نمونه‌های یادگیری تفاوت دارند.

اولین نکته‌ای که در کد با آن مواجه می‌شوید دیکشنری config است که hyperparameter های کد در آن قرار داده شده‌اند. در ادامه تنظیمات اولیه مانند seed و device که کد روی آن اجرا شود تنظیم شده. سپس کلاس شبکه عصبی را داریم که توابع initialization و forward را دارد که فرآیند آموزش شبکه‌ی عصبی لوکال را پیاده‌سازی کند. در ادامه مبتنی بر کد متلب مقاله و در برخی موارد با استفاده از متن مقاله، توابع متفاوتی نظیر: بدست آوردن خطای بسته، محاسبه‌ی نویز، محاسبه‌ی انرژی و محاسبه‌ی rate کانال را داریم که کد خاصی ندارند و صرفا بازی ریاضی است؛ به طوری که برای محاسبه‌ی هر یک چند پارامتر در هم ضرب و تقسیم و جمع و منهی خواهند شد. تابع init_weights برای وزن‌دهی اولیه به شبکه‌های عصبی استفاده و تابع evaluate برای ارزیابی هر راند از اجراست.

در ادامه مهم‌ترین قسمت کد یعنی تابع train را داریم که فرآیند آموزش در این تابع رخ می‌دهد. از خطوط اولیه‌ی تابع که شامل مقدار دهی برخی موارد نظیر تعداد resource block ها، استراتژی‌های مختلف، نویز هر کاربر و همچنین نرمالایز کردن دیتاست است بگذریم به یک دور تکرار خواهیم رسید (منظور من خط 149 است) که این تعداد دور اجرای الگوریتم فدریشن را نشان می‌دهد. در هر دور تمامی مدل‌ها (استراتژی‌های proposed و baselineها) که کد آنها منطبق با توضیحات ارائه شده در قسمت روش پیشنهادی است تنظیم می‌شوند و پس از آن حلقه‌ی یادگیری محلی بر روی دیتاست برای هر یک اجرا می‌شود. بعد از هر دور یادگیری محلی یک اگریگیشن به صورت average انجام می‌شود تا عمل federation صورت گیرد و در نهایت accuracy هر کدام محاسبه می‌شود. در پایان بعد از اجرا شدن تمامی این دورها، نمودار دقت بر حسب تعداد دور نمایش داده می‌شود.

در پایان نیز قسمت main را داریم که در اکثر کدها تعدادی آرگومان ورودی به هنگام اجرا می‌گیرد که مقادیر پیش‌فرض نیز برای آنها در نظر گرفته شده است و بدون پاس‌دادن هیچ مقداری متناسب با نموداری که قرار است اجرا شود تنظیم شده‌اند.

شاکله‌ی اصلی تمامی کدها بر همین اساس محوریت دارد و توضیح ریز جزئیات نیز در جلسه‌ی ارائه‌ی حضوری داده خواهد شد.

۴. چالش‌ها و مشکلات

۱.۴ NaN شدن خطا

مشکل اولی که من در پیاده‌سازی با آن مواجه شدم این بود که مقدار loss زیاد و زیاد می‌شد تا جایی که به NaN می‌رسید. برای رفع این مشکل من تدابیر زیر را در نظر گرفتم:

- مقدار نویز که بسیار زیاد می‌شد را به صورت `sinr = np.clip(sinr, 1e-6, 1e6)` محدود کردم. همچنین برای مدل هم از `torch.nn.utils.clip_grad_norm` استفاده کردم.
- در نظر گرفتن مقدار کم (epsilon = 1e-6) برای زمان‌هایی که عدد ممکن بود صفر شود و در ادامه در محاسبات عبارت تقسیم به صفر ممکن بود ایجاد شود.

