باسمه تعالی

دانشگاه صنعتی شریف

دانشکده مهندسی برق

درس علوم اعصاب: یادگیری، حافظه، شناخت

پروژه اول

آموزش شبکه‌های عصبی بازگشتی انگیزشی ـ بازدارنده
(Excitatory-Inhibitory Recurrent Neural Networks)

استاد درس: دکتر کربلایی آقاجان

امیرحسین افشارراد

۹۵۱۰۱۰۷۷

محمدمهدی ابوالحسنی

۹۵۱۰۰۹۸۳

۳ آذر ۱۳۹۷

۱ مقدمه و معرفی مدل

۱.۱ تصحیح مدل مورد استفاده در تمرین قبل

ابتدا برای شروع به انجام شبیه‌سازی‌های مربوط به این پروژه، اقدام به تصحیح و تکمیل مدل مورد استفاده در تمرین قبل کردیم. در نتیجه‌ی این اقدام، خروجی‌های مطلوب حاصل شدند و دو مشکل اصلی مرتفع شد؛ اول مشکل عدم ارائه‌ی پاسخ صحیح توسط شبکه در تشخیص بیت پریتی که ناشی از عدم انجام کامل فرایند backpropagation بود؛ و دیگر مشکل بریده شدن خروجی سیگنال‌های سینوسی در مقادیر 1 و -1 که ناشی از وجود تابع فعال‌سازی tanh بود. قطعه کد زیر مدل این شبکه را نشان می‌دهد:

```
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        self.input2hidden = nn.Linear(input_size, hidden_size)
        self.hidden2hidden = nn.Linear(hidden_size, hidden_size)
        self.hidden2output = nn.Linear(hidden_size, output_size)
        self.NonlinearFunc = nn.Tanh()

    def forward(self, input, hidden):
        hidden = self.NonlinearFunc(self.input2hidden(input) + self.
            hidden2hidden(hidden))
        output = self.NonlinearFunc(self.hidden2output(hidden))
        return output, hidden

    def init_hidden(self):
        return Variable(torch.zeros(1, self.hidden_size))
```

۲.۱ تکمیل مدل بر مبنای مقاله‌ی مرجع

مدل مذکور در قسمت قبل با انجام تغییرات اندکی، به مدل جدید و مطلوب تبدیل می‌شود. معادله ۱ رابطه‌ای است که مبنای تعریف مدل مورد استفاده در این پروژه است.

$$\mathbf{x}_t = (1-\alpha)\mathbf{x_{t-1}} + \alpha(W^{rec}\mathbf{r_{t-1}} + W^{in}\mathbf{u_t}) + \sqrt{2\alpha\sigma^2_{rec}}\boldsymbol{N}(0,1) \qquad (۱)$$

در اثر اعمال تغییرات، کد مورد استفاده برای تعریف مدل به شکل زیر در می‌آید:

```
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
```

```
8
9           self.input2hidden = nn.Linear(input_size, hidden_size)
10          self.hidden2hidden = nn.Linear(hidden_size, hidden_size)
11          self.hidden2output = nn.Linear(hidden_size, output_size)
12          self.NonlinearFunc = nn.Tanh()
13
14      def forward(self, input, hidden):
15          a = 0.5
16          hidden = self.NonlinearFunc((1-a)*hidden + a*self.input2hidden(input) +
                a*self.hidden2hidden(nn.functional.relu(hidden)))
17          output = self.NonlinearFunc(self.hidden2output(hidden))
18          return output, hidden
19
20      def init_hidden(self):
21          return Variable(torch.zeros(1, self.hidden_size))
```

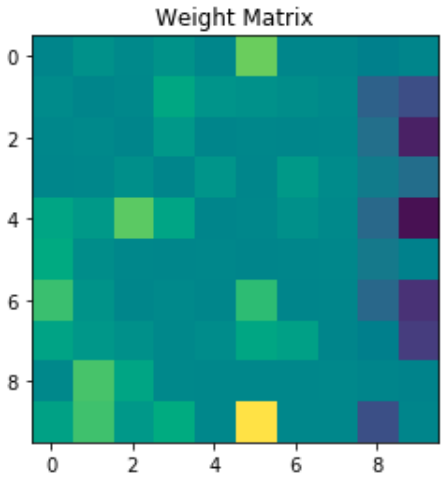
نکته‌ی قابل ذکر این است که به منظور پیاده‌سازی تسک‌هایی که خروجی‌های آن‌ها مقادیر باینری هستند، یک تابع غیرخطی tanh نیز به مدل اضافه شده است. ضمناً شایان ذکر است، به منظور ساده‌تر شدن مدل (و با پرسش از دستیار آموزشی درس) از نویز موجود در داخل نورون‌ها صرف نظر کردیم و صرفاً از ورودی‌های دارای نویز استفاده کردیم.
مورد دیگری که نیاز به انجام داشت تا به مدل توصیف‌شده در مقاله دست یابیم، تشکیل دسته‌های جداگانه‌ی نورون‌های انگیزشی و بازدارنده بود. این کار متناظر با آن است که بخشی از ستون‌های ماتریس وزن شبکه، مقادیر تماماً مثبت (یا صفر) و بخش دیگر، مقادیر تماماً منفی (یا صفر) داشته باشد. همچنین قطر اصلی ماتریس وزن نیز باید صفر باشد. می‌توان این محدودیت‌ها را در تعریف اولیه‌ی مدل اعمال کرد، اما روشی که ما در این پروژه مورد استفاده قرار دادیم، به شرح زیر است: در فرایند آموزش شبکه، هر بار که وزن‌های شبکه بروزرسانی می‌شوند، محدودیت‌های مطلوب را روی آن‌ها اعمال می‌کنیم. به عبارت دقیق‌تر، پس از هر بروزرسانی وزن‌ها، قطر اصلی ماتریس وزن را صفر کرده و مقادیر 80% از این ماتریس را با اعمال قدر مطلق مثبت کرده، و مقادیر باقیمانده را با روش مشابه به اجبار به مقادیر منفی (یا صفر) تبدیل می‌کنیم. قطعه کدی که این کار را انجام می‌دهد در ادامه قابل مشاهده است:

```
for j in range (hidden_size):
    list(rnn.hidden2hidden.parameters())[0].data[j, j].data.copy_(torch.tensor
        (0))
    for i in range (hidden_size):
        sign = 1
        if j >= hidden_size * 4 / 5:
            sign = -1
        if (list(rnn.hidden2hidden.parameters())[0].data[i, j].item() * sign <
            0):
            list(rnn.hidden2hidden.parameters())[0].data[i, j].data.copy_(torch.
                tensor(0))
```

همچنین یک نمونه از ماتریس‌های وزن پس از اعمال تغییرات مذکور در شکل ۱ قابل مشاهده است. (با کمی دقت) می‌توان مشاهده کرد که قطر اصلی تماماً صفر است و دو ستون انتهایی، مقادیری کمتر یا مساوی صفر دارند، در حالی که هشت ستون اول دارای مقادیر نامنفی‌اند.

۲ تسک اول: Perceptual Decision Making

این تسک در متن مقاله به دو شکل متفاوت پیاده‌سازی شده است که در این جا به بررسی هر دو روش می‌پردازیم. در ادامه نیز اتصالات شبکه را به گونه‌ای (که در ادامه توضیح داده خواهد شد) تغییر می‌دهیم و با یک روش سوم نیز به پیاده‌سازی مسأله می‌پردازیم.



شکل ۱: ماتریس وزن با محدودیت صفر بودن قطر اصلی و وجود نورون‌های انگیزشی و بازدارنده

۱.۲ تولید دیتاست

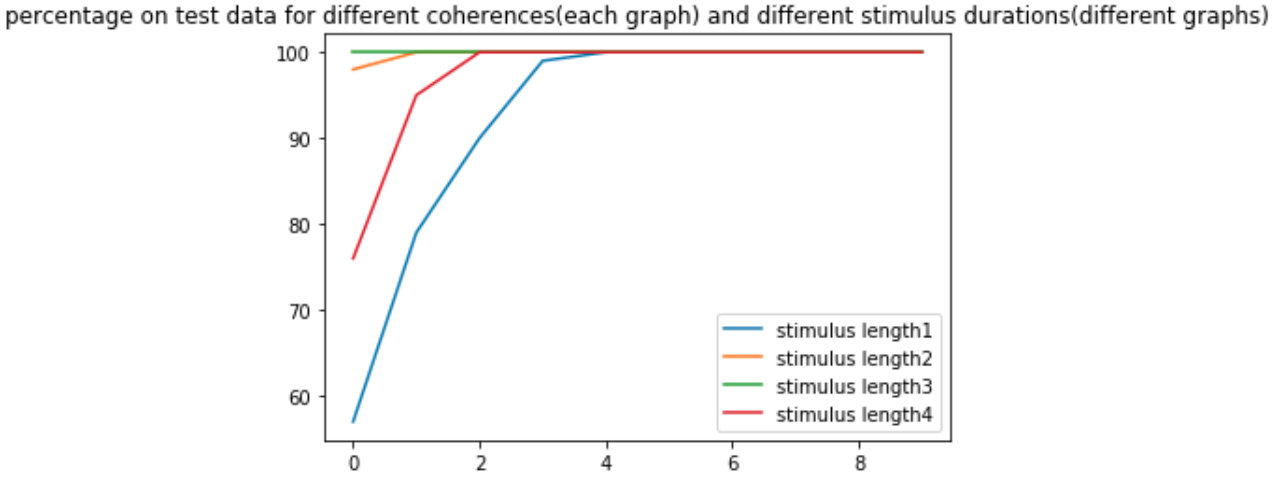
آزمایش مرجع مربوط به این بخش، آزمایش حرکت تصادفی نقاط بوده است. برای مدل‌سازی این آزمایش، نیاز داریم درصد حرکت نقاط در دو جهت مختلف را به گونه‌ای در ورودی مدل کنیم. می‌دانیم که در آزمایش اصلی، هرچقدر درصد نقاطی که در یک جهت حرکت می‌کنند بیشتر باشد، استنتاج برای تعیین جهت حرکت عمومی نقاط ساده تر است. برای مدل کردن این رفتار، از دو مقدار ثابت استفاده می‌کنیم که نماد درصد حرکت نقاط هستند؛ بنابراین هر چه این دو مقدار به یک‌دیگر نزدیک‌تر باشند، تصمیم‌گیری سخت‌تر خواهد بود. به منظور پیاده‌سازی این دیتاست، تابع `decision_making_dataset` نوشته شده و در کد مرجع موجود است.
یک نکته‌ی مهم آن است که در متن مقاله زمان شروع ورودی به شبکه اعلام می‌شود که ما برای سادگی، تحریک را از لحظه‌ی صفر می‌دهیم و این ورودی را حذف می‌کنیم.

۲.۲ پیاده‌سازی اول: تصمیم‌گیری پس از مشاهده کل ورودی

در این پیاده‌سازی، شبکه پس از مشاهده‌ی تمام ورودی‌ها، تصمیم خود را برای انتخاب یکی از دو ورودی (متناظر با یکی از دو جهت در آزمایش اصلی) اعلام می‌کند. طبق تعریف اولیه‌ی شبکه، پس از اعمال هر ورودی در واحد زمان، شبکه یک خروجی اعلام می‌کند. برای سهولت و یک‌پارچگی بیشتر، تعریف اولیه شبکه را تغییر نمی‌دهیم و به همین شکل باقی می‌گذاریم؛ و نحوه‌ی استفاده‌ی خود از شبکه را به گونه‌ی مطلوب تنظیم می‌کنیم. برای این کار، پس از اتمام هر ورودی در طول زمان، تنها آخرین خروجی را مد نظر قرار می‌دهیم و با مقایسه‌ی آن با خروجی مطلوب، وزن‌ها را بروزرسانی می‌کنیم. دقت کنید که خروجی‌ها دو بیتی هستند که یکی صفر و دیگری یک است که نشان‌دهنده‌ی تصمیم شبکه است. شکل ۲ نمودار درصد پاسخ‌گویی شبکه روی داده‌ی تست را به ازای تحریک‌هایی با طول مختلف (نمودارهای مختلف) و به ازای مقادیر مختلف تفاوت دو ورودی (هر نمودار) نشان می‌دهد. مشاهده می‌شود که با افزایش تفاوت دو مقدار، درصد پاسخ‌گویی زیاد می‌شود. همچنین به طور تقریبی مشاهده می‌شود که افزایش طول تحریک نیز موجب بهبود نتایج می‌گردد.

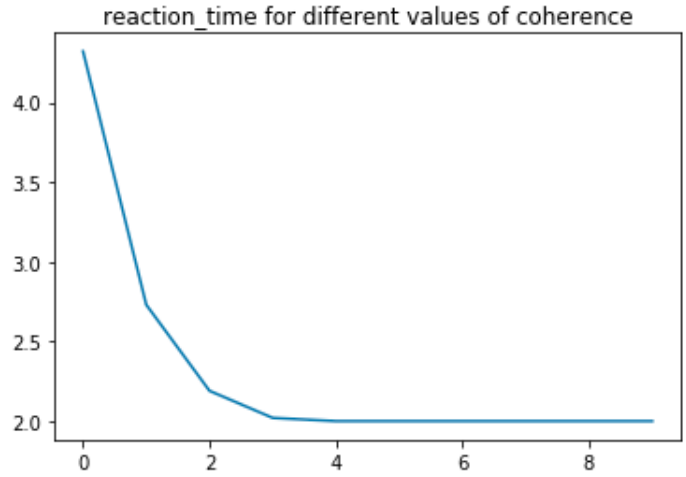
۳.۲ پیاده‌سازی دوم: تصمیم‌گیری به محض رسیدن به تصمیم قطعی

در دومین پیاده‌سازی مربوط به این بخش، شبکه نیازی به صبر کردن تا انتهای ورودی را ندارد و به محض این که به تصمیم قطعی برسد، تصمیم خود را اعمال می‌کند. برای پیاده‌سازی این فرایند، شرط تصمیم قطعی را به این شکل تعریف کردیم: اگر خروجی شبکه در سه واحد زمانی متوالی یکسان باشد، شبکه به تصمیم قطعی رسیده است.
واضح است که می‌توان تعاریف دیگری را نیز برای این مدل‌سازی در نظر گرفت.
شکل‌های ۳ و ۴ به ترتیب زمان واکنش شبکه به ازای مقادیر مختلف تفاوت مقدار تحریک‌ها، و نیز درصد پاسخ صحیح

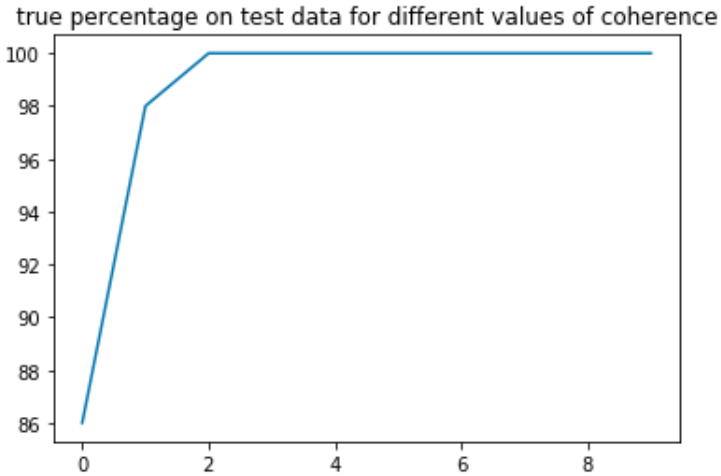


شکل ۲: نمودارهای درصد پاسخ صحیح دیتای تست برای طول‌های مختلف تحریک و تفاوت مقادیر دو تحریک

شبکه در این شرایط را نمایش می‌دهند. مشاهده می‌شود که با افزایش تفاوت دو ورودی، سرعت پاسخ شبکه و نیز دقت آن افزایش می‌یابد.



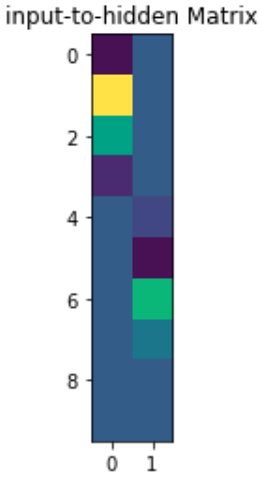
شکل ۳: زمان واکنش شبکه به ازای مقادیر مختلف تفاوت مقدار دو تحریک



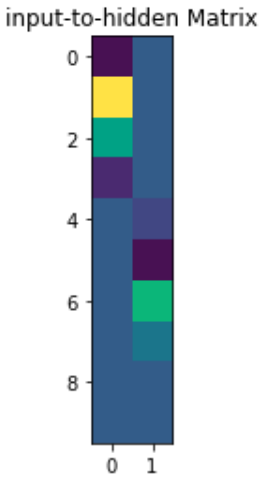
شکل ۴: درصد پاسخ صحیح شبکه به دیتای تست برای مقادیر مختلف تفاوت مقدار دو تحریک

۴.۲ پیاده‌سازی دوم: محدودیت الگوی اتصال نورون‌ها

در این بخش، اتصالات نورون‌ها را محدود می‌کنیم، به گونه‌ای که نورون‌های دریافت‌کننده‌ی ورودی اول و دوم جدا باشند. بنابراین بین این نورون‌ها اتصالی وجود نخواهد داشت تصمیم‌گیرندگان مربوط به هر یک از دو ورودی جدا خواهند بود. در این حالت شکل‌های ۵، ۶، و ۷ به ترتیب نشانگر ماتریس اتصالات ورودی به لایه‌ی درونی، ماتریس اتصالات درونی، و ماتریس اتصالات لایه‌ی درونی به خروجی می‌باشند. مشاهده می‌شود که در این حالت مقادیر صفر در نواحی معینی از این ماتریس‌ها وجود دارد. برای مشخص شدن بهتر موضوع، شکل ۸ را در نظر بگیرید که شکل متناظر با این فرایند در مقاله‌ی مرجع است.



شکل ۵: ماتریس اتصالات ورودی به لایه‌ی درونی

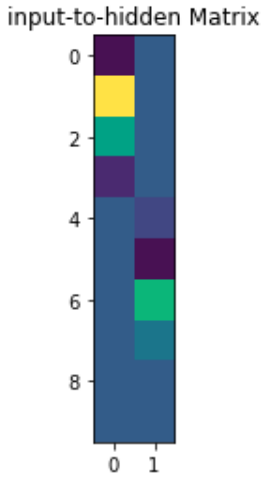


شکل ۶: ماتریس اتصالات درونی

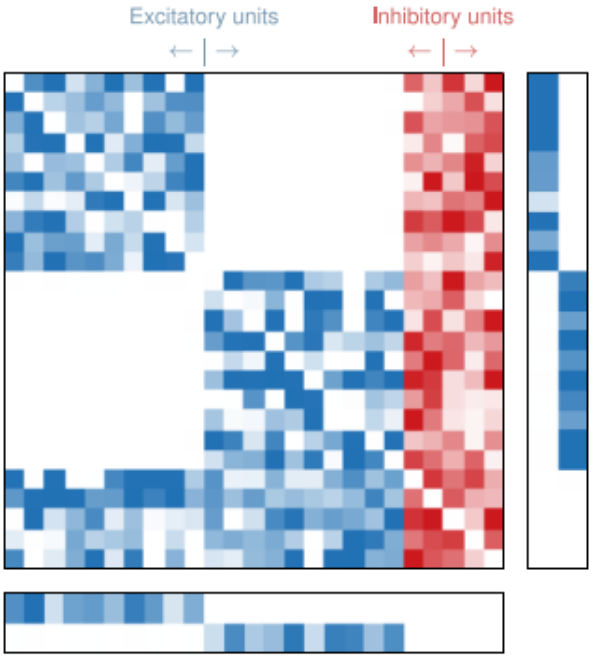
در این حالت میزان زمان واکنش و درصد پاسخ صحیح آن برای مقادیر مختلف تفاوت مقدار دو ورودی در شکل‌های ۹ و ۱۰ قابل مشاهده است. مجدداً مشاهده می‌شود که افزایش تفاوت دو ورودی موجب افزایش سرعت و دقت شبکه در پاسخ‌گویی می‌شود.

۳ تسک دوم: Parametric Working Memory

این تسک مربوط به مقایسه‌ی دو فرکانس مختلف و انتخاب فرکانس بالاتر است. تفاوت این تسک با تسک قبلی در آن است که این بار، دو ورودی نه به صورت هم‌زمان، بلکه به صورت متوالی (و با وجود یک وقفه‌ی کوتاه بین دو ورودی) به شبکه



شکل ۷: ماتریس اتصالات لایه‌ی درونی به خروجی



شکل ۸: شکل موجود در مقاله‌ی مرجع متناظر با شکل‌های ۵ و ۶ و ۷

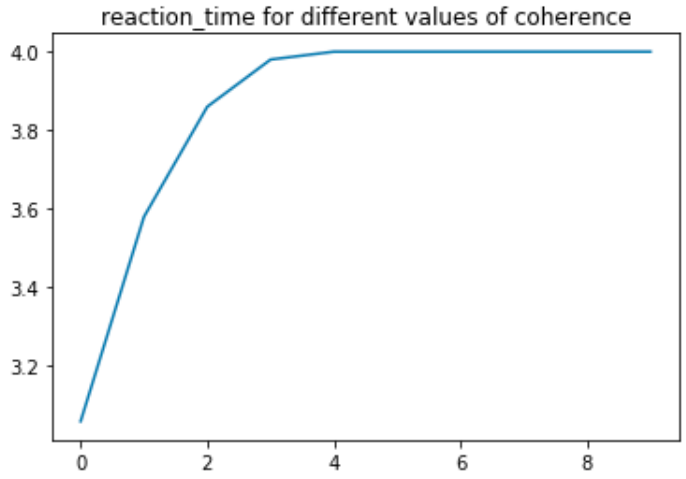
اعمال می‌شوند.

۱.۳ تولید دیتاست

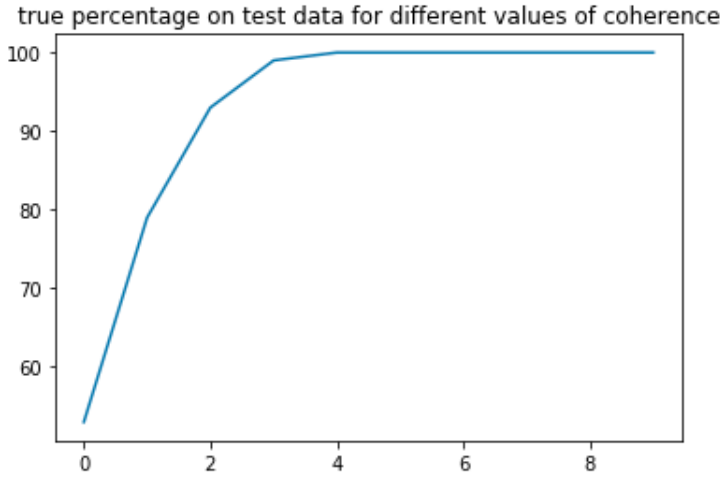
برای مدل کردن این ورودی‌ها، طبق توضیح مقاله، از دو مقدار عددی (همراه با نویز) استفاده می‌کنیم. نمونه‌ای از ورودی‌های تولیدشده در شکل ۱۱ قابل مشاهده است.

۲.۳ پیاده‌سازی تسک

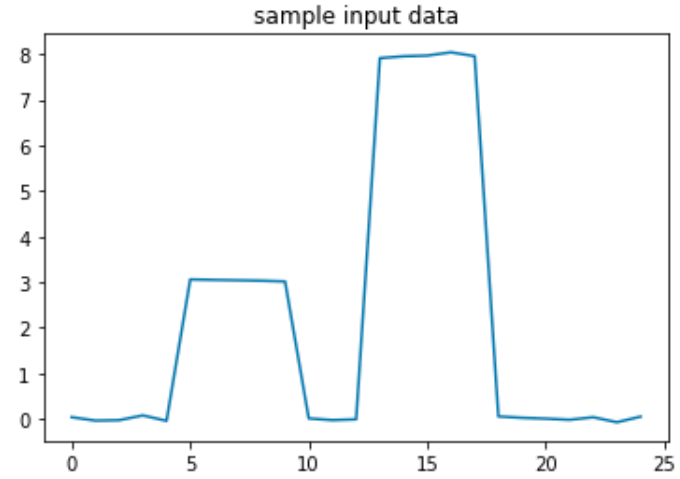
پیاده‌سازی این تسک به صورت اعمال یک ورودی (شکل ۱۱) به شبکه و دریافت دو خروجی است که متناظر با مقدار (فرکانس) اول و مقدار (فرکانس) دوم هستند؛ بنابراین هر کدام از این دو که توسط شبکه بزرگ‌تر شناخته شوند، در خروجی مقدار یک خواهند داشت و دیگری دارای مقدار صفر خواهد بود. شکل ۱۲ درصد پاسخ صحیح شبکه را به دیتای تست برای مقادیر مختلف تفاوت مقدار دو تحریک نشان می‌دهد. مