دانشگاه صنعتی خواجه نصیرالدین طوسی

به نام مهربان‌ترین مهربانان

پروژه پایانی درس مبانی سیستم‌های هوشمند

موضوع:

تخمین وضعیت شارژ باتری بر اساس شبکه عصبی و عصبی-فازی

استاد درس:

دکتر علیرضا فاتحی

دانشجو:

ریحانه هادی پور

زمستان ۱۴۰۰

فهرست:

چکیده:

همراه با گسترش مرزهای دانش، نیاز به ابزار پرتابل علی الخصوص ابزارهای محاسباتی هرچه بیشتر احساس می‌شود، اما چالش اصلی در این راستا داشتن یک منبع انرژی پرتابل و قابل اعتماد و البته برنامه ریزی است. به همین دلیل سعی بر شناسایی مدل باتری بعنوان یک منبع انرژی قابل حمل با هدف تخمین دقیق از میزان شارژ باقیمانده داریم، تا به کاربر این امکان را بدهیم تا با اطمینان خاطر بیشتری برای استفاده از دستگاه پرتابل خود برنامه ریزی کند. با توجه به رفتارهای پیچیده دینامیکی یون‌ها در باتری از مدل‌های عصبی و عصبی- فازی برای شناسایی استفاده شده است. و در انتها نتایج حاصله را با استفاده نتایج مدل ANFIS بعنوان یک مدل قابل اعتماد در شناسایی مقایسه می‌کنیم.

مقدمه:

توسعه سیستم تشخیص باتری هوشمند یک ضرورت برای صنعت حمل و نقل در آینده است. این فن آوری به صورت بالقوه تاثیر عمیقی در صنایع دیگر مانند لوازم الکترونیکی قابل حمل نیز خواهد شد. با وجود آنکه باتری به طور فریب دهنده‌ای بسیار ساده به نظر می‌رسد، یک سیستم پیچیده و غیر خطی متشکل از تعامل فرآیندهای فیزیکی و شیمیایی می‌باشد. در حال حاضراساس روش شناسایی باتری، اندازه گیری پاسخ شارژ و شکل موج ولتاژ باتری ناشناخته می‌باشد. این شکل موج ولتاژ حاوی تمام اطلاعات مورد نیاز برای شناسایی دقیق باتری می‌باشد. وضعیت شارژ (SOC)، که پارامتری برای توصیف مقدار انرژی موجود در باتری است، یک ابزار قدرتمند در مدیریت باتری می‌باشد ولی تخمین آن چالش برانگیز است و محاسبه‌ی online مقدار soc در سیستم‌های استفاده کننده از باتری‌های ذخیره کننده‌ی انرژی بسیار با اهمیت است. ما برای حل مشکل تخمین دقیق SOC از روش‌های شبکه عصبی و شبکه عصبی فازی برای تخمین SOC استفاده می‌کنیم.

تعریف **SOC** :

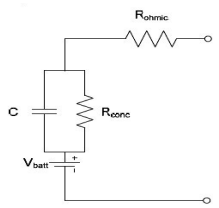
در بیشتر سیستم‌هایی که از یک باتری استفاده می‌کنند، وضعیت شارژ (SOC)، یک پارامتری است که مقدار شارژ باتری دارد و یا مقدار زمانی که باتری می‌تواند دوام بیاورد را نشان می‌دهد و می‌تواند کلید حل مشکلات ما باشد. قبل ازاندازه‌گیری SOC، ما باید تعریف دقیقی ازSOC داشته باشیم. ساده‌ترین و مرسوم‌ترین تعریف به صورت زیر می‌باشد:

$$SOC = (SOC_0 - \frac{\eta \int_0^t i(t)dt}{C_a}) \%100$$

این تعریف برای حالت تخلیه‌ی باتری (دشارژ) می‌باشد و SOC_0 نشان دهنده‌ی وضعیت شارژ در لحظه اول شروع به دشارژ کردن می‌باشد. SOC_0 ممکن است حالت شارژ کامل(۱۰۰٪) و یا هر مقدار شارژ باشد، $i(t)$ نشان دهنده‌ی جریان دی شارژ می‌باشد، C_a ظرفیت تخلیه باتری می‌باشد که در ابتدا کاملا شارژ شده است و تابعی از جریان تخلیه است و η ضریب ثابت می‌باشد. (عامل درجه حرارت در این پژوهش نادیده گرفته شده)

مدار معادل باتری:

هسته پژوهش ما بر این اساس است که هر باتری ناشناخته‌ای می‌تواند به صورت یک مدار معادل خطی مدل شود، این کار ما بر اساس کار مشابه در[۲]،[۳] و [۴] می‌باشد. این مدار معادل، مدلی برای توجیه شکل موج ولتاژ و تعیین کمی چهار پارامتر R_{ohmic},R_{conc},C,V_{batt} می‌باشد. این مدار معادل در شکل(۱) نشان داده شده است.



شکل ۱– مدار معادل باتری

این چهار پارامتر نشان دهنده بخش‌های مختلف از باتری است. R_{ohmic} نشان دهنده الکترود و مقاومت بسته‌ای باتری، R_{conc} نشان دهنده مقاومت داخلی باتری است که حداکثر جریانی که یک باتری می‌تواند ارائه دهد را تعیین می‌کند. C خازن باتری است که از اتصال مجموعه‌ای از لایه‌های تشکیل دهنده‌ی سلول‌های باتری تشکیل می‌شود و نشان دهنده مقدار معینی از بار الکتریکی ذخیره شده در داخل باتری است. V_{batt} نشان دهنده ولتاژ بی باری باتری است.

برای به دست آوردن پارامترهای مدل آزمون‌های مختلفی ارایه شده است. یکی از این روش ها [۵] این است که ابتدا یک منبع جریان با یک جریان ثابت شارژ I (A) را برای یک زمان محدود به ترمینال‌های مدار معادل شکل نشان داده شده متصل می‌کنیم. با استفاده از افزایش نمایی در ولتاژ و با در نظر گرفتن سه نمونه پشت سر هم از ولتاژ ترمینال در $t=T_1, T_2, T_3$ ثابت زمانی مدار با استفاده از رابطه‌ی زیر شناسایی شود:

جایی که V_1, V_2, V_3 نشان دهنده‌ی نمونه‌های متوالی از ولتاژ ترمینال و Δt نشان دهنده‌ی زمان بین دو نمونه برداری می‌باشد.

جمع آوری داده ها به صورت عملی :

به منظور شناسایی باتری نیازمند داده‌های کافی می‌باشیم. جهت جمع آوری داده یک شارژر که قابلیت شارژ و دشارژ باتری تهیه شده و از نتایج پژوهشی که قبلا انجام شده در این پروژه استفاده خواهیم کرد و جمع آوری داده‌ها جز اهداف ما نیست.

شناسایی توسط شبکه عصبی

آموزش شبکه عصبی:

در ابتدا وزن‌ها و بایاس‌ها را بصورت رندوم در بازه [5.0 -5.0] مقدار دهی می‌کنیم، علت این مو ضوع این است که نرون‌های ما از قسمت خطی شروع به فعالیت کنند و به قسمت غیر خطی بروند. شبکه می‌تواند برای تقریب توابع، تشخیص الگو و یا طبقه‌بندی الگوها مورد استفاده قرار گیرد. فرایند آموزش به یک سری مثال‌ها از رفتار مورد انتظار شبکه نیاز دارد که شامل ورودی شبکه P و هدف T می‌شود. در طول فرایند آموزش وزن‌ها و بایاس‌ها تنظیم می‌شوند تا تابع کارایی شبکه (NET.PERFORMFUN) حداقل شود. تابع کارایی پیش فرض برای شبکه‌های MSE FEEDFORWARD، می‌باشد که منطبق بر همان تابع عملکردی است که در کلاس بحث کردیم.

در ادامه این بخش الگوریتم‌های آموزش مختلف را در شبکه‌های FEDDFORWARD مورد بررسی قرار می‌دهیم. تمامی این توابع از شیب تابع کارایی برای تنظیم وزن‌ها و بایاس‌ها استفاده می‌کنند. این شیب با استفاده همان تکنیک BACKPROPAGATION تعیین می‌شود. محاسبه BACKPROPAGATION از قانون زنجیره‌ای در حساب دیفرانسیل قابل محاسبه است. الگوریتم اساسی BP، وزن‌ها را در جهت شیب منفی اصلاح می‌کند، که در ادامه شرح داده خواهد شد.

الگوریتم BACKPROPAGATION :

الگوریتم‌های مختلفی برای BACKPROPAGATION یا همان BP وجود دارد که در این بخش به تعدادی از آن‌ها تا حدی اشاره خواهیم کرد. در ساده ترین پیاده سازی یادگیری BP، وزن‌ها و بایاس‌ها در جهتی که تابع کارایی کاهش می‌یابد یعنی خلاف شیب آن به روز می‌شود. یک تکرار از این الگوریتم را میتوان به صورت زیر نوشت:

$$X_{k+1} = X_k - £_K G_K$$

که X_K بردار فعلی وزن‌ها و یا بایاس‌ها است و G_K شیب فعلی و $£_K$ سرعت یادگیری (LEARNING RATE)

دو روش مختلف برای این الگوریتم پیاده سازی شده است: روش گام به گام (INCEMENTAL) و روش دسته‌ای (BATCH)

در روش گام به گام وزن‌ها و بایاس‌ها بعد از هر اعمال ورودی به روز می‌شوند در حالی که در روش دسته‌ای پس از اعمال تمام ورودی ها عملیات به روز رسانی انجام خواهد شد.

- آموزش دسته‌ای کاهش شیب : همانگونه که در کلاس درس مطرح شد در روش دسته‌ای وزن‌ها و بایاس‌ها پس از اعمال تمامی اعضای مجموعه آموزش UPDATE می‌شوند. شیبها محاسبه شده و برای هر ورودی با هم جمع می‌شوند تا در نهایت میزان وزن و بایاس‌ها از طریق آن به روز شود. در این روش وزن‌ها و بایاس‌ها در جهت عکس تابع کارایی به روز می‌شوند. یک EXTENTION که برای این روش وجود دارد این است که ما بیاییم برای آن از پارامتر سرعت یادگیری استفاده کنیم، در واقع این پارامتر در شیب ضرب شده و برای به روز رسانی وزن‌ها و

بایاس‌ها استفاده می‌شود. سرعت یادگیری باعث می‌شود که اندازه هر گام اصلاحی بیشتر شود. اگر اندازه این پارامتر خیلی بزرگ باشد آموزش ثبات کافی نخواهد داشت و اگر خیلی کوچک باشد الگوریتم به زمان زیادی برای همگرا شدن نیاز خواهد داشت.

✓ آموزش دسته‌ای کاهش شیب با گشتاور : در این روش به شبکه اجازه داد می‌شود که علاوه بر تغییرات شیب به تغییرات سطح خطا نیز واکنش نشان دهد. در این روش خطاهای ناچیز نادیده گرفته می‌شوند و تغییر وزن‌ها در هر مرحله برابر خواهد بود با مجموع تغییرات اخیر ایجاد شده روی وزن‌ها و تغییرات محاسبه شده از طریق BP.

✓ آموزش با تغییر سرعت یادگیری(LR) : در روش‌های استاندارد کاهش شیب مقدار سرعت یادگیری(LR) در تمام مراحل آموزش ثابت است در حالی که همانطور که در قسمت قبل توضیح داده شد کارایی الگوریتم به شدت به مقدار LR وابسته است. برای کارایی بیشتر الگوریتم بهتر است مقدار LR در فرآیند زمان آموزش بر اساس سطح کارایی تغییر کند.

✓ پس انتشار ارتجاعی: شبکه‌های چند لایه معمولاً از توابع انتقال در لایه مخفی خود استفاده می‌کنند. این توابع اغلب کوبنده (SQUASHING) نامیده می‌شوند زیرا آن‌ها ورودی که در محدوده بینهایت می‌باشد را به محدوده کوچکی فشرده می‌کنند. عموماً در این توابع در صورتی که ورودی آن‌ها بزرگ باشد شیب تابع به سمت صفر می‌رود که این مسئله از الگوریتم SD ایحاد مشکل می‌کند، زیرا در این صورت شیب دارای مقدار کوچکی می‌شود و در نتیجه تغییر کمی در وزن‌ها و بایاس‌ها ایجاد می‌کند و در نتیجه وزن‌ها و بایاس‌ها فاصله زیادی از مقدار بهینه خود خواهند داشت. ایده‌ای که RESILIENT BP ارائه می‌دهد این است که تنها از علامت مشتق برای به روز رسانی استفاده کنیم و مقدار تغییرات وزن‌ها با استفاده از یک مقدار به روز رسانی مجزا تعیین شود. این الگوریتم کارایی بسیار بالاتری نسبت به سایر الگوریتم‌های استاندارد SD دارد و علاوه بر آن حافظه کمتری برای این الگوریتم نیاز است.

- الگوریتم‌های شیب توأم : الگوریتم پایه BP وزن‌ها را در خلاف جهت شیب اصلاح می‌کند. این همان جهتی است که تابع کارایی در آن سمت به سرعت کاهش می‌آید. این اصلاح وزن‌ها الزاماً تنها در یک جهت منتهی به همگرایی بیشتر نخواهد شد بنابر این بهتر است توأماً چند جهت را چک کنیم. در بیشتر الگوریتم‌های مورد بحث از سرعت یادگیری برای تعیین اندازه گام‌ها در به روز رسانی وزن ها استفاده می‌شود. در بیشتر الگوریتم‌های شیب توأم اندازه هر گام به ازای هر تکرار تنظیم می‌شود. برای این منظور یک عملیات جستجو بین تمامی شیب‌های توأم انجام می‌شود که مقدار تابع کارایی را در طول آن خط حداقل می‌سازد.

- الگوریتم‌های شبه نیوتن : روش‌های نیوتن معمولاً دارای همگرایی بهتر و سریعتر نسبت به الگوریتم‌های شیب توأم می‌باشند اما چون به لحاظ محاسباتی به علت محاسبه ماتریس هسیان، بسیار پر هزینه هستند در نتیجه چندان برای شبکه‌های عصبی مناسب به نظر نمی رسند. به همین دلیل از الگوریتم‌های شبه نیوتن استفاده می‌شود که در آن‌ها به تقریب ماتریس هسیان بسنده می‌کنند.

الگوریتم LEVENBERG-MARQUARDT : همانند روش‌های شبه نیوتن این روش نیز سعی در کاهش محاسبات با استفاده از عدم محاسبه ماتریس هسیان دارد. زمانی که تابع کارایی بصورت مجموع مربعات باشد ماتریس هسیان به روش $H = J^T J$ قابل تخمین است. و همچنین شیب نیز به صورت $g = J_e^T$ قابل محاسبه است. J ماتریس ژاکوبین می‌باشد که شامل مشتقات اول از خطاهای شبکه نسبت به وزن‌ها و بایاس‌ها است و e بردار خطای شبکه استو پیچیدگی محاسبات آن نسبت به محاسبه ماتریس هسان بسیار کمتر است.

مقایسه بین انواع آموزش :

با بررسی مثال‌هایی در زمینه‌های مختلف مشخصه‌های زیادی از الگوریتم‌های مورد استفاده قابل استنباط است. عموما در مسائل تخمین تابع با پارامترهای شبکه کمتر از ۱۰۰؛ الگوریتم LEVENBERG-MARQUARDT کارایی بالایی از خود نشان می‌دهد وسرعت بالایی دارد و البته

دقت بالای آن کاملا برجسته است، زیرا در بسیاری از موارد این الگوریتم به حداقل خطا رسیده است. در مقابل در مسائل تشخیص الگو LM اصلا کارایی مناسبی ندارد. فضای مورد نیاز برای این الگوریتم بالا می‌باشد که البته با استفاده از REDUC_MEM می‌توان آن را کاهش داد که البته این مسئله باعث افزایش زمان میگردد.

- LM برای مسائل حداقل سازی مربعات طراحی شده و در تشخیص الگو جواب مناسبی نمی دهد زیرا نرون‌های آن اشباع می‌شوند.
- ما در این پروژه از الگوریتم TRAINLM برای آموزش شبکه استفاده کرده ایم.که در واقع از همان الگوریتم LEVENBERG-MARQUARDT استفاده می‌کند.

الگوریتم RESILIENT BP دارای بیشترین سرعت در مسائل تشخیص الگو می‌باشد ولی این الگوریتم در مسائل تخمین تابع خوب عمل نمی‌کند،کارایی این تابع با کاهش ERROR GOAL کاهش می‌یابد اما در مقابل حافظه مورد نیاز آن در مقایسه با بقیه کمتر است.

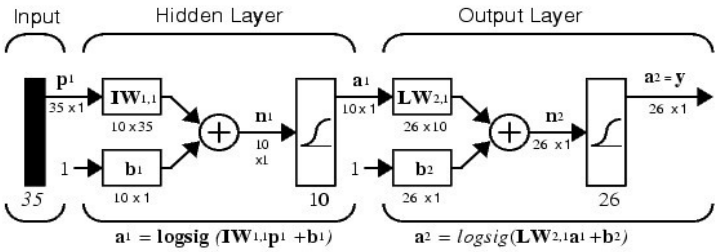
الگوریتم‌های شیب توام و به خصوص SCALED CONJUGATE GRADIENT، به نظر می‌آید که برای حل گستره وسیعی از مسائل کارایی خوبی دارند،به خصوص مسائلی با تعداد پارامتر زیاد SCG به اندازه LM سریع هستند و همچنین در مسائل تشخیص الگو به اندازه RBP سرع

ت دارند و علاوه برآن کارایی آن‌ها با کاهش ERROR GOAL کاهش پیدا نمیکند.الگوریتم‌های شیب توام معمولا فضای حافظه متوسط اشغال می‌کنند.

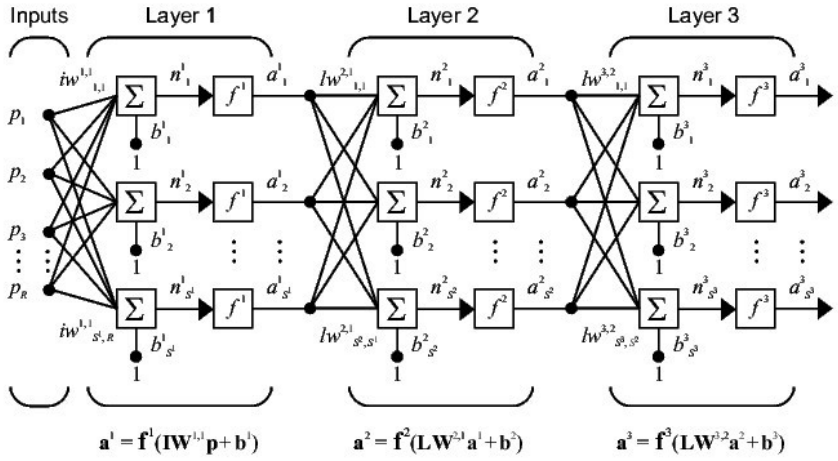
- الگوریتم کاهش گرادیان (GRADIENT DESCENT) عموما بسیار کند است زیرا برای یک یادگیری با ثبات نیاز به سرعت یادگیری کوچکی دارد. معمولا روش گشتاور(MOMENTUM) از روش کاهش گرادیان ساده سریع تر است زیرا آن ضمن یک یادگیری با ثبات میتواند سرعت یادگیری بزرگتری اختیار کند. اما این الگوریتم نیز برای بسیاری از مسایل خیلی کند است. این الگوریتم معمولا با یادگیری گام به گام(INCREMENTAL LEARNING) به شکل مطلوب عمل می‌کند.

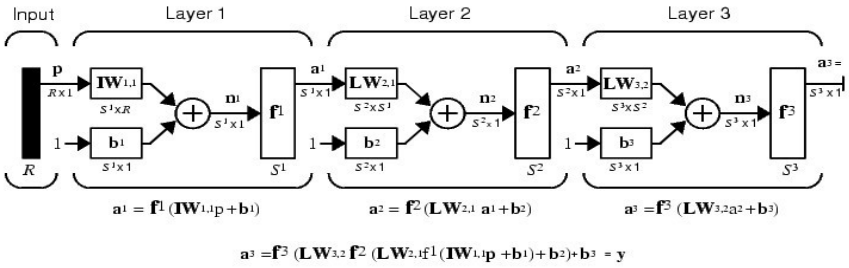
شبکه عصبی چند لایه(MLP)

شبکه‌های feedforward اغلب یک یا چند لایه مخفی از نرون‌های sigmomid می‌باشند و از یک لایه پایانی خطی استفاده می‌کنند. شبکه‌های چند لایه از نرون ها با یک تابع انتقال غیر خطی استفاده می‌کنند که به شبکه اجازه می‌دهد که توانایی یادگیری رابطه خطی و غیر خطی را بین ورودی ها و خروجی ها داشته باشد.لایه خروجی خطی به شبکه این امکان را می‌دهد که خروجی خارج از محدوده ۱ و 1- داشته باشد که البته توابع اشباع شونده‌ای وجود دارند که بتوان خروجی را در بازه ۰ و ۱ محدود کرد. مانند logsig.

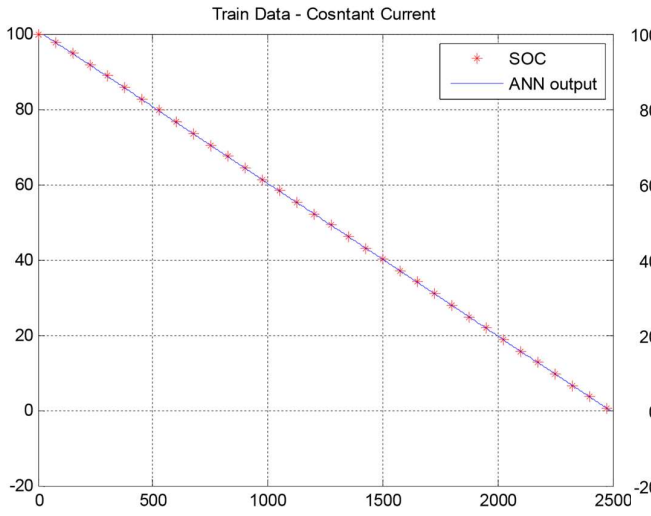


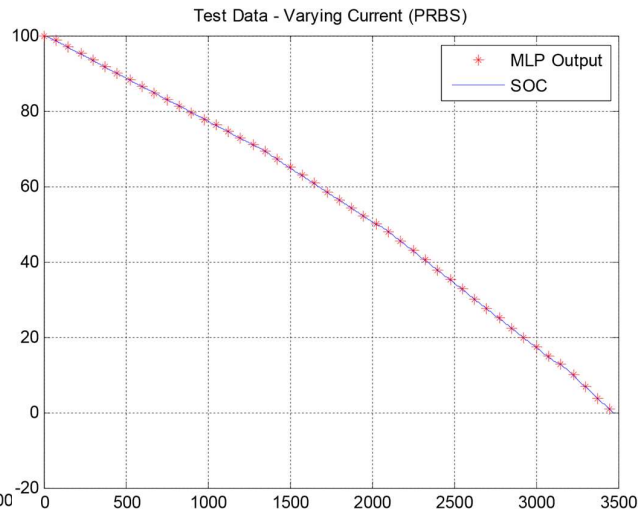
ثابت شده است که شبکه‌های عصبی چند لایه یک موتور تقریب تابعی است به شرط آنکه تعداد نرون‌های لایه میانی به تندازه کافی زیاد باشند. می توان با استفاده از الگوریتم backpropagation وزن‌ها ی شبکه عصبی را نیز تنظیم نمود. ساختار کلی شبکه عصبی را در زیر مشاهده می‌کنیم :

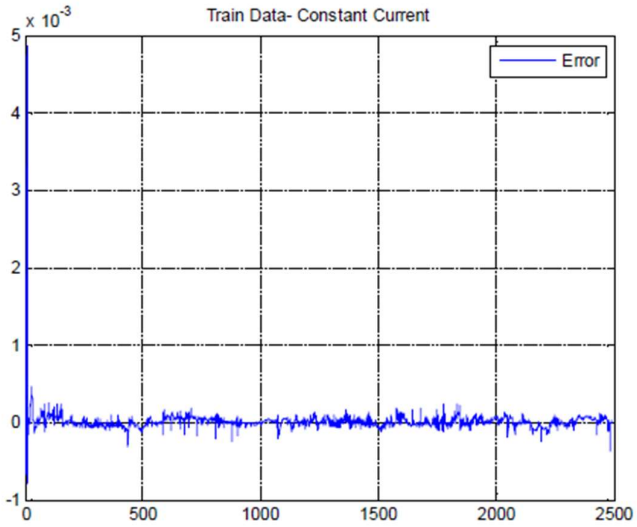




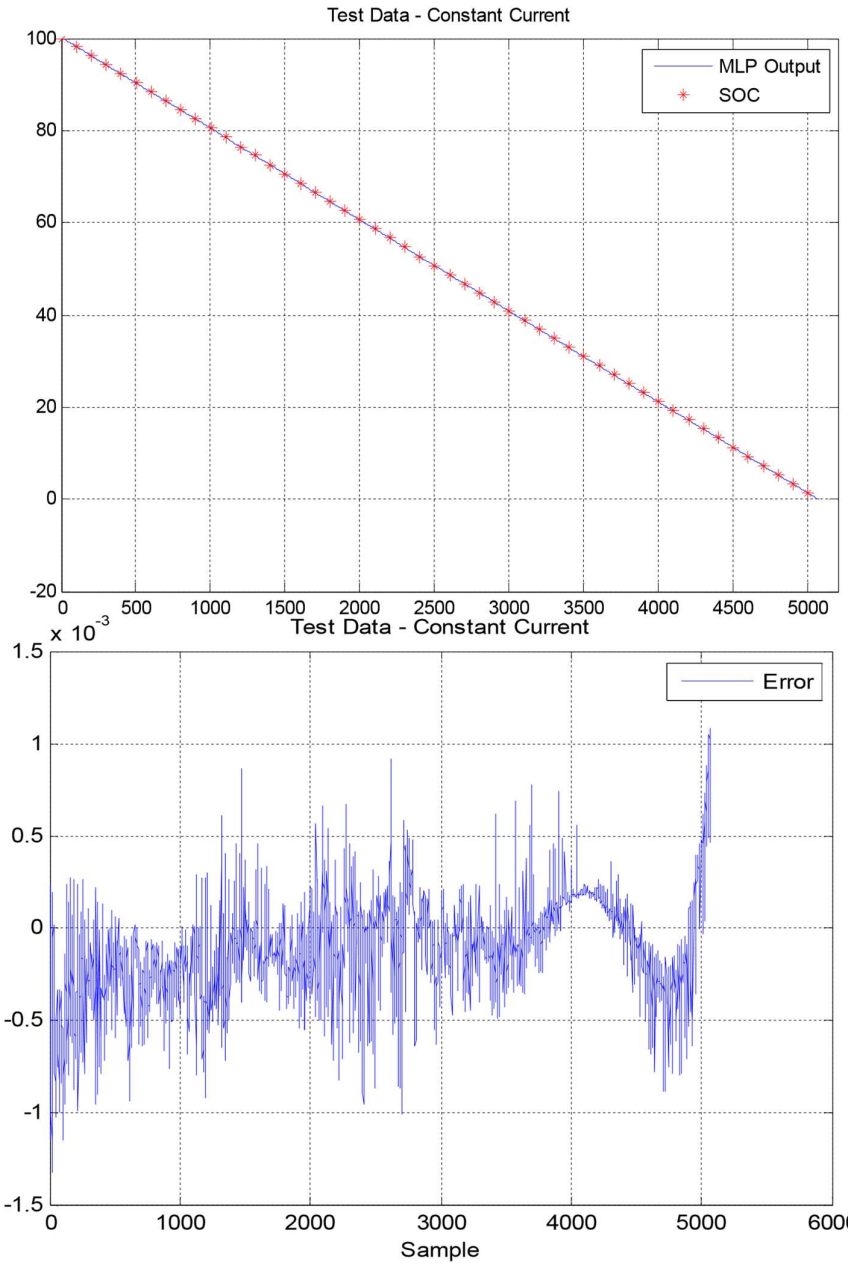
نتیجه حاصل از آموزش شبکه MLP دو لایه به صورت زیر می‌باشد. مشاهده می‌شود که با استفاده از این شبکه می‌توان سیستم را به خوبی شناسایی کرد. برای شناسایی ۳ سری داده‌های جمع آوری شده در جریان دشارژ ثابت به عنوان داده‌های آموزش در نظر گرفته شد. پس از آموزش، شبکه برای دو حالت جریان ثابت و جریان متغیر PRBS فراخوانی شده و با خروجی‌های واقعی که به عنوان داده‌های تست جمع آوری شده بود مقایسه شد. مشاهده می‌شود که تخمین زده به طور مناسبی generalization دارد و می‌تواند مقدار شارژ باتری را در وضعیت‌های مختلف تخمین بزند.