۲.۴ عدم وجود اطلاعات لازم در متن مقاله

متاسفانه برخی از موارد در مقاله توضیح داده نشده بود. برای مثال TABLE II در متن مقاله وجود داشت اما هیچ اشاره به هیچ‌کدام از پارامترهای این جدول نشده بود و اصلا هیچ کجای متن مقاله نیز اشاره‌ای به این جدول نشده بود. من مقادیر را بر اساس کد متلب سعی کردم پیدا کنم و جای‌گذاری کنم. همچنین تعداد لایه‌ها نیز ذکر نشده بود که من یک FNN با یک لایه‌ی مخفی با تعداد نورون گزارش شده در نظر گرفتم. همچنین میزان **batch size** و عوامل این چنینی هم ذکر نشده بودند که من مقادیر تقریبا منطقی (۸) را در نظر گرفتم. علاوه بر این با توجه به اینکه بسیاری از داده‌ها (مخصوصا داده‌های Linear Regression) به صورت تصادفی تولید می‌شوند، هیچ seed مشخصی اعلام نشده بود فلذا نتایج متفاوتی بدست می‌آمد. سوای از این الگوریتم Munkres در متن مقاله تنها گفته شده که از آن استفاده شده است و در کد متلب نیز کل آن پیاده‌سازی شده بود که ممکن است پیاده‌سازی خاص منظوره‌ای بوده باشد اما من از کتابخانه‌ی munkres پایتون استفاده کردم. علاوه بر مشکلات یاد شده، محیط شبیه‌سازی و توان پردازشی ذکر نشده بود که در کل من بر روی سیستم خودم با یک RTX 3060 ti و core i7 Gen 11 نتایج را بدست آوردم. نهایتا برخی از نتایج کمی متفاوت بدست آمدند که در ادامه به آنها خواهیم پرداخت.

۳.۴ زمان اجرای هر کدام از کدها

با توجه به تنظیمات و پارامترهای مسئله هر دور از اجرا زمان زیادی را به خود اختصاص می‌دهد. برای مثال یکی از نمودارها دقت مدل‌های proposed و baseline1..3 را بر اساس تعداد کاربران در بازه‌ی ۳ تا ۱۸ با step=3 رسم کرده که این بدین معنی است که ۴ مدل هر کدام سه بار (به دلیل وجود پارامتر average) برای تعداد مختلف کاربران اجرا خواهند شد (یعنی ۷۲ دور اجرا تنها برای یکی از نمودارها) و با توجه به وجود الگوریتم munkres که در کتابخانه‌ی پایتون به صورت CPU Based پیاده‌سازی شده، سربار این کد در همین قسمت اجرای munkres است و برای هر نمودار بیش از یک ساعت زمان لازم است.

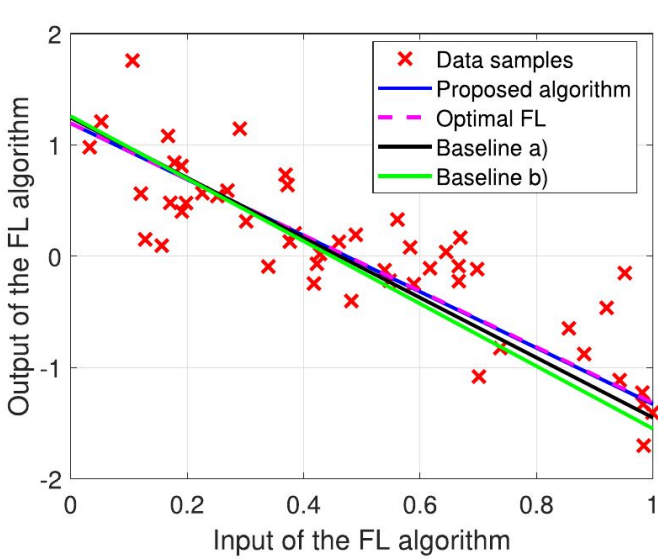
۵. نتایج

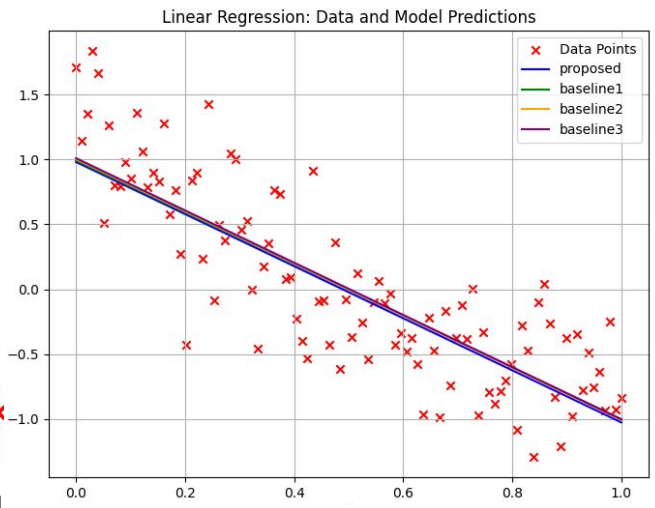
۱.۵ نتایج Linear Regression

مطابق با توضیحات گفته شده در قسمت چالش‌ها، به دلیل عدم عنوان برخی از پارامترها برای یادگیری، نتایج بدست آمده به میزان کمی متفاوت با نتایج مورد انتظار بوده‌اند. در ادامه تمامی نمودارها را نمایش خواهیم. نمودارهای سمت راست، نتایج بدست آمده توسط کد من و نمودارهای سمت چپ نمودارهای مقاله هستند.

۱.۱.۵ نمودار خط رگرشن/داده‌های مدل‌ها

با توجه به اینکه seed مشخصی برای نمونه‌ها مشخص نشده بود، من هم seed را برابر 0 در نظر گرفتم اما نتایج در اغلب اوقات به همین صورت بود. نتیجه‌ی بدست آمده این چنین بود که loss روش ارائه شده (روش proposed) کمتر از loss دیگر روش‌ها بود. نتایج و نمودارها در پوشه‌ی run پیوست شده در کنار فایل pdf قرار دارند که آن را با استفاده از tensorboard می‌توانید اجرا نمایید.

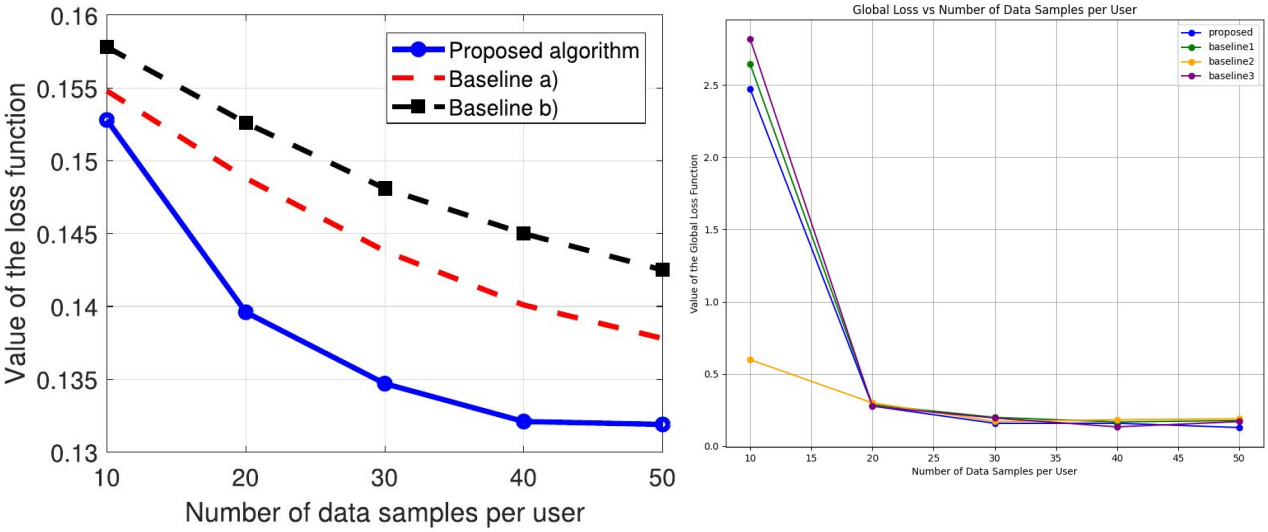




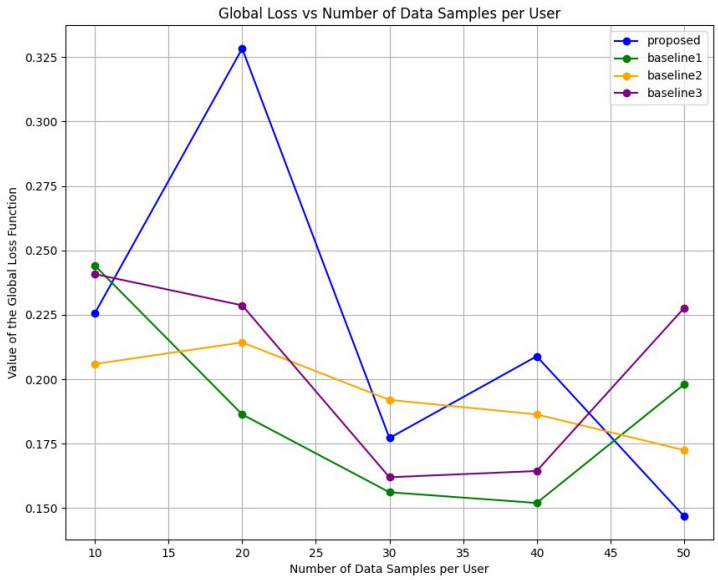
Proposed	0.1308
Baseline 1	0.1476
Baseline 2	0.1817
Baseline 3	0.2277

۲.۱.۵ نمودار loss بر حسب تعداد داده‌های کاربران برای روش‌های متفاوت

برای نمودار دوم نتایج من منطبق با نتایج ارائه شده‌ی مقاله نبود. اکثر نمودارهایی که من بدست می‌آوردم به صورت زیر بود که در نهایت امر روش proposed خوب بود و حتی در تعداد ۳۰ تا هم بنا بر ادعای مقاله نتیجه‌ی مورد نظر را بدست می‌آورد اما روش Baseline 2 که البته چون تمامی پارامترهای را random انتخاب می‌کند در راند اول با تعداد سمپل کمتر loss کمتری داشته اما در ادامه بجز مقدار 40 در تمامی مقادیر روش proposed مطابق نتایج مقاله کمترین loss را داشته و لاس نهایی هم نزدیک به گزارش مقاله است.

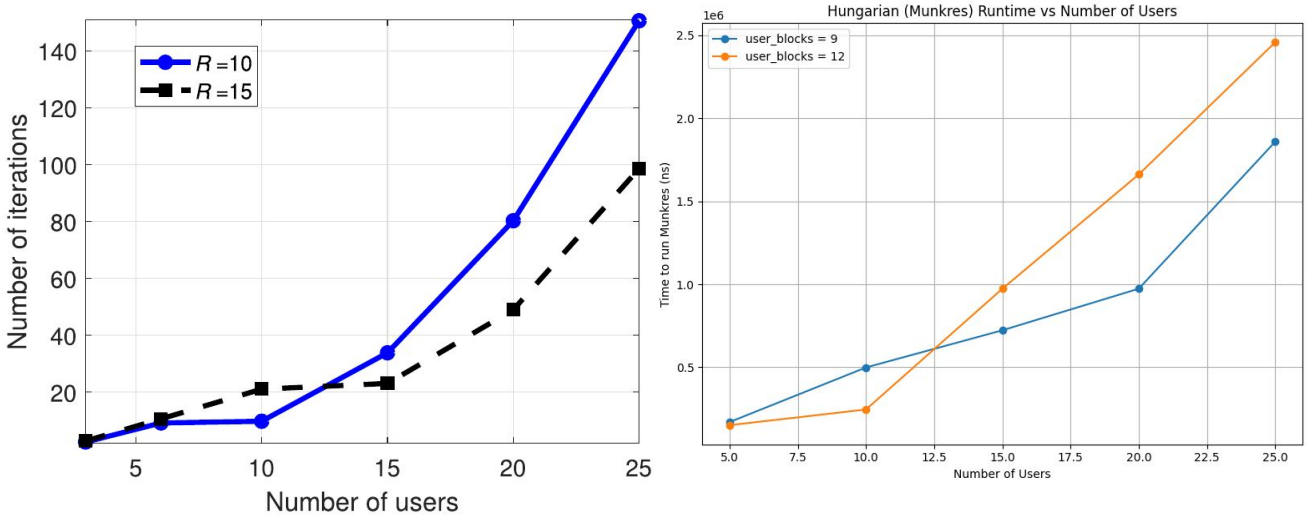


نکته‌ای که حائز اهمیت است این بود که با تغییر hyper parameter ها نتایج متفاوت بدست می‌آید؛ برای مثال می‌توان به تصویر زیر اشاره کرد. بنابراین برای نتایج ذکر شده در مقاله به نظر دور از واقعیت نیست اما باید شرایط دقیقا همان باشد.



