به نام خداوند جان و خرد

درس شبکه‌های عصبی

پروژه کامپیوتری (۱): مدل سازی سیستم به کمک پرسپترون چند لایه

استاد درس:

دکتر محمد فرخی

دانشجو:

عرفان ریاضتی

بهار ۱۴۰۳

دانشگاه علم و صنعت ایران – دانشکده مهندسی برق

فهرست مطالب

چکیده

همانطور که می‌دانیم شبکه‌های عصبی در مسائل مختلفی مانند تخمین تابع، کلاس بندی، و یا کنترل مورد استفاده قرار گیرند. در درسماشین‌های یادگیری و شبکه‌های عصبی با تعدادی از این ساختارهای یادگیری آشنا شدیم. به طور مثال پرسپترون روزن‌بلات که یک الگوریتم کلاس بندی باینری و خطی می‌باشد، می‌تواند برای جداسازی الگوهای خطی جداپذیر به کار گرفته شوند. همچنین، الگوریتم حداقل مربعات از دیگر الگوریتم‌هایی است که بر اساس آن می‌توان وزن‌های یک نورون خطی را محاسبه کرد. اما هر یک از این الگوریتم‌ها دارای معایبی، از جمله وابستگی به الگوهای خطی‌جداپذیر و یا پیچیدگی محاسباتی، هستند. در این مرحله، برای رفع ایرادات الگوریتم‌های گفته شده، پرسپترون چند لایه یکی از شبکه‌های یادگیری قدرتمند و کاربردی خواهد بود. این شبکه عصبی با به کارگیری الگوریتم پس‌انتشار خطا، قابلیت جداسازی الگوهای غیر خطی را دارد. همچنین، به دلیل استفاده از این الگوریتم محاسبات آن به نسبت سریع و آسان است. در ادامه، تلاش می‌کنیم تا با استفاده از این الگوریتم خروجی یک سیستم غیر خطی را با استفاده از نمونه‌های ورودی و پارامترهای شبکه یاد گرفته و تخمین بزنیم.

کلید واژه‌ها: پرسپترون چند لایه، الگوریتم پس انتشارخطا، یادگیری با نظارت،لایه پنهان، فضای ویژگی، اعتبارسنجی.

مقدمه

در این گزارش تلاش می‌کنیم تا یک سیستم غیر خطی را به کمک پرسپترون چند لایه تخمین بزنیم. هدف از این پروژه، به دست آوردن معماری شبکه شامل تعداد لایه‌های پنهان و تعداد نورون‌ها، پیدا کردن وزن‌های مناسب، و در نهایت پارامترهای شبکه مانند ضریب آموزش می‌باشد. در ابتدا به شرح مسئله و ساخت مجموعه‌داده‌ها می‌پردازیم. سپس، آن را به صورت مرحله به مرحله در نرم‌افزار متلب پیاده‌سازی می‌کنیم. در این گزارش تلاش شده است تا هر بخش از پیاده‌سازی و ارتباط آن با تئوری درس توضیح داده شود.

شرح مسئله

سیستمی غیرخطی به معادله زیر داده شده است. همانطور که مشاهده می‌شود این یک سیستم دیجیتال می‌باشد که حالت آن در هر لحظه براساس یک سیگنال کنترلی و مقدارسیستم در دوره‌های قبل تعیین می‌شود.

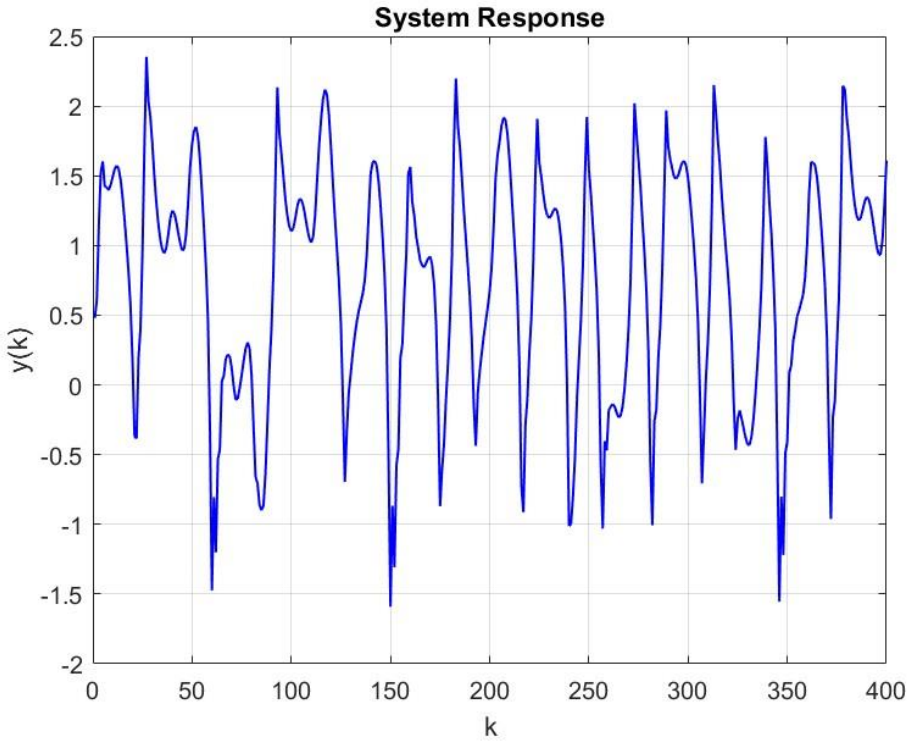
$$y(k) = \alpha \left[\frac{y(k-1)y(k-2)(y(k-2)+\beta)}{1+y^2(k-1)y^2(k-2)} + u(k-1) \right] \qquad (1)$$

که در این معادله $\alpha = 1.2$ و $\beta = [1.1,1.5]$ می‌باشند. همچنین، معادله سیگنال کنترلی $u(k)$ به شرح زیر است.