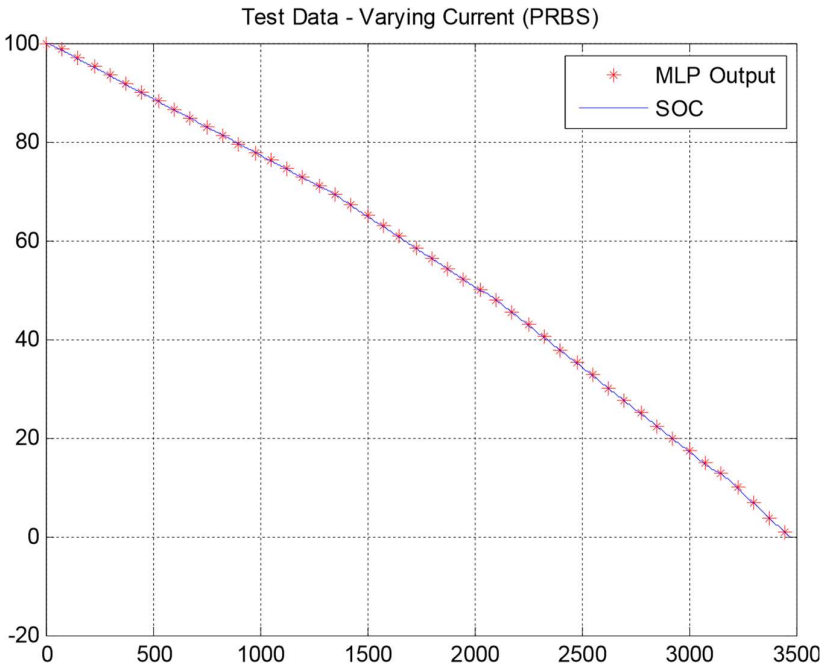


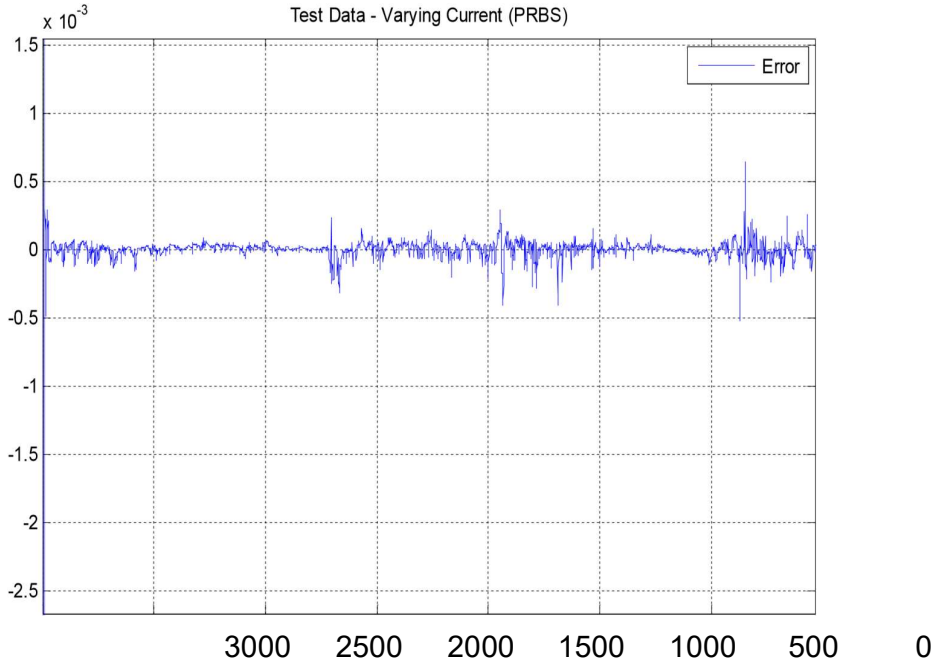


همانطور که مشاهده می‌شود خطای تخمین به خوبی به سمت صفر رفته است

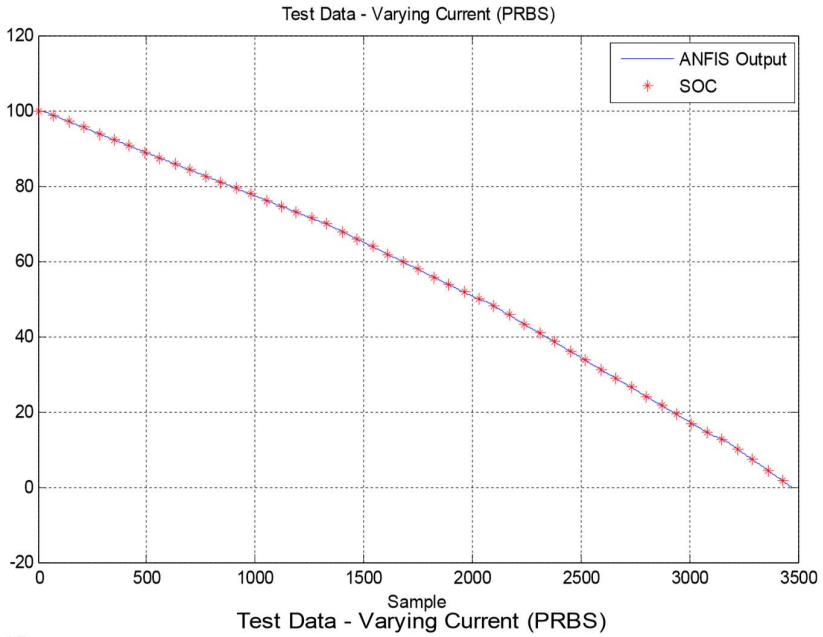


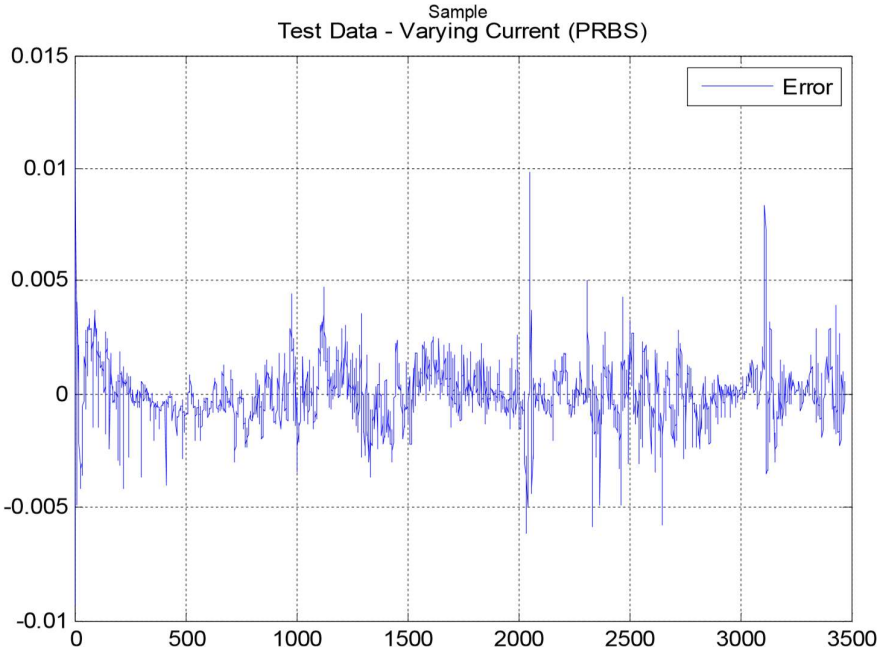
همانطور که مشاهده می‌شود خطای تخمین به خوبی به سمت صفر رفته است





همانطور که مشاهده می‌شود خطای تخمین به خوبی به سمت صفر رفته است

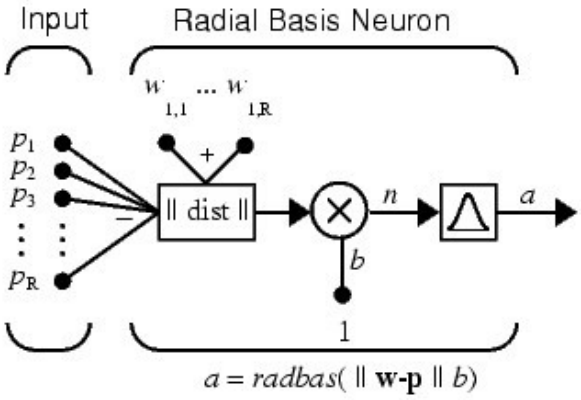




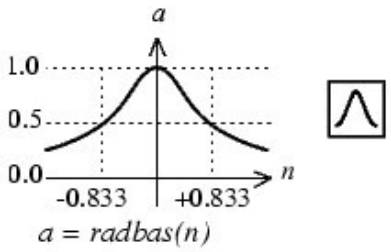
همانطور که مشاهده می‌شود خطای تخمین به خوبی به سمت صفر رفته است

شبکه‌های پایه شعاعی (RBF):

تفاوت عمده شبکه RBF و MLP در شباهت سنجی ورودی است که در MLP این شباهت سنجی بصورت ضرب داخلی بوده و در شبکه RBF این شباهت سنجی بصورت فاصله اقلیدسی است به این معنی که فاصله ورودی با مرکز تمامی گوسی ها سنجیده می‌شود، که ساختار کلی آن بصورت زیر آورده شده است.



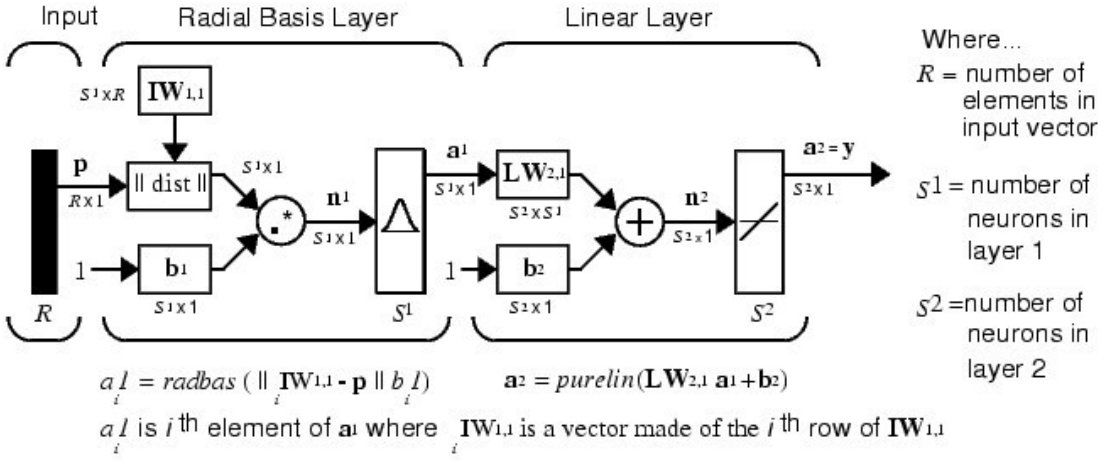
و تابع نرون radial basis بصورت زیر است :



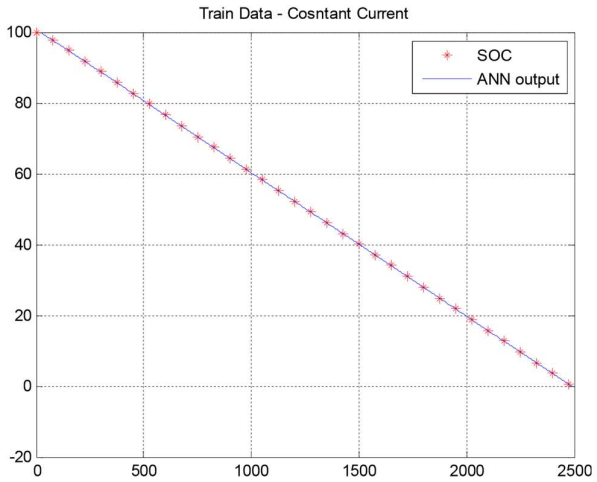
Radial Basis Function

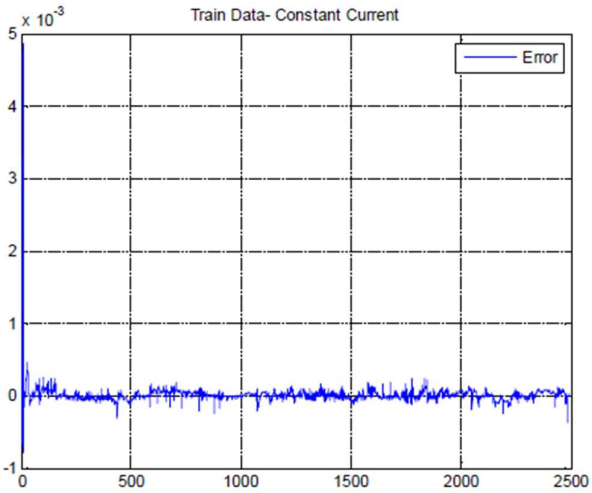
این تابع با ورودی صفر دارای حداکثر خروجی خود یعنی ۱ می‌باشد. (به نوعی Fuzzy Partition مشاهده می‌شود)

معماری شبکه radial basis از دو لایه تشکیل شده است که معماری آن در شکل زیر نشان داده شده است



لایه مخفی دارای S1 نرون و لایه خروجی دارای S2 نرون می‌باشد. تابع ||distt|| بردار ورودی P و بردار وزن‌ها ی ورودی W را به عنوان ورودی دریافت کرده و یک بردار با S1 عنصر تولید می‌کند.





همانطور که مشاهده می‌شود خطای تخمین به خوبی به سمت صفر رفته است

با توجه به پاسخ مطلوب تر RBF نسبت به MLP سوال این است که چرا همیشه از شبکه‌های radial basis به جای شبکه‌های feedforward استفاده نمی کنیم؟ زیرا در تعداد نرون‌های شبکه‌های radial basis است. تعداد نرونها در این نوع شبکه ها بسیار بیشتر از شبکه‌های feedforward است زیرا شبکه‌های feedforward از توابع sigmoid به عنوان تابع انتقال استفاده می‌نمایند که این توابع می‌توانند بر روی محدوده گسترده‌ای از فضای ورودی عمل نمایند اما نرون‌های radbas تنها می‌توانند روی فضای محدودی عکس العمل نشان دهند و در نتیجه برای گسترش این فضا نیاز به نرون‌های بیشتری وجود خواهد داشت. اما از سوی دیگر طراحی این شبکه ها نسبت به شبکه‌های feedforward به زمان کمتر و حتی گاهی اوقات به تعداد نرون‌های کمتری نیاز دارد.

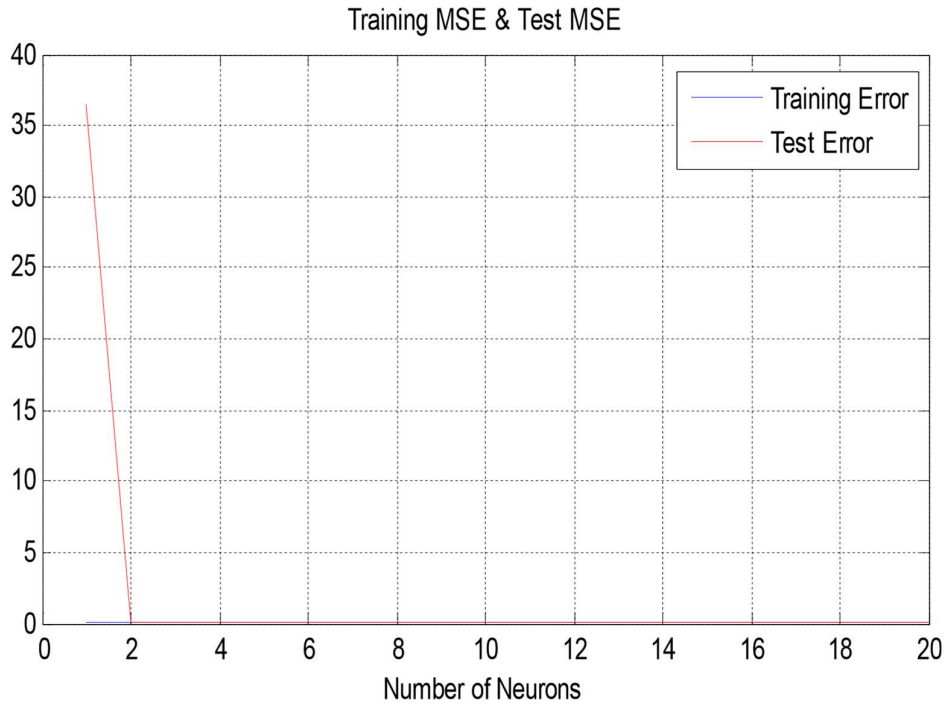
شناسایی توسط مدل‌های عصبی- فازی

LOLIMOT: (localy linear model tree)

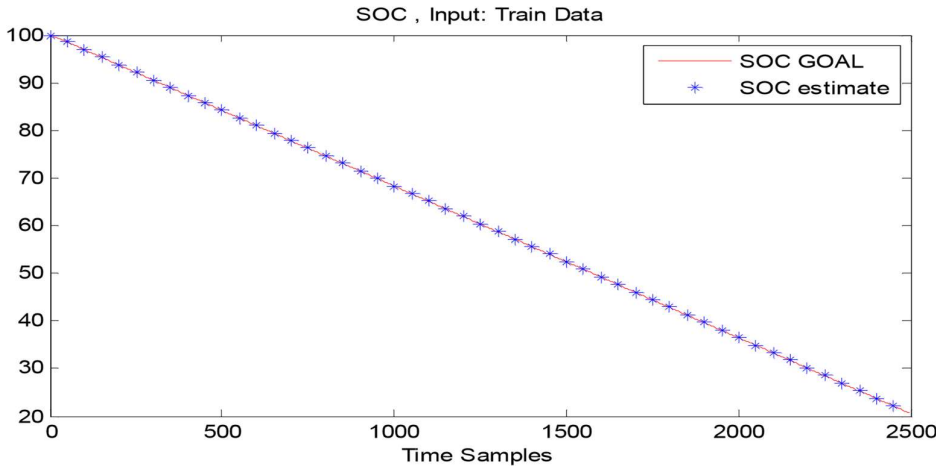
در این قسمت از پروژه از مدل نرو فازی خطی محلی با الگوریتم LoLiMoT(برای یادگیری) استفاده می‌کنیم.

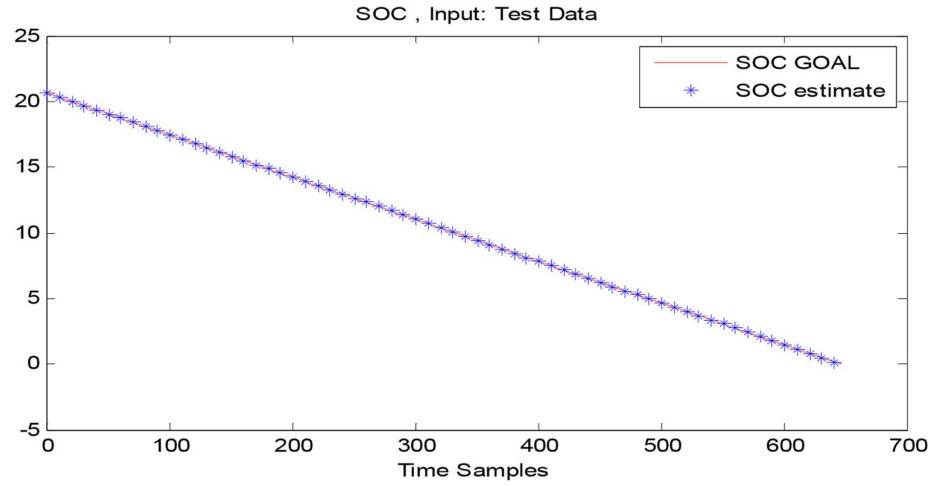
در این روش یک مدل پیچیده به چند مدل ساده تر و کوچک تر شکسته شده و مدل‌های کوچک تر به صورت خطی و تقریباً مستقل شناسایی می‌شوند. در این مدل هر نرون از یک مدل خطی محلی (LLM) و یک تابع validity که محدوده اعتبار این مدل خطی محلی را نشان می‌دهد، تشکیل می‌شود. مجموع مقدار توابع validity، که معمولاً گوسی انتخاب می‌شوند، به ازای هر ورودی برابر ۱ است، یعنی این توابع نرمالیزه هستند. این الگوریتم فضای ورودی را به موازات محورهای عمودی به دو قسمت مساوی تقسیم می‌کند، به این ترتیب فضای ورودی به مستطیل هایی تقسیم می‌شود. در مرکز هر یک از این مستطیل ها یک تابع گوسی با انحراف معیاری برابر یک سوم اندازه هر بعد آن قرار داده می‌شود. در نتیجه پارامترها غیرخطی مدل بدون نیاز به روش‌های بهینه سازی غیرخطی به دست می‌آیند. در الگوریتم LOLIMOT تقسیم بندی فضای ورودی به این ترتیب انجام می‌پذیرد که ابتدا بدترین LLM (در اولین مرحله کل فضای ورودی که دارای یک نرون است) که بیشترین مربعات خطای وزن دار خروجی را ایجاد کند، انتخاب میشود. این LLM به صورت axis-orthogonal به دو قسمت مساوی تقسیم و در مرکز هر قسمت یک نرون قرار داده شده و خطای مدل سازی در هر قسمت محاسبه می‌شود. برای تمام جهت‌های ممکن که برای شکستن فضا وجود دارد، این کار انجام شده و خطای مدل سازی محاسبه می‌شود. جهتی که گذاشتن نرون در هر یک از دو قسمت آن خطای مدل سازی کمتری ایجاد کند ،انتخاب می‌گردد. بار دیگر بدترین LLM انتخاب شده و به صورت axis-orthogonal شکسته می‌شود. این کار تا برقراری یک شرط توقف که می‌تواند تعداد نرون ها یا رسیدن به حد قابل قبولی از خطای مدل سازی باشد ،ادامه می‌یابد.
شکل زیرنتایج حاصل از شناسایی تابع اول با استفاده از این روش را نشان می‌دهد. علت کاهش تعداد نرون ها نسبت به RBF دراین است که در این الگوریتم، تقسیم بندی فضا با درنظرگرفتن میزان خطا درهرناحیه و به صورت هوشمندانه انجام می‌شود و بدیهی است که درمقایسه با RBF تعدادنرون ها بایدکاهش یابد.

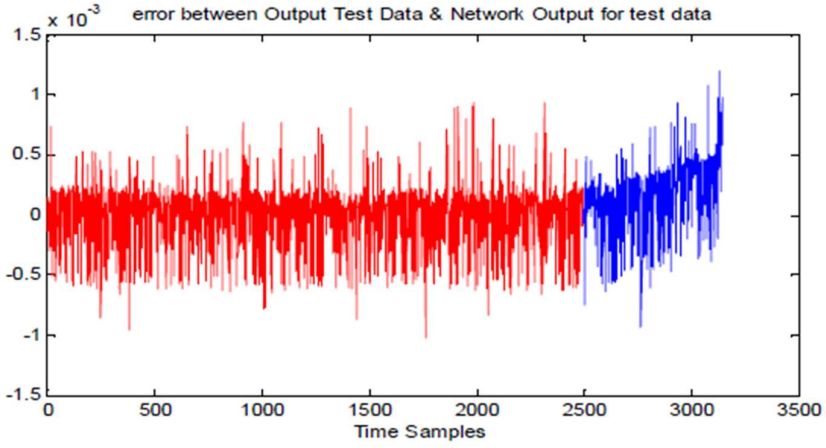
لازم به ذکر است در اینجا اجمالا اشاره می‌شود در این قسمت ابتدا داده ها را نرمالیزه می‌کنیم ،که مرکز توابع گوسی و واریانس آن‌ها به صورت خودکار شناسائی می‌شوند و تعداد نرونها را از مقدار کم تا زیاد تغییر می‌دهیم و با مشاهده همزمان میانگین مربعات خطای آموزش و ارزیابی، تعداد بهینه نرون ها را بدست می‌آوریم.



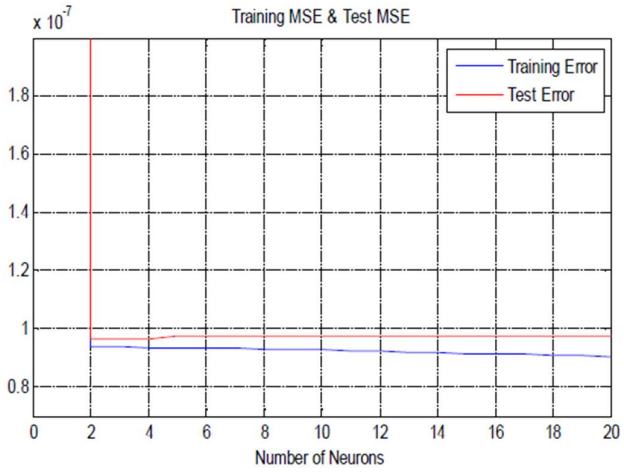
که با بزرگنمایی داریم:

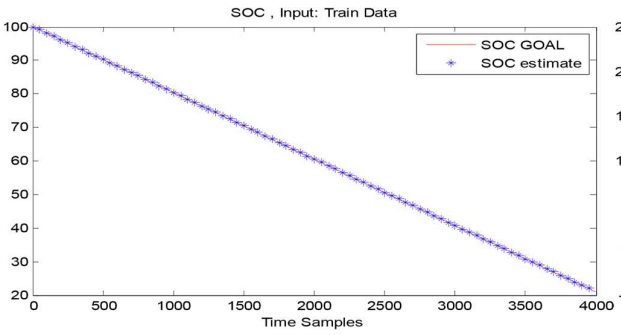

SOC , Input: Train Data
SOC GOAL
SOC estimate
100
90
80
70
60
50
40
30
20
0
500
1000
1500
2000
2500
Time Samples

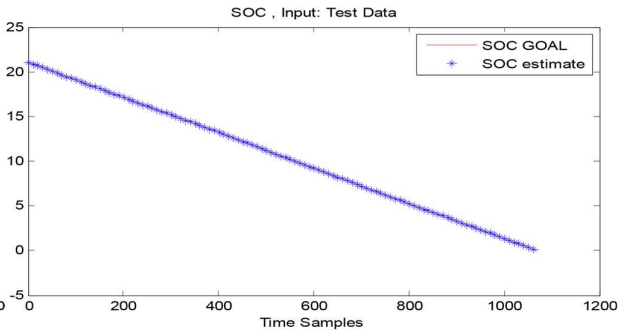

SOC , Input: Test Data
SOC GOAL
SOC estimate
25
20
15
10
5
0
-5
0
100
200
300
400
500
600
700
Time Samples

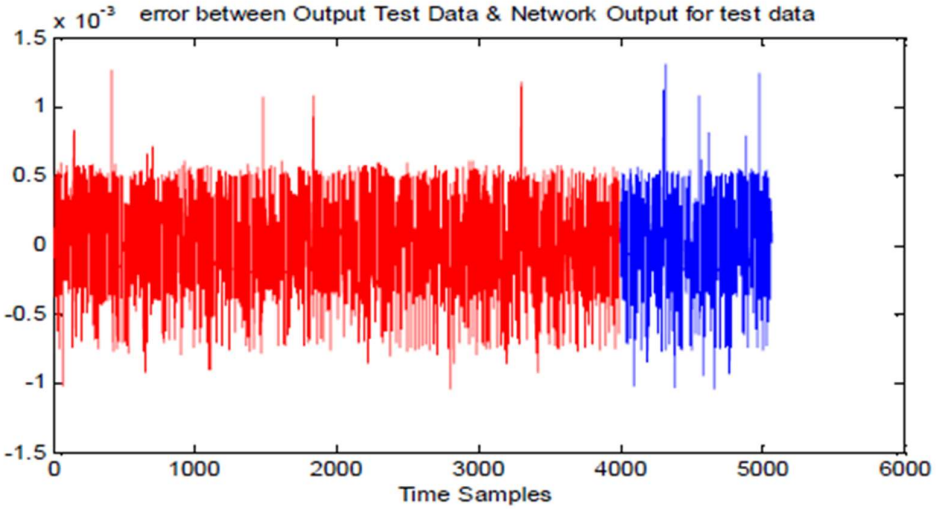

x 10^-3
error between Output Test Data & Network Output for test data
1.5
1
0.5
0
-0.5
-1
-1.5
0
500
1000
1500
2000
2500
3000
3500
Time Samples

برای داده‌ی دوم:

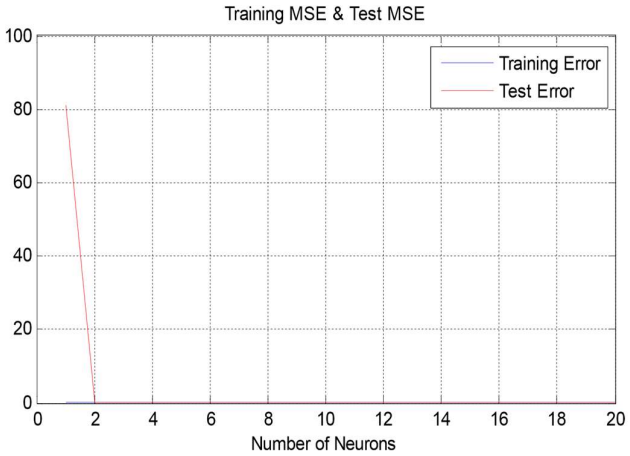


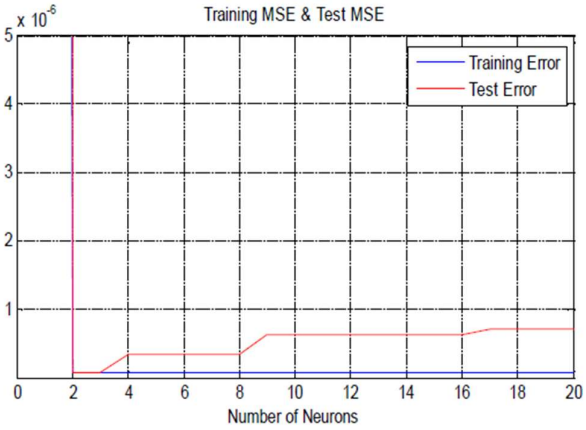


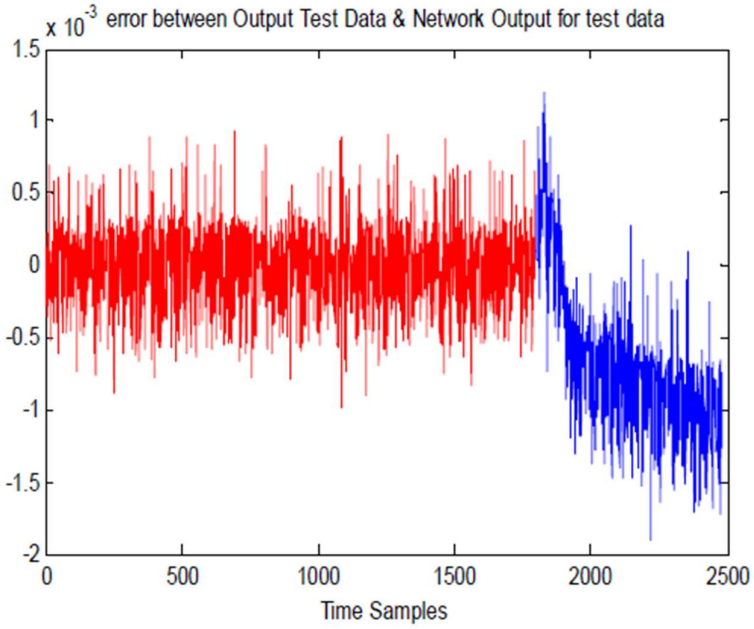




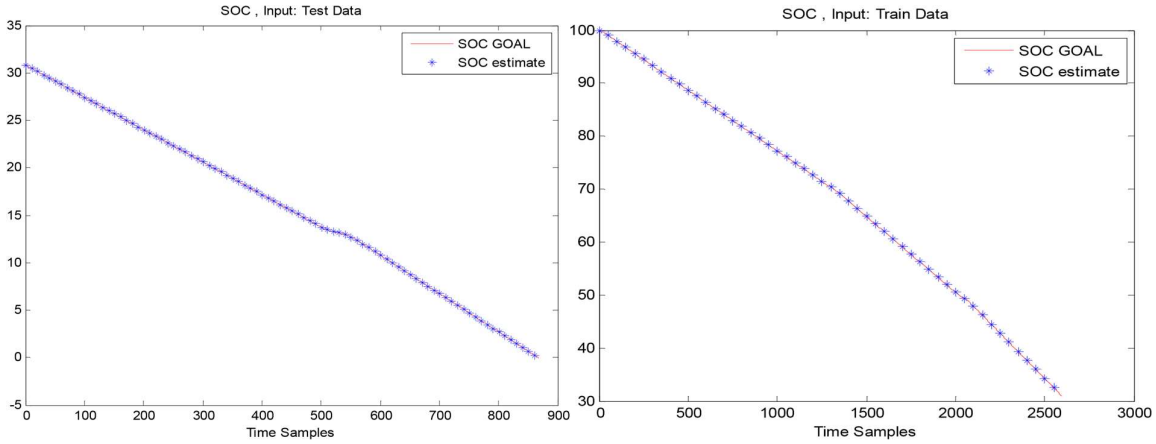
داده‌ی سوم:

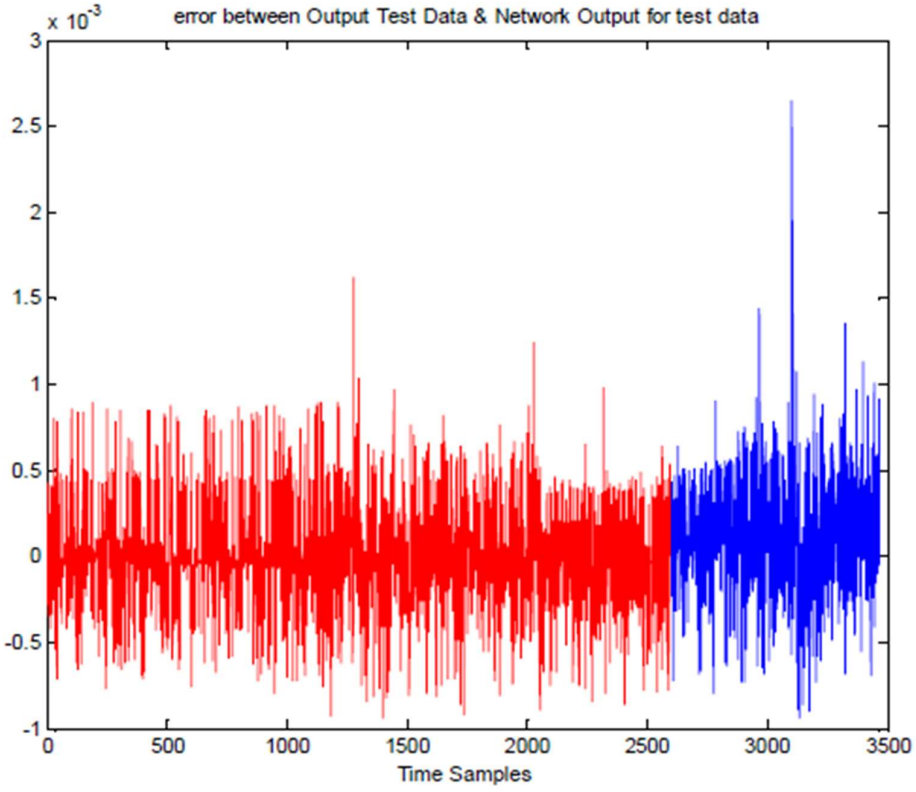

Training MSE & Test MSE
Training Error
Test Error
Number of Neurons


x 10^-6
Training MSE & Test MSE
Training Error
Test Error
Number of Neurons


x 10^-3 error between Output Test Data & Network Output for test data
Time Samples

داده‌ی چهارم:


SOC , Input: Test Data
SOC GOAL
SOC estimate
Time Samples
SOC , Input: Train Data
SOC GOAL
SOC estimate
Time Samples


x 10^-3
error between Output Test Data & Network Output for test data
Time Samples

شناسایی توسط مدل ANFIS

(Adaptive Network Fuzzy Inferense System)

همانطور که در [۱۰] بیان شده است ،ANN در عین یادگیری خوبی که دارد شفافیتی برای حل مسله ندارد ودر مقابل آن منطق فازی دارای شفافیت لازم بوده اما ارائه راه حل دقیق برای آن دشوار است پس باادغام ANN و منطق فازی ،ANFIS ایجاد می‌شود که ویژگی‌های مثبت هر دو را دارد و ما نیز از ANFIS برای تخمین SOC ایجاد می‌کنیم.

در ادامه، مدلANFIS با ۵ ورودی(ـــ , V, i و SOC) و یک خروجی (SOC) را پیاده سازی می‌کنیم. برای این منظور از قواعد فازی اگر–پس Takagi_Sugeno به همرا یادگیری هایبرید(روش گرادیان مبنا وحداقل مربعات تخمین (LSE) برای آموزش استفاده می‌شود که روش propagation برای یادگیری firing و روش حداقل مربعات تخمین(LSE) برای یادگیری پارامترهای خطی مربوط به قسمت فازی استفاده می‌کنیم).

در تخمین SOC باتری ها معمولا با دو مشکلات اصلی روبرو هستیم :

۱. ممکن است ۱۰۰٪ نباشد.

در بسیاری از تحقیقات موجود در تخمین SOC مانند[۱۲،۱۱و۱۳] از SOC ۱۰۰ ٪ استفاده شده است که نشان می‌دهد، برای تخمین SOC از باتری که به طور کامل شارژ شده، استفاده می‌شود. با این حال، در بسیاری از کارها، ممکن است باتری از۱۰۰ ٪SOC شروع به دی شارژ شدن نکند. سه دلیل برای این مورد وجود دارد: (الف) خود تخلیه شدن در اثر گذشت مدت زمان طولانی (ب)تخلیه متناوب باتری(پ) در اصل به طور کامل شارژ نشده باشد. باتری با شارژ کامل ویژگی‌های آشکاری دارد که یکی از این ویژگی ها ولتاژ ترمینال اولیه نسبتا بالا می‌باشد. در این پروژه کار ما بر این اساس است که مقدار را نمی دانیم.

۲. انتخاب و اتخاذ داده‌های آموزش

می بایست به تعداد کافی داده‌ی آموزش برای black_box مهیا شود. پیش بینی SOC برای شرایط خاص، نیاز به داده ها آموزش تحت همان شرایط دارد به عنوان مثال، به منظورتخمین SOC تحت تخلیه‌ی بارثابت (CLD) به داده های آموزش تحت شرایطCLD نیاز داریم.

انتخاب ورودی:

انتخاب ورودی یکی از مسایل مهم در شناسایی می‌باشد. که روش‌های مختلفی برای این منظور وجود دارد. یکی از این روش‌های جدید استفاده از آنالیز حساسیت است و ژنتیک الگوریتم [۱۴،۱۳] و روش‌های آنالیز کورلیشن [۱۵] که به کرات استفاده می‌شود. آنالیز کورلیشن یک متد مستقیم و قابل فهم است ما نیز از آنالیز کورلیشن برای انتخاب ورودی ها استفاده می‌کنیم و ورودی‌های مهم و عمق هر کدام را به دست می‌آوریم. با توجه به رفتار غیرخطی و پیچیده‌ای که از باتری باطری سراغ داریم و با توجه به اینکه SOC به بسیاری از عوامل بستگی دارد، سه متغیرهای اساسی جریان تخلیه i، SOC و ولتاژ ترمینال باتری v به تنهایی قادر به شناسایی سیستم نخواهد بود و اگر تنها از سه متغیر به عنوان ورودی استفاده کنیم مدل ANFIS نمی تواند SOC باتری را به دقت پیش بینی کند به همین منظور برخی از متغیرهای ورودی جدید ،که باید

به مدل اضافه شده تا موجب بهبود دقت پیش بینی شوند را تعریف می‌کنیم. معمولا این متغییرهای جدید را از روی متغیرهای اساسی و به صورت متوسط گیری، انتگرال گیری و تفاضل به دست می‌آوریم. متغیرهای در نظر گرفته شده مطابق جدول زیر می‌باشند:

ولتاژ	ولتاژ ترمینال باتری
جریان	جریان دشارژ
مشتق ولتاژ	$dV = \frac{V(t)-V(t-1)}{T}$
آمپر ساعت	$As(t) = \int_0^t i(t)dt$

Soc خروجی ANFIS می‌باشد و با جریان ثابت‌های مختلف (که در این جا ۰.۷ و ۱.۲ و ۱.۵ آمپر) دیتا ها را جمع آوری کرده. با توجه به وجود ۴ تا ورودی $p_j , q_j , r_j , s_j , t_j$ به صورت از ۱۰ تا mf استفاده می‌کنیم (لایه‌ی اول) و هر کدام از MF ها نیز با در نظر گرفتن حالت مثلثی سه تا پارامتر خواهند داشت. برای لایه‌ی دوم نیز ۱۶ تا fixed node خواهیم داشت (2) (در حقیقت همان وزن هر رول ها می‌باشند).در لایه‌ی سوم ۱۶ تا fixed node برای عمل normalization مقدار اثر گذاری هر رول استفاده می‌کنیم. در لایه‌ی چهارم نیز ۱۶ تا if-then rule خواهیم داشت هر کدام از این رول ها نیز خودشان ۵ تا پارامتر دارند: $p_j , q_j , r_j , s_j , t_j$

$$f_j = \mathrm{SOC} * p_j + \mathrm{V} * q_j + \mathrm{i} * r_j + \frac{\mathrm{dv}}{\mathrm{dx}} * s_j + t_j$$

در نهایت هم با استفاده‌ی ترکیبی از LS و SD پارامترهای ANFIS به دست می‌آیند.

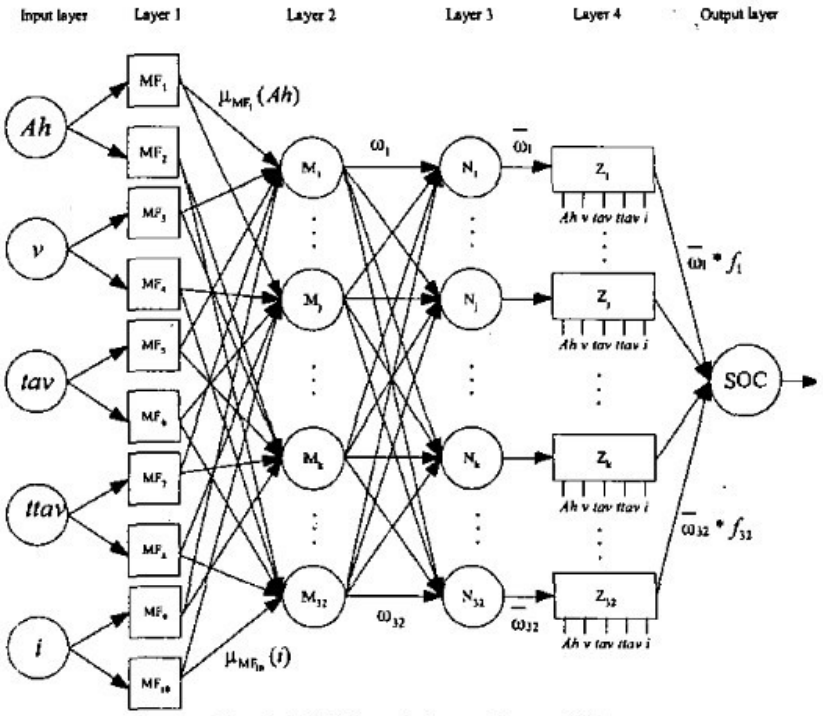
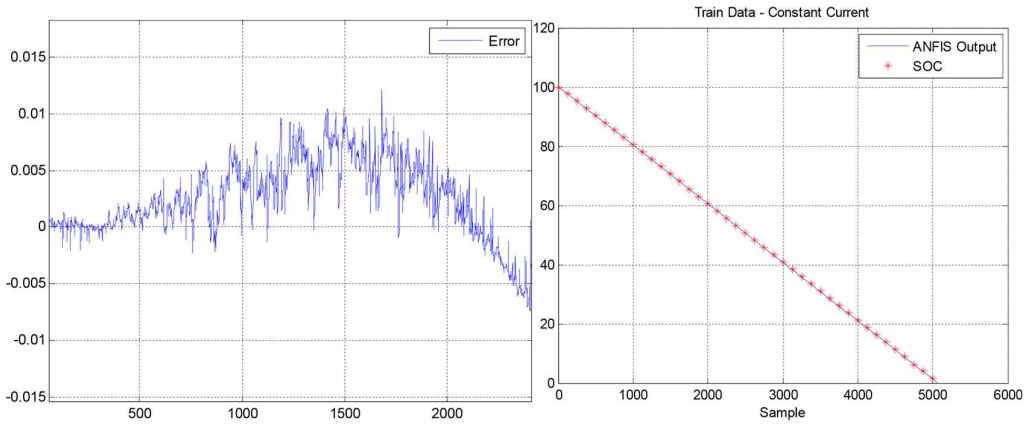


Fig. 4 ANFIS model to estimate SOC

زیاد تعداد بودن داده ها آموزش ممکن است عملکردکلیت ANFIS را تضمین کند اما نیاز به مقدار محاسبات. زیاد خواهد داشت وتعداد بسیار کم از داده‌ی آموزش عملکردکلیت ANFIS بعید است در کل۱۹۲= ۳۲*۶ تا پارامتر خطی داریم که از LS به دست می‌آید. برای ۳*۱۰ تا المان که المآن‌ها ی mf ها می‌باشند از بهینه سازی غیر خطی استفاده می‌کنیم. در نهایت انتظار می‌رود SOC با گذشت زمان و با تخلیه شدن به صورت زیر در آید: در نهایت MSEهای به دست آمده نشان دهنده‌ی کارکرد بهتر ANFIS نسبت به ANN می‌باشد. این مقاله حالت بار ثابت را نیز بررسی کرده است. در این حالت دیگر SOC بر حسب زمان خطی نخواهد بود. نمودار حاصله از شناسایی از روش ANFIS را مشاهده می‌کنیم:



همانطور که مشاهده می‌شود خطای تخمین به خوبی به سمت صفر رفته است. همانطور که انتظار می‌رفت مدل ANFIS به پاسخ مطلوب تری منتج می‌شود.

نتیجه گیری :

همان‌طور که انتظار می‌رفت در مدل سازی باتری‌های قابل شارژ مجدد بهترین نتیجه توسط مدل ANFIS بدست آمد. البته در این مقایسه روش‌های خطی وجو نداشته اند ولی در مدل‌های عصبی ما با وارد کردن ورودی‌ها بطور مستقیم در خروجی یک مدل خطی بصورت موازی با یک مدل غیر خطی درایو کرده ایم، از این منظر می‌توان مدل خطی را نیز لحاظ نمود.

مراجع

[1] D.U. Sauer, ***G.*** Bopp, A. Jossen, et al., State-of-charge-what do we really speak about? INTELEC, 1999

[2] S.J. Lee, J.H., Kim, J.M. Lee, and B.H. Cho, "The State and Parameter Estimation of an Li-Ion Battery Using a New OCVSOC Concept," *Power Electronics Specialists Conference*, 2007. IEEE 17-21 June 2007 Page(s):2799 - 2803

[3] S. Abu-Sharkh, D. Doerffel, "Rapid test and non-linear model characterization of solid-state lithium-ion batteries," *Journal of Power Sources* 130 (2004) 266–274

[4] S. Buller, M. Thele, E. Karden, and R.W. De Doncker, "Impedance-based non-linear dynamic battery modeling for automotive applications," *Journal of Power Sources*)3002(311

[5] G Bahhah, A.A. Girgis, Input feature selection for real-time transient stability assessment for artificial neural network (ANN) using ANN sensitivity analysis, IEEE International Conference on Power Industry Computer Applications, pp. 295-300, 1999.

[6] C.C. Peck, A.P. Dhawan, C.M. Meyer, Genetic algorithm based input selection for a neural network functionapproximator with applications to SSME health monitoring, IEEE International Conference on Neural Networks, v01.2, pp.1115 -1122, .3991

پیوست:

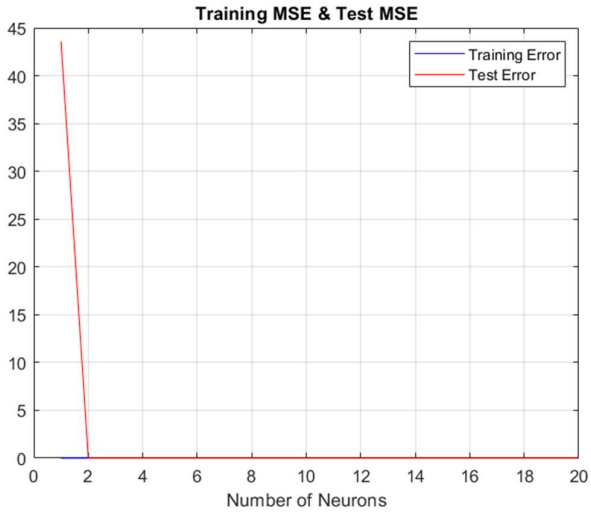
نمونه کد:

```
clc;
clear;
close all;
load input2;
load output2;
```

Normalizing output data T1 = gen_output1; T2 = gen_output2; T3 = gen_output3; T4 = gen_output4;

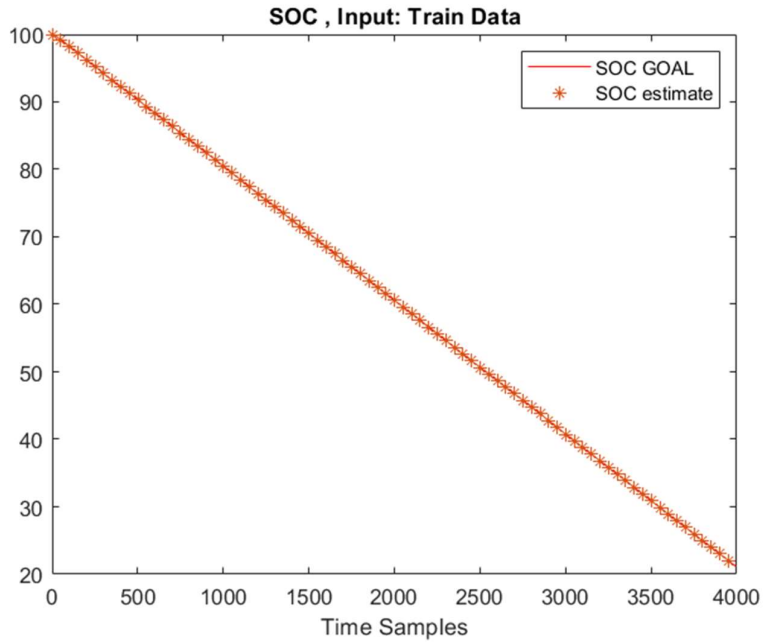
```
T =output2;

% [gen_output1,min1,max1] = premnmx(gen_output1);
% [gen_output2,min2,max2] = premnmx(gen_output2);
% [gen_output3,min3,max3] = premnmx(gen_output3);
% [gen_output4,min4,max4] = premnmx(gen_output4);
%
 cut_indx=4000;
%
% input1_tr = gen_input1(1:cut_indx)';
% output1_tr = gen_output1(1:cut_indx)';
% input1_te = gen_input1(cut_indx+1:end)';
% output1_te = gen_output1(cut_indx+1:end)';
%
% input2_tr = gen_input2(1:cut_indx)';
% output2_tr = gen_output2(1:cut_indx)';
% input2_te = gen_input2(cut_indx+1:end)';
% output2_te = gen_output2(cut_indx+1:end)';
%
% input3_tr = gen_input3(1:cut_indx)';
% output3_tr = gen_output3(1:cut_indx)';
% input3_te = gen_input3(cut_indx+1:end)';
% output3_te = gen_output3(cut_indx+1:end)';
%
% input4_tr = gen_input4(1:cut_indx)';
% output4_tr = gen_output4(1:cut_indx)';
% input4_te = gen_input4(cut_indx+1:end)';
% output4_te = gen_output4(cut_indx+1:end)';

input_tr = input2(1:4000,:)';
output_tr = output2(1:4000);

input_te =input2(4001:5073,:)';
output_te =output2(4001:5073);
in_reg = 3;   % Number of input regressors
out_reg = 1;  % Number of output regressors
lag = 1;      % System delay

P_tr = [];
P_te = [];

t = lag + in_reg +out_reg;
```

```
for k = lag:in_reg+lag-1
    P_tr = [P_tr ; input_tr(:,t-k:end-k)];
end
for k = 1:out_reg
    P_tr=[P_tr ; output_tr(:,t-k:end-k)];
end

for k = lag:in_reg+lag-1
    P_te = [P_te ; input_te(:,t-k:end-k)];
end

for k = 1:out_reg
    P_te = [P_te ; output_te(:,t-k:end-k)];
end
T_tr = output_tr(:,t:end);
T_te = output_te(:,t:end);
T_train = T(:,t:cut_indx)';
T_test = T(:,cut_indx+1:end)';


number_of_neurons = 20;
input_range = minmax(P_tr);
number_of_outputs = 1;

[Model_data,Model_out_tr,W] =
train_lolimot(P_tr',T_tr(1,:)',P_te',T_te(1,:)',input_range,number_of_outputs,number_of_neuron
s,0);
```

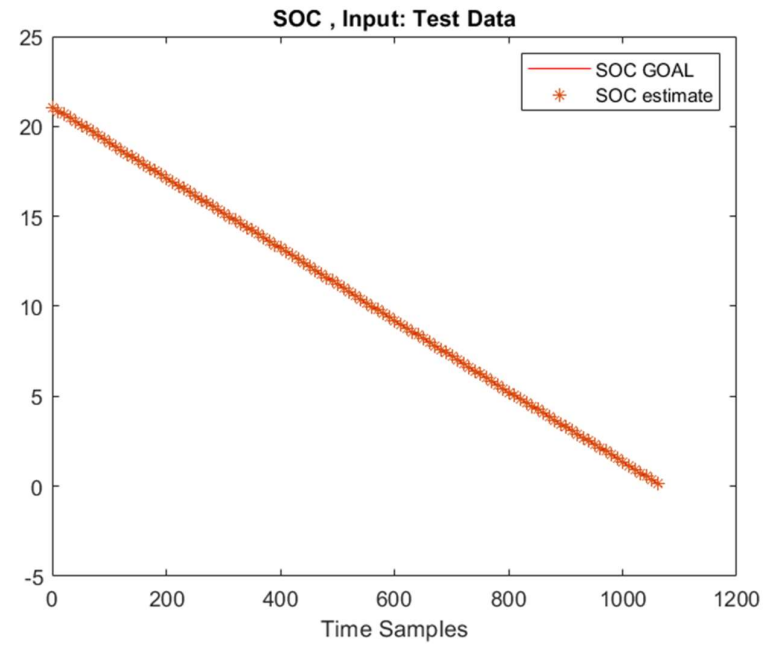



```
[Error,Model_out_te] = sim_lolimot(Model_data,W,P_te',T_te(1,:)');


MSE_tr = mse(Model_out_tr-T_tr(1,:)');
MSE_te = mse(Model_out_te-T_te(1,:)');

figure
plot(1:length(T_tr(1,:)),T_tr(1,:),'r');
hold on;
plot(1:50:length(T_tr(1,:)),Model_out_tr(1:50:end),'*')
legend('SOC GOAL','SOC estimate')
```

```
title('SOC , Input: Train Data')
xlabel('Time Samples')
```




```
figure
plot(1:length(T_te(1,:)),T_te(1,:),'r');
hold on;
plot(1:10:length(T_te(1,:)),Model_out_te(1:10:end),'*')
legend('SOC GOAL','SOC estimate')
title('SOC , Input: Test Data')
xlabel('Time Samples')
```



```
figure(4)
plot(1:length(T_tr),T_tr(1,:)-Model_out_tr','r')
title('error between Output train Data & Network Output for train data')
%legend('Real Output')
xlabel('Time Samples')

hold on


plot(length(T_tr)+1:length(T_tr)+length(T_te),T_te-Model_out_te','b')
title('error between Output Test Data & Network Output for test data')
%legend('Real Output')
xlabel('Time Samples')
```