۳.۱.۵ نمودار تعداد iteration برای تعداد کاربران متفاوت به منظور یافتن RB بهینه توسط الگوریتم Hungorian

برای شبیه‌سازی این قسمت برای آنکه من از کتابخانه‌ی munkres استفاده کرده بودم تعداد iteration را نمی‌توانستم محاسبه کنم؛ بجای این کار من زمان مورد نیاز برای محاسبات را اندازه گرفتم و به نانو ثانیه گزارش می‌کنم که نتیجه‌ی مشابهی خواهیم داشت. همچنین برای R=10 و R=15 هم مقادیر channel_interference را نداشتم (جایی از مقاله ذکر نشده) و همچنین در کد متلب نیز وجود نداشت. برای همین از مقادیر R=9 و R=12 که در کد متلب بودند استفاده کردم (البته این مقادیر به نظر مقدار تصادفی سعودی به نظر می‌رسند اما بهرحال به دلیل اینکه من هر مقداری در نظر بگیرم نتیجه‌ی متفاوتی بدست می‌آورم تصمیم گرفتم از ۹ و ۱۲ که مقادیر آنها را دارم استفاده کنم). بر این اساس نتیجه‌ی زیر بدست آمد که تقریبا متناسب با اعداد گزارش شده بود.



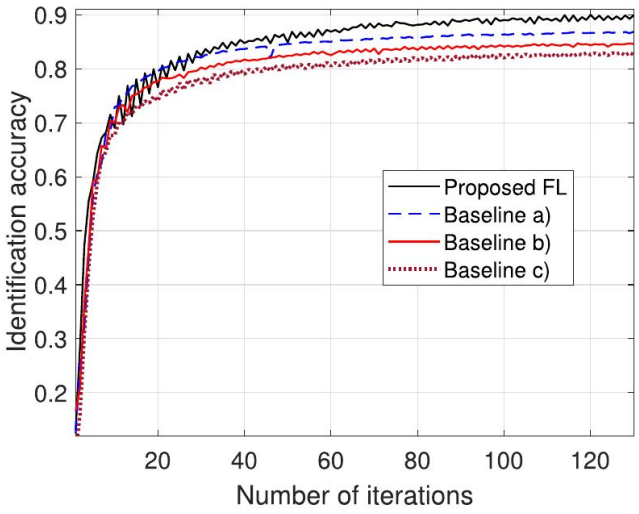
نکته‌ی حائز اهمیت این است که برای این نمودار من دیگر محاسبات یادگیری فدرال را انجام ندادم چون فرآیند کشف RB بهینه پیش از فرآیند یادگیری صورت می‌گیرد و این نمودار تنها گزارشی از زمان مورد نیاز برای RB بهینه است. مقادیر متفاوت برای R ها را هم از کد متلب بهره بردم.