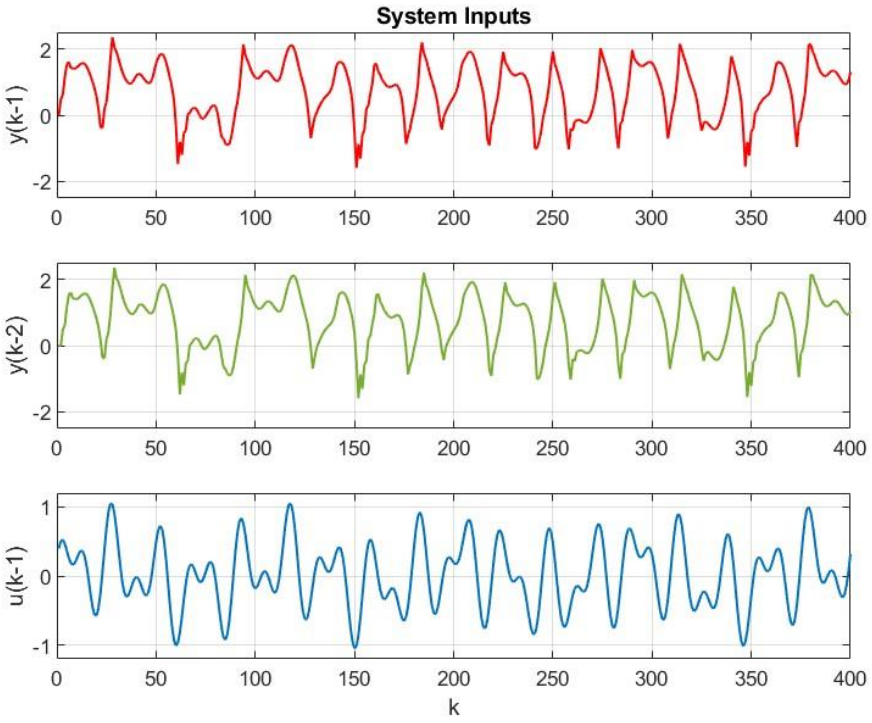
$$u(k) = 0.5\sin\left(\frac{\pi k}{11}\right) + 0.4\cos\left(\frac{\pi k}{6.5}\right) + 0.2\sin\left(\frac{\pi k}{45}\right) \quad 1 \le k < 400 \qquad (2)$$

ساخت مجموعه داده

به کمک نرم‌افزار متلب توابع داده شده را با رعایت قیدها برای نمونه‌های خواسته شده محاسبه می‌کنیم. در این مرحله مقدار دلخواه $\beta = 1.1$ را در نظر می‌گیریم. تمامی برنامه‌ها در بخش ضمیمه آورده شده‌اند. در این مرحله از اشاره به کدهای متلب نوشته شده اجتناب می‌کنیم. خروجی سیستم داده شده به شکل زیر می‌باشد.



شکل ۱- خروجی سیستم اصلی $y(k)$



شکل ۲- ورودی‌های سیستم بر اساس پاسخ سیستم در دوره‌های قبل و سیگنال کنترلی $u(k)$

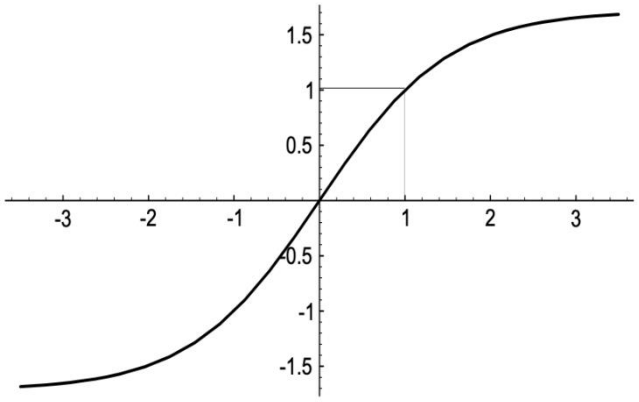
در این مرحله مشاهده می‌کنیم که سیستم ما دارای ۴۰۰ نمونه می‌باشد. حال با در دست داشتن ورودی‌ها و پاسخ صحیح سیستم به ازای هر نمونه، می‌توانیم شبکه عصبی دلخواه را بنا نهاده و آموزش دهیم. اما پیش از آن باید بر مجموعه داده حاضر پیش پردازش‌ها و تغییراتی ایجاد شوند.

پیش‌پردازش داده‌های خروجی دلخواه

بسیار مهم است پاسخ‌های سیستم در بازه کاری توابع فعال ساز S شکل (Sigmoid) قرار بگیرند. در اینجا، از تابع تانژانت هایپربولیک به عنوان تابع فعال‌ساز استفاده می‌کنیم. این تابع به صورت زیر تشکیل می‌شود.

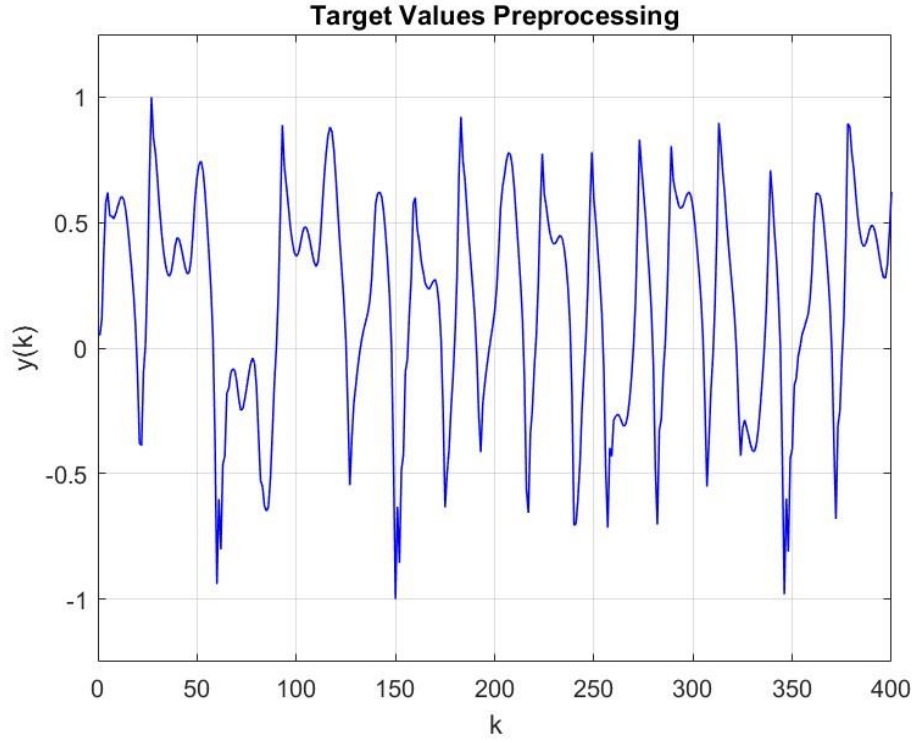
$$\varphi(v) = a\tan h(bv) \qquad (3)$$

با جایگذاری $\alpha = 1.7159$ و $\beta = 2/3$ در معادله (۳)، تابع فعال‌ساز به شکل زیر خواهد شد.



شکل ۳- تابع فعال‌ساز تانژانت هایپربولیک براساس مقادیر $\alpha = 1.7159$ و $\beta = 2/3$

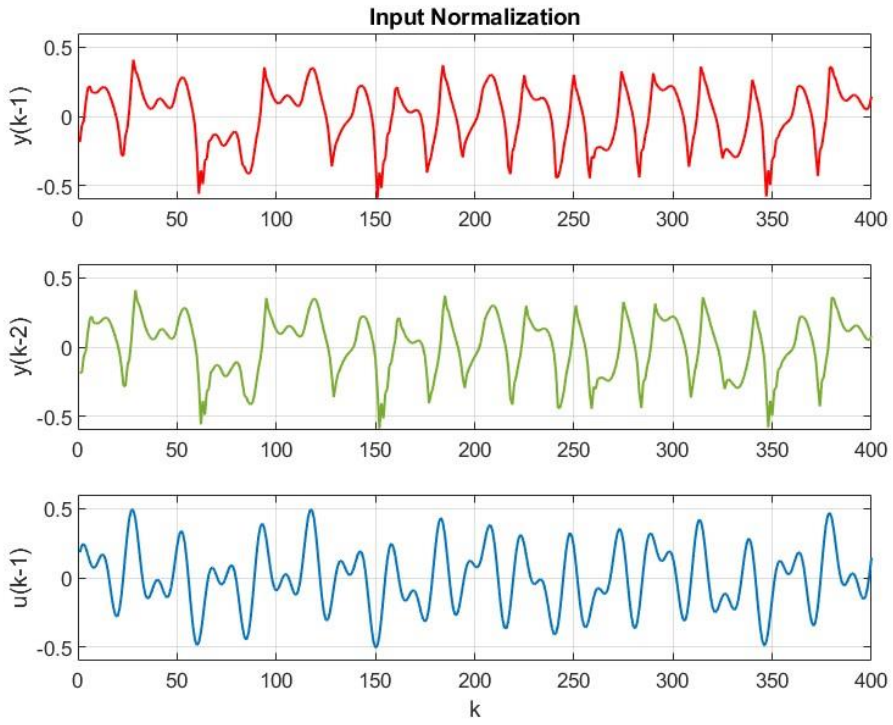
با توجه به نمودار تابع فعال ساز خروجی‌های دلخواه سیستم نیز باید با یک فاصله و آستانه‌ای از کران‌های تابع تانژانت هایپربولیک قرار گیرند. به طور مثال، بازه [1,1-] می‌تواند انتخاب مناسبی باشد. این انتخاب باعث می‌شود تا خروجی‌ها به ناحیه اشباع تابع فعال‌ساز متمایل نشده و در نهایت فرایند یادگیری کند نشود. خروجی این تغییرات بر داده‌های خروجی در دیتاست به شکل زیر است.



شکل ۴- داده‌های دلخواه خروجی پیش‌پردازش شده

نرمال‌سازی داده‌های ورودی

علاوه بر داده‌های خروجی داده‌های ورودی نیز دستخوش تغییراتی می‌شوند. میانگین ورودی‌ها در کل مجموعه داده باید نزدیک به صفر باشند. در غیر این صورت در عمل ممکن است شاهد نوساناتی حول مقداری از خطای یادگیری و یا حتی پیوستگی افزایش یا کاهش همگی وزن‌ها بدون توازن و تنها به دلیل وجود دریفت خواهیم بود. ابتدا مقدار میانگین کل را از تمامی المان‌ها در دیتاست کم کرده و سپس با اعمال ضریبی، مقدار کران داده‌ها را محدود می‌کنیم. داده‌های نرمال‌سازی شده به شکل زیر خواهند بود.



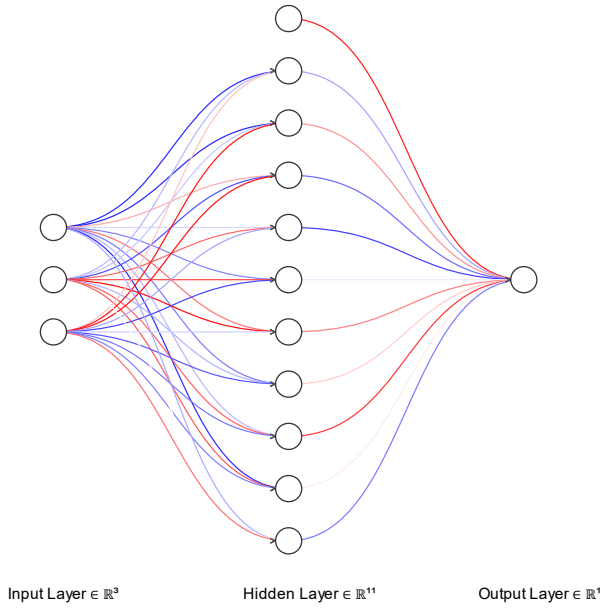
شکل ۵- ورودی‌های نرمال‌سازی شده

حال که پیش پردازش به درستی بر روی همگی داده‌ها اعمال شده است، با در دست داشتن مجموعه داده‌ها اندک اندک وارد مرحله تشکیل شبکه عصبی دلخواه می‌شویم. ابتدا مجموعه داده را به سه بخش آموزش، اعتبارسنجی، و آزمون به ترتیب با نسبت‌های 0.6، 0.2، 0.2، تقسیم می‌کنیم.

پرسپترون چند لایه

معماری شبکه پرسترون چند لایه متشکل از یک لایه ورودی و خروجی، و یک یا دو لایه پنهان می‌باشد. در مسئله ما، تعداد ورودی‌ها بدون احتساب مقدار آستانه برابر با سه ورودی، و تعداد نورون‌های خروجی معادل با یکی می‌باشد. تعداد لایه‌های پنهان و نورون‌های آنها به انتخاب می‌توانند دلخواه باشند. در مرحله اول فرض می‌شود که شبکه تنها دارای یک فضای ویژگی می‌باشد. و تعداد نورون‌های این لایه برابر با ده نورون باشد.

می‌دانیم، پرسپترون چندلایه از الگوریتم پس انتشارخطا برای یادگیری وزن‌ها و محاسبه تاثیر هر یک در ایجاد خطا استفاده می‌کند. در واقع فرایند آموزش در این شبکه شامل دو بخش است: فاز پیشخورد و پس انتشار خطا. در ادامه به توضیح پیاده سازی این شبکه می‌پردازیم.



شکل ۶- معماری پرسپترون چند لایه با یک فضای ویژگی به همراه مقدار آستانه

مقداردهی اولیه وزن‌ها

بدون دانش اولیه وزن‌ها و مقادیر آستانه را با یک توزیع یکنواخت و میانگین صفر مقداردهی می‌کنیم. این کار باعث می‌شود توزیع میدان‌های محلی هر نورون در ناحیه خاصی از تابع فعال‌ساز قرار گیرد. به این ترتیب، مقدار اولیه وزن‌های هر نورون وابسته به تعداد اتصالات گذشته آن است. بدین ترتیب، با درنظر نگرفتن مقادیر آستانه، در صورتی که m تعداد اتصالات هر لایه به لایه‌های قبلی خود باشد، توزیع وزن‌ها به صورت زیر خواهد بود.

$$\sigma_{w0} = m^{-\frac{1}{2}} \quad (4)$$

در نتیجه، انتظار می‌رود مقادیر اولیه وزن‌ها در در بین لایه ورودی و پنهان به نسبت بزرگ‌تر از مقادیر این وزن‌ها در بین لایه‌های پنهان و خروجی باشند. در ادامه نمونه‌ای از مقادیر اولیه وزن‌ها را برای معماری نشان داده شده در شکل (۶) مشاهده می‌کنید. دقت کنید که مقادیر اولیه آستانه برابر با صفر در نظر گرفته شده‌اند.

```
wji = 10×4
          0    0.2923    0.1881    0.4790
          0    0.0648    0.5275   -0.0092
          0   -0.1383    0.9097    0.3080
          0    0.1195    0.1012   -0.1071
          0    0.0121   -0.2634    0.2204
          0   -0.1215    0.2678    0.6384
          0    0.5903   -0.4400    0.0523
          0    0.0738   -0.0913   -0.1002
          0    0.9101    0.0906   -0.1667
          0   -0.0987    0.4965    0.2388

wkj = 1×11
          0    0.0334    0.0531    0.0014    0.0041   -0.0158    0.0403
   -0.0019   -0.0118    0.0546    0.0202
```

فاز پیش خورد (Feed Forward)

در این مرحله، بدون تغییر دادن وزن‌ها، مقدار خروجی نرون‌ها و در نهایت مقدار خروجی تخمین زده شده سیستم را به دست می‌آوریم. در نهایت، سیگنال‌های لحظه‌ای خطا و انرژی که به صورت زیر تعریف می‌شوند را بدست می‌آوریم. در ادامه، به کمک این سیگنال‌ها، الگوریتم پس انتشار خطا را پیاده سازی می‌کنیم.

$$e_j(n) = d_j(n) - o_j(n) \quad (5)$$

$$\varepsilon_j(n) = \frac{1}{2} e_j^2(n) \quad (6)$$

پس‌انتشار خطا (Back Propagation)

در این مرحله، با استفاده از مقادیر خروجی نرون‌ها در مرحله قبل، لایه به لایه به عقب بر گشته و اثر هر وزن برروی خطای ایجاد شده را به کمک گرادیان محلی محاسبه می‌کنیم. معادله محاسبه گرادیان محلی برای وقتی که j یک نورون در لایه خروجی و در لایه پنهان باشد به ترتیب زیر آمده است.

$$\delta_j(n) = e_j(n)\varphi_j'\left(v_j(n)\right) \tag{7}$$

$$\delta_j(n) = \varphi_j'\left(v_j(n)\right)\sum_k \delta_k(n)w_{ki}(n) \tag{8}$$

همچنین، محاسبه مشتق تابع فعال‌ساز نسبت به v_j به سادگی معادل زیر بدست می‌آیند.

$$\varphi_j'\left(v_j(n)\right) = \frac{b}{a}\left[a - y_j(n)\right]\left[a + y_j(n)\right] \tag{9}$$

در نهایت، به کمک رابطه عمومی دلتا، مقدار وزن‌ها را بروزرسانی می‌کنیم. در این رابطه، η معادل ضریب آموزش بوده، و α معادل تکانه (momentum) می‌باشد.

$$\Delta w_{ji}(n) = \alpha w_{ji}(n-1) + \eta\delta_j(n)y_i(n) \tag{10}$$

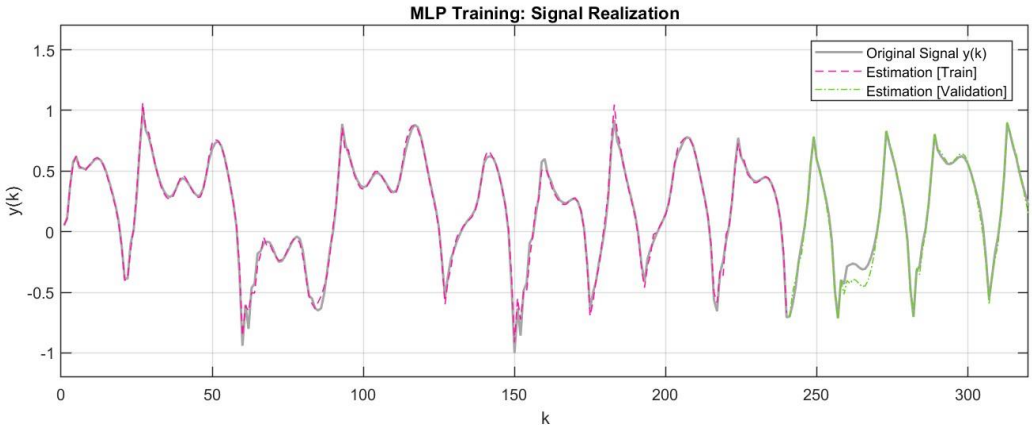
اعتبارسنجی و معیارهای اتمام یادگیری

در نهایت، این فرایند را برای تمامی داده‌های آموزش انجام داده و دوره را دوباره تکرار می‌کنیم. همانطور که در گذشته نیز اشاره شد، بخشی از داده‌ها به جهت اعتبارسنجی (validation) داده‌های آموزش استفاده می‌شوند. در انتهای هر دوره، فاز پیشخورد را به کمک وزن‌های به دست آمده از آموزش اینبار برروی داده‌های اعتبار سنجی پیاده‌سازی می‌کنیم. خطای گزارش شده در این مرحله اطلاعات مهمی از فرایند یادگیری در اختیار قرار می‌دهد چرا که می‌تواند میزان overfitting آموزش را به خوبی نشان دهد. به همین دلیل می‌تواند معیار خوبی برای اعلام اتمام سریعتر آموزش باشد.

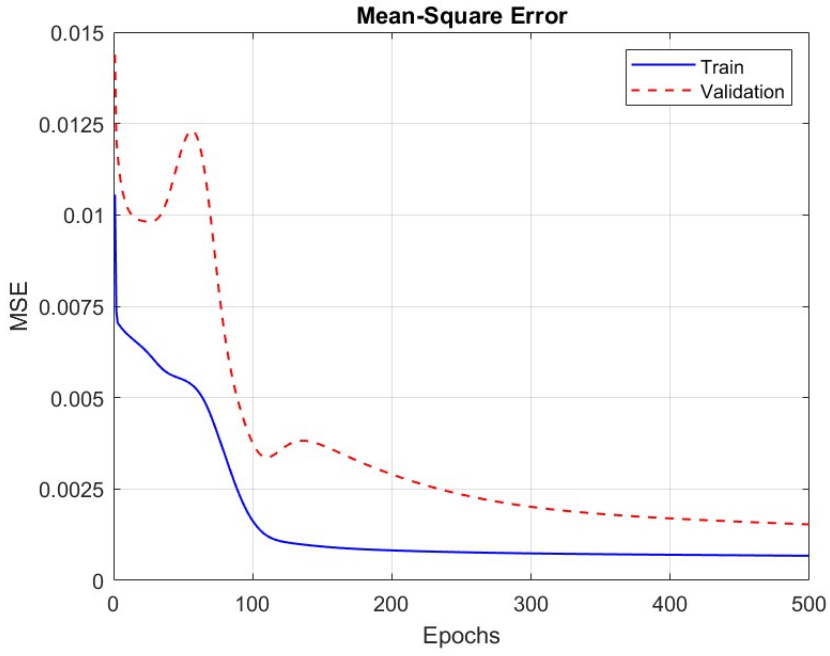
یکی دیگر از معیارهای اتمام یادگیری، تعیین حداقلی برای خطای آموزش است. همچنین می‌توانیم مقدار بیشینه‌ای را برای دوره تکرار آموزش (epoch) نیز در نظر بگیریم.

حالت شماره ۱

در این قسمت شبکه را با معماری شکل (۶) تشکیل می‌دهیم. دیتاست ۴۰۰ نمونه‌ای خود را با نسبت‌های ۰.۶، ۰.۲، ۰.۲ به زیرداده‌های آموزش، اعتبارسنجی، و آزمون تقسیم می‌کنیم. در این مرحله، تعداد نورون‌های لایه پنهان برابر ۱۰ عدد می‌باشند. ضریب آموزش معادل ۰.۲ و از ضریب تکانه صرف نظر می‌کنیم. خروجی سیستم به شکل زیر می‌باشد.



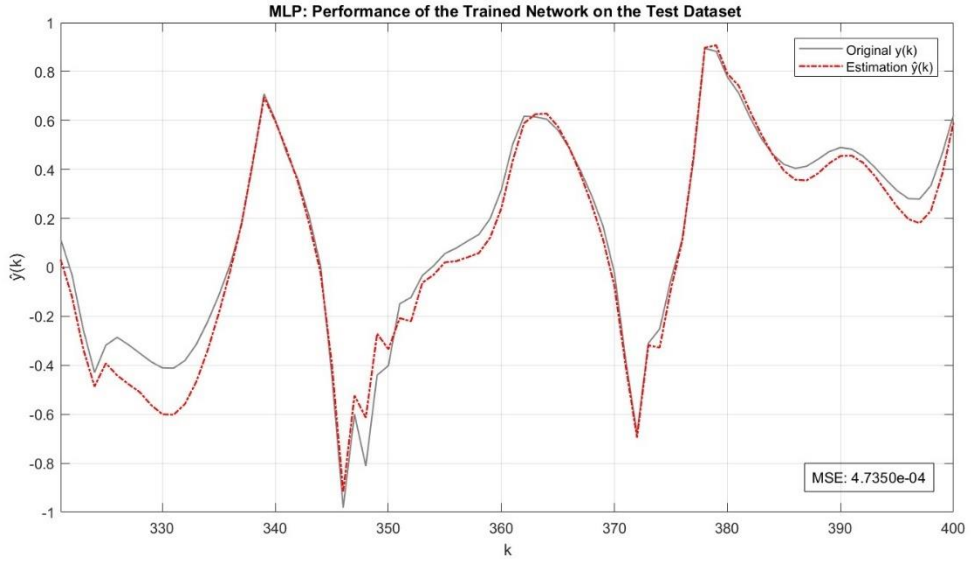
(الف)



(ب)

شکل ۷– تعداد ده نورون در لایه پنهان، ضریب آموزش ۰.۲ (الف) نمودار مقایسه تخمین خروجی برای داده‌های آموزش و اعتبارسنجی (ب) خطای حداقل مربعات برای داده‌های آموزش و اعتبارسنجی

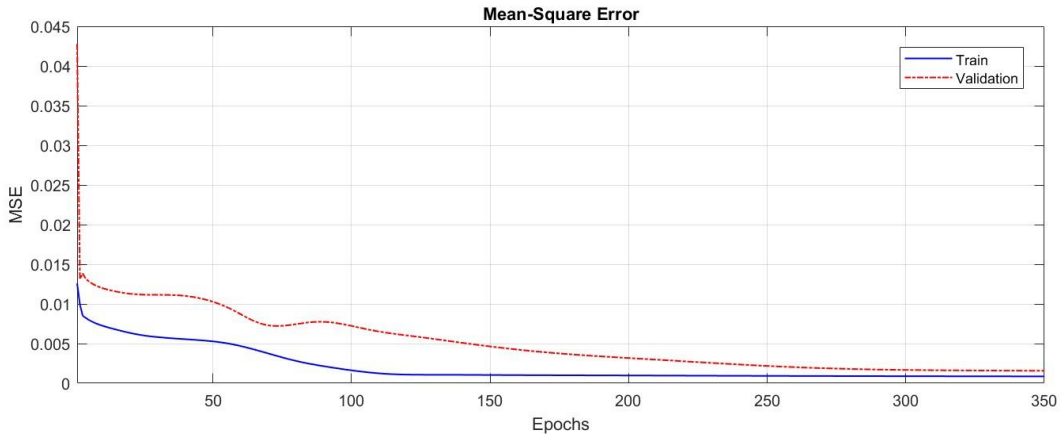
مشاهده می‌شود در قسمت‌هایی که داده‌های آموزش به خوبی فراگرفته شده‌اند، خطای داده‌های اعتبارسنجی رو به افزایش بوده است. این بدان معناست که در آن نواحی، شبکه در مسیر overfitting قرار گرفته است. اما این رویکرد در گذر زمان همگام هم همگرایی و نزول خطای آموزش، کاهش یافته است. مقایسه عملکرد شبکه با سیستم اصلی در داده‌های تست به صورت زیر است. خطای شبکه برروی داده تست معادل 4.7350e-04 می‌باشد.



شکل ۸– تعداد ده نورون در لایه پنهان، ضریب آموزش ۰.۲؛ نمودار مقایسه تخمین خروجی برای داده‌های آزمون

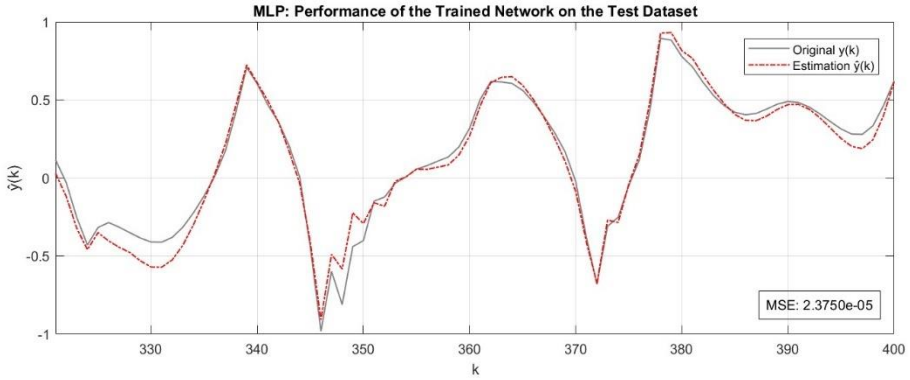
حالت شماره ۲

تعداد نورون‌های لایه پنهان برابر ۱۰ عدد می‌باشند. ضریب آموزش معادل ۰.۲۵ و ضریب تکانه برابر با ۰.۱۵ می‌باشد.



شکل ۹– تعداد ده نورون در لایه پنهان، ضریب آموزش ۰.۲۵، ضریب تکانه ۰.۱۵؛ میانگین مربعات خطا در آموزش و اعتبارسنجی

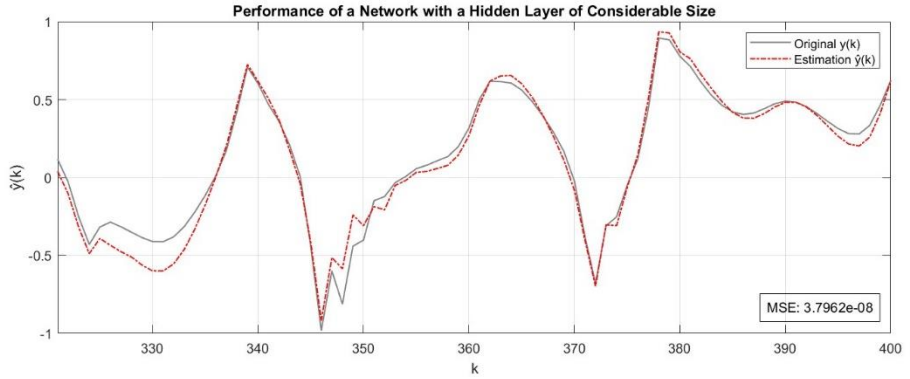
مشاهده می‌شود که رابطه میان یادگیری و اعتبار سنجی بسیار بهبود یافته است. با اینحال، با انجام صحیح و خطا متوجه می‌شویم که افزایش ضرایب آموزش و تکانه هر دو موجب ناپایداری و نوساناتی در خطا می‌شوند. بدین ترتیب، مجموع و نسبت این دو ضریب باید مقدار قابل قبول به نسبت کوچک و ثابتی باشد. مقایسه عملکرد شبکه با سیستم اصلی در داده‌های تست به صورت زیر است. خطای شبکه برروی داده تست معادل 2.3750e-05 می‌باشد.