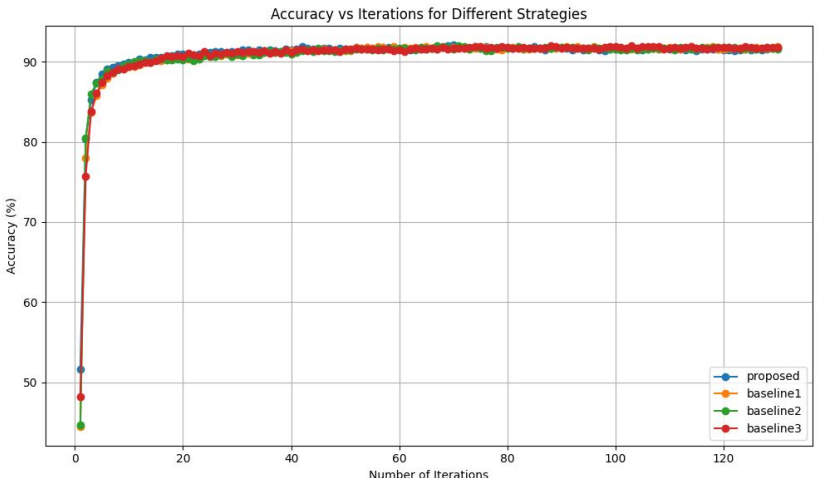
۲.۵ نتایج MNIST

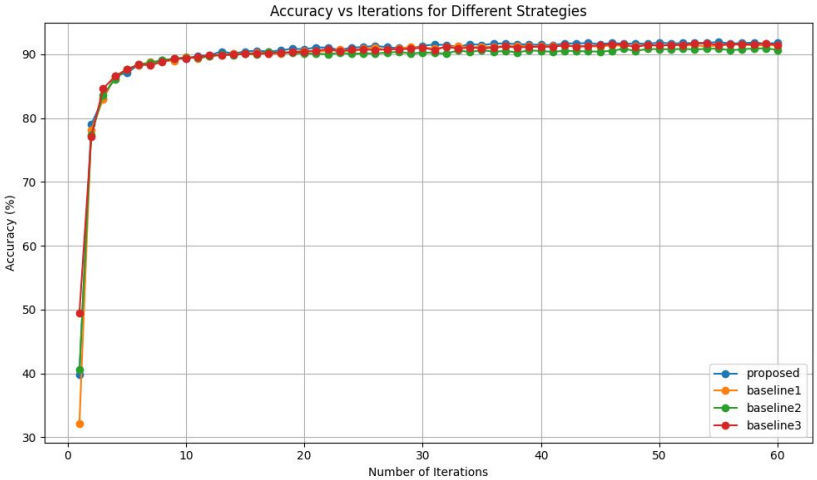
مطابق با توضیحات گفته شده در قسمت چالش‌ها، به دلیل عدم عنوان برخی از پارامترها برای یادگیری، نتایج بدست آمده کمی متفاوت با نتایج مورد انتظار بوده‌اند. در ادامه تمامی نمودارها را نمایش خواهیم. نمودارهای سمت راست، نتایج بدست آمده توسط کد من و نمودارهای سمت چپ نمودارهای مقاله هستند.

۱.۲.۵ نتایج accuracy مدل‌های متفاوت بر اساس تعداد iteration

مطابق با نتیجه‌ی بدست آمده هر ۴ مدل دقت‌های تقریبا یکسانی داشته‌اند اما مدل Proposed در ابتدا دقت بیشتری داشته است (در تعداد user کمتر) و همچنین با توجه به افزایش تعداد کاربران سریع‌تر به convergence رسیده است. البته شایان ذکر است که به دلیل اینکه بنده از **clipping** استفاده کردم که در قسمت چالش‌ها و مشکلات توضیح دلیل استفاده را داده‌ام تغییرات دقت کمتر و نمودار حالت نرم‌تری پیدا کرده است. در نهایت با مشاهده‌ی مقدار accuracy می‌توان به این نتیجه رسید که نتایج رفتار مشابه نتایج مقاله داشته است اما نتایج مقاله به 0.9 رسیده‌اند اما من برای تمامی مدل‌های مقدار بیش از 90 بدست آوردم.



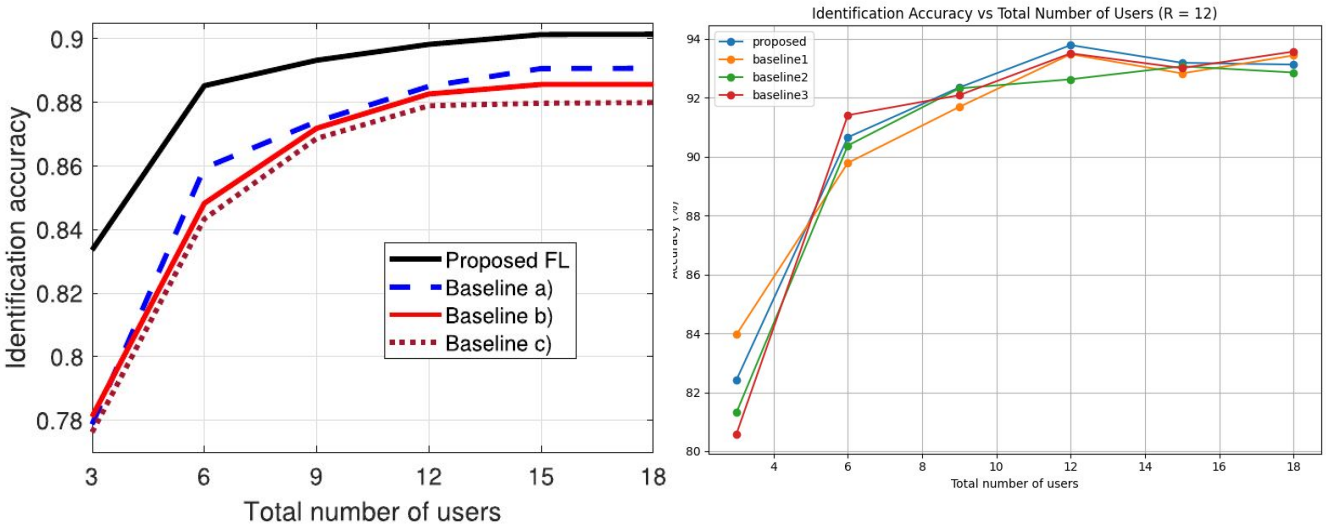




همانطور که در شکل سمت راست قابل مشاهده است، الگوریتم proposed عملکرد بهتری نسبت به دیگر الگوریتم‌ها داشته. در این شکل من تعداد دور اجرا را بر روی 60 تنظیم کردم تا این موضوع واضح شود.

۲.۲.۵ نتایج accuracy مدل‌های متفاوت بر اساس تعداد user ها

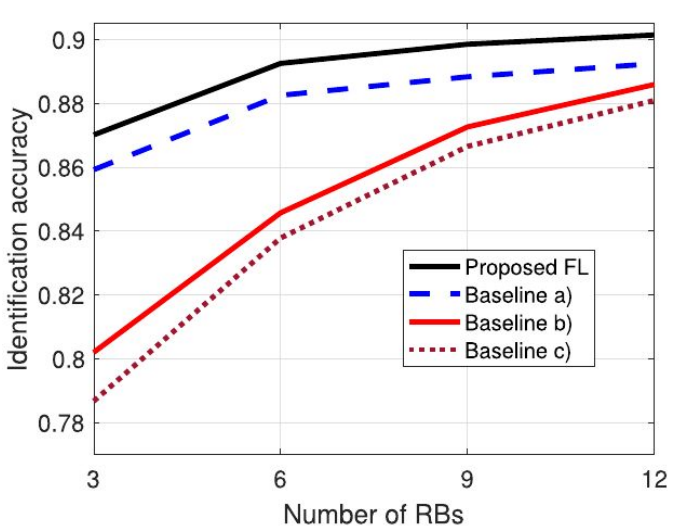
مطابق با تصویر زیر مدل proposed در تعداد user معادل 18 دقت کمتری بدست آورده است. اما از آنجا که مجددا مقاله هیچ صحبتی در مورد دیگر پارامترها نکرده (برای مثال در مورد تعداد iteration حرفی نزده و proposed در تعداد iteration کمتر نتایج بهتری بدست می‌آورد می‌توان انتظار داشت که در شیب نتایج متناظر با نمودار گزارش شده در مقاله باشد. اما نکته‌ی حائز اهمیت در این نتیجه این است که دقت بدست آمده در آزمایش من از دقت گزارش شده در مقاله بسیار بیشتر است (مقاله به دقت 0.9 رسیده ولی من مقدار 0.94 را بدست آورده‌ام). لازم به ذکر است نتایج زیر را با تعداد iteration معادل 130 بدست آورده‌ام.

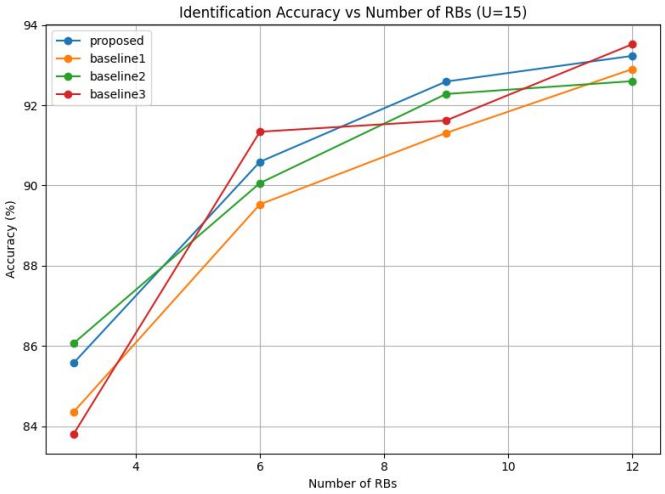


- البته این نتیجه می‌تواند به دلایل دیگری هم بوجود آمده باشد اما حدس من بر این است که تعداد iteration بیشترین تاثیر را بر روی نتیجه‌ی بدست آمده خواهد داشت.

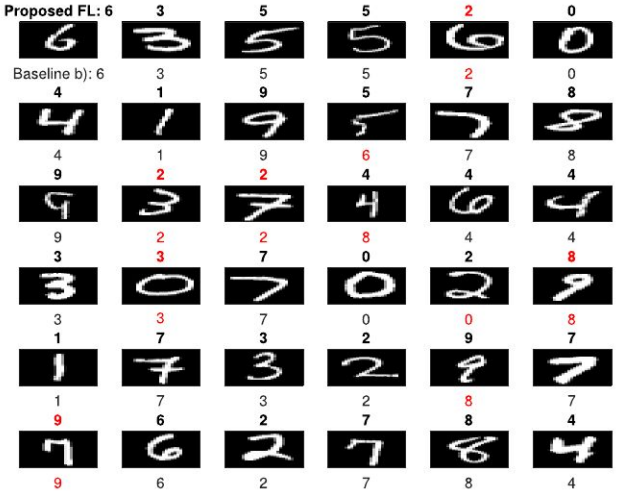
۳.۲.۵ نتایج accuracy مدل‌های متفاوت بر اساس تعداد RB ها

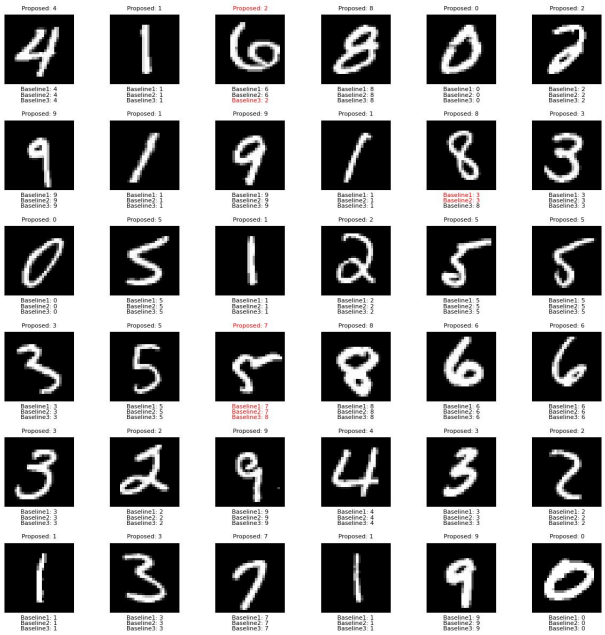
مطابق با تصویر زیر مدل proposed در اغلب RB های متفاوت دقت بیشتری نسبت به سایر مدل‌ها داشته است. اما مجددا مشکل قبل را داریم. رفتار مدل‌ها و همچنین دقت‌های آنها شبیه به اعداد گزارش شده‌ی مقاله‌ی اصلی است (دقت‌ها کمی بیشترند در پیاده‌سازی بنده) اما baseline 3 رفتار تصادفی‌ای داشته و سعود و نزول آن نسبت به دیگر روش‌ها متفاوت است به طوری که از کمتری دقت به یکباره به بیشترین دقت می‌رسد و این رفتار باز هم تکرار می‌شود. بنده مشکل این موضوع را متوجه نشدم اما به طور کلی می‌توان گفت به نتایج شبیه به مقاله‌ی اصلی رسیدم.





۴.۲.۵ نتایج پیش‌بینی ۳۶ تصویر MNIST توسط مدل‌ها





بر اساس نمودارهای گزارش شده در مقاله نمودار رو برو نیز وجود داشت که اطلاعات خاصی به ما نمی‌دهد اما صرفا شهودی از میزان خوب کار کردن مدل‌ها بدست خواهیم آورد. این ۳۶ نمونه به طور تصادفی از مجموعه داده‌ی MNIST انتخاب می‌شوند. من علاوه بر Baseline 2 موارد Baseline های 1 و 3 را نیز گزارش کردم. مطابق با نتایج بنده موارد بوده که proposed درست تشخیص داده اما Baseline 2 غلط تشخیص داده و برعکس مواردی هم بوده که Baseline 2 درست تشخیص داده و proposed اشتباه تشخیص داده که این مورد را در نتایج گزارش شده‌ی مقاله نمی‌بینیم. بر اساس تصاویر روبرو، تصویر بالا تصویر گزارش شده در مقاله است و تصویر پایین نتیجه‌ی شبیه‌سازی بنده است.

در صورت بروز هرگونه مشکل و هرگونه سوال به آیدی تلگرام زیر پیام دهید.

@AmirArsalanYavari

با تشکر