شکل ۱۰– تعداد ده نورون در لایه پنهان، ضریب آموزش ۰.۲۵، ضریب تکانه ۰.۱۵؛ مقایسه تخمین خروجی برای داده‌های آزمون

حالت شماره ۳

اشکال استفاده از شبکه با یک لایه پنهان آن است که در آن تمامی نورون‌ها به صورت جامع در تعامل با یکدیگر هستند. بدان معنا که هر نورون لزوما ویژگی منحصر به فردی را یاد نمی‌گیرد در عین حال آموزه‌هایش بر عملکرد دیگر نورون‌ها اثرگذار است. این بخش نشان می‌دهد که در شبکه‌ای با یک لایه پنهان، از نقطه‌ای دیگر نمی‌توان عملکر شبکه را بهبود بخشید بدون اینکه عملکرد بخشی آسیب نبیند. در این قسمت، تعداد نورون‌ها در تنها لایه پنهان برابر با ۱۰۰ تعیین شده است. همچنین، ضریب آموزش معادل ۰.۲ و از ضریب تکانه صرف نظر شده است. مشاهده می‌کنیم بر خلاف تغییرات قابل توجه در ساختار شبکه، میزان عملکرد بهبود چشمگیری پیدا نمی‌کند.



شکل ۱۰– تعداد صد نورون در لایه پنهان، ضریب آموزش ۰.۲۵؛ مقایسه تخمین خروجی برای داده‌های آزمون

نتیجه‌گیری

می‌توان نتیجه گرفت که MLP یک شبکه بسیار قدرتمند است که در ساده ترین شکل می‌تواند نتیجه‌های خوبی را خروجی دهد. با این حال چند نکته در مورد این شبکه وجود دارد. اول آنکه برای آموزش این شبکه به داده‌های زیاد و پیش پردازش‌هایی مانند حذف کردن میانگین، از بین بردن کوواریانس بین داده‌ها، و تدارک دیدن شرایطی که در آن خروجی در بازه سیگنال تابع فعال‌ساز قرار گیرد؛ نیاز داریم. همچنین، به همان دلیل توابع فعال‌ساز، مقدار دهی اولیه وزن‌ها اهمیت بسیار دارد. در غیر این صورت، شیکه اشباع شده و یادگیری کند می‌شود.

همچنین، برای بهبود عملکرد پرسپترون چند لایه، می‌توانیم از دو لایه پنهان استفاده کنیم. در این حالت لایه پنهان اول ویژگی‌های محلی را فرا گرفته، در حالی که لایه دوم ویژگی‌های عمومی‌تری را یاد خواهد گرفت. از دیگر راهکارهایی که می‌تواند موجب افزایش عملکرد این شبکه شود عبارتند از: استفاده از ضرایب آموزش متفاوت و متغیر، استفاده از ماتریس هس، و تنظیم‌گر لاندا (regularization).

باتشکر از توجه شما،

عرفان ریاضتی

پیوست

برنامه متلب نوشته شده برای این قسمت به شرح زیر است.

```
clear; close all; clc; format compact;
%%
N = 400;

k = 1:N;
desired_origin = d(N);
inputs_origin = x(N);

figure
plot(k, desired_origin);

figure;
subplot(3,1,1);
plot(k, inputs_origin(1,:));
subplot(3,1,2);
plot(k, inputs_origin(2,:));
subplot(3,1,3);
plot(k, inputs_origin(3,:));
%%
% offset the desired outputs between sigmoidal bound
max_d = max(desired_origin);
min_d = min(desired_origin);
width = max_d - min_d;
offset = (max_d + min_d) / 2;
desired_norm = 2 * (desired_origin - offset) / width;

figure
plot(k, desired_norm);
%%
% mean removal preprocess on the input data
inputs_norm = inputs_origin;
mean_inputs = [mean(inputs_norm(1,:)); mean(inputs_norm(2,:));
mean(inputs_norm(3,:))];
inputs_norm = inputs_norm - mean_inputs;

max_inputs = [max(inputs_norm(1,:)); max(inputs_norm(2,:)); max(inputs_norm(3,:))];
min_inputs = [min(inputs_norm(1,:)); min(inputs_norm(2,:)); min(inputs_norm(3,:))];
width_inputs = max_inputs - min_inputs;

inputs_norm = inputs_norm ./ width_inputs;

figure;
subplot(3,1,1);
plot(k, inputs_norm(1,:));
subplot(3,1,2);
plot(k, inputs_norm(2,:));
subplot(3,1,3);
plot(k, inputs_norm(3,:));
%%
```

```
% shuffling the data presented to the network
% inputs_random = zeros(size(inputs_norm));
% desired_random = zeros(size(desired_norm));
%
% j = 0;
% index_shuffle = randperm(N);
% for i = index_shuffle
%     j = j + 1;
%     inputs_random(:, j) = inputs_norm(:, i);
%     desired_random(j) = desired_norm(i);
% end
%
% figure
% plot(k, desired_random);
%
% figure;
% subplot(3,1,1);
% plot(k, inputs_random(1,:));
% subplot(3,1,2);
% plot(k, inputs_random(2,:));
% subplot(3,1,3);
% plot(k, inputs_random(3,:));
%%
dimIn = 3;          % number of inputs exc. bias
dimOut = 1;         % number of outputs
dimHidden = [100];    % dimensions of each hidden layer ex. bias

nHiddenLayers = size(dimHidden, 2);

N = 400;
trainRatio = 0.6;
validRatio = 0.2;
testRatio = 0.2;

nTrain = uint16(N * trainRatio);
nValid = uint16(N * validRatio);
nTest = uint16(N * testRatio);

maxEpoch = 500;     % maximum number of iterations
eta = 0.25;
alpha = 0.0;
%%
% adding +1 row for including synaptic threshold
xi = ones([size(inputs_norm, 1)+1, nTrain]);
xi(2:end, :) = inputs_norm(:, 1:nTrain);
dk = desired_norm(1:nTrain);
%%
% weight initialization
m = [dimIn, (dimIn+dimOut)*dimHidden(1)];
stand_dev = m.^(-1/2);

wji = zeros([dimHidden(1), (dimIn+1)]);
wkj = zeros([dimOut, (dimHidden(1)+1)]);

wji(:, 2:end) = normrnd(0, stand_dev(1)^2, [dimHidden(1), dimIn]);
```

```
wkj(:, 2:end) = normrnd(0, stand_dev(2)^2, [dimOut, dimHidden(1)]);

vj = zeros([dimHidden(1), 1]);
yj = zeros([(dimHidden(1)+1), 1]);
yj(1, :) = 1;

vk = zeros([dimOut, 1]);
yk = zeros([dimOut, nTrain]);
%%
ek = -1 * ones(size(dk));   % instantaneous error signal
E = (1/2) * ek.^2;          % total instantaneous error energy
MSE = zeros([1, maxEpoch]);

delta_k = 0;    % local gradient for the output neuron
delta_j = 0;    % local gradient for the hidden layer

delta_wji = zeros(size(wji));
delta_wkj = zeros(size(wkj));
%%
xvalid = ones([size(inputs_norm, 1)+1, nValid]);
xvalid(2:end, :) = inputs_norm(:, nTrain+1:nTrain+nValid);
dvalid = desired_norm(nTrain+1:nTrain+nValid);

yvalid = zeros([dimOut, nValid]);

ek_valid = -1 * ones(size(dvalid));   % instantaneous error signal
E_valid = (1/2) * ek_valid.^2;        % total instantaneous error energy
MSE_valid = zeros([1, maxEpoch]);
%%
% MLP
for epoch = 1:maxEpoch
    for n = 1:nTrain
        % feed forward phase
        vj = wji * xi(:,n);
        yj(2:end) = phi(vj);

        vk = wkj * yj;
        yk(n) = phi(vk);

        % back-propagation algorithm
        ek(n) = dk(n) - yk(n);
        E(n) = (1/2) * ek(n)^2;

        delta_k = ek(n) * phi_derivative(yk(n));
        delta_j = delta_k * wkj(2:end)' .* phi_derivative(yj(2:end));

        delta_wji = (alpha * delta_wji) + (eta * delta_j * xi(:, n)');
        wji = wji + delta_wji;

        delta_wkj = (alpha * delta_wkj) + (eta * delta_k * yj');
        wkj = wkj + delta_wkj;
    end

    % cross validation
    for n = 1:nValid
```

```
        vj = wji * xvalid(:, n);
        yj(2:end) = phi(vj);

        vk = wkj * yj;
        yvalid(n) = phi(vk);

        ek_valid(n) = dvalid(n) - yvalid(n);
        E_valid(n) = (1/2) * ek_valid(n)^2;
    end

    MSE(epoch) = sum(E)/numel(E);
    MSE_valid(epoch) = sum(E_valid)/numel(E_valid);
end
%%
figure;
plot(1:maxEpoch, MSE, 'b');
hold on;
plot(1:maxEpoch, MSE_valid, 'r');
%%
figure;
plot(k, desired_norm, 'r');
hold on
plot(1:nTrain, yk, 'b')
hold on;
plot(nTrain+1:nTrain+nValid, yvalid, 'g')
%%
xtest = ones([size(inputs_norm, 1)+1, nTest]);
xtest(2:end, :) = inputs_norm(:, nTrain+nValid+1:nTrain+nValid+nTest);
dtest = desired_norm(nTrain+nValid+1:nTrain+nValid+nTest);

ytest = zeros([dimOut, nTest]);
%%
% MLP test
for n = 1:nTest
    vj = wji * xtest(:, n);
    yj(2:end) = phi(vj);

    vk = wkj * yj;
    ytest(n) = phi(vk);
end

%%
ek_test = dtest - ytest;
E_test = (1/2) * ek_test(n).^2;
MSE_test = sum(E_test)/numel(E_test);
%%
figure;
plot(nTrain+nValid+1:nTrain+nValid+nTest, dtest, 'b');
hold on;
plot(nTrain+nValid+1:nTrain+nValid+nTest, ytest, 'r');
%%
% *Functions*
function [control_sig] = u(k)
    control_sig = 0.5*sin(k*(pi/11)) + 0.4*cos(k*(pi/6.5)) + 0.2*sin(k*(pi/45));
end
```

```
function [desired_vec] = d(N)
    alpha = 1.2;
    beta = 1.1;

    if N <= 1
        N = 1;
    elseif N >= 400
        N = 400;
    end

    desired_vec = zeros([1, N]);
    for k = 1:N
        if k < 3
            desired_vec(k) = alpha * u(k-1);
        else
            desired_vec(k) = alpha * ((desired_vec(k-1)*desired_vec(k-
2)*(desired_vec(k-2) + beta)) / (1 + ((desired_vec(k-1)^2)*(desired_vec(k-2)^2))) +
u(k-1));
        end
    end
end

function [input_vec] = x(N)
    if N <= 1
        N = 1;
    elseif N >= 400
        N = 400;
    end

    input_vec = zeros([3, N]);
    desired_vec = d(N);

    for k = 1:N
        if k == 1
            input_vec(:, k) = [    0,    0,  u(k-1)]';
        elseif k == 2
            input_vec(:, k) = [ desired_vec(1),    0,  u(k-1)]';
        else
            input_vec(:, k) = [ desired_vec(k-1),  desired_vec(k-2), u(k-1)]';
        end
    end
end

function [activation] = phi(v)
    a = 1.7159;
    b = 2/3;
    activation = a*tanh(b*v);
end

function [derivative] = phi_derivative(y)
    a = 1.7159;
    b = 2/3;
    derivative = (b/a)*(a - y).*(a + y);
end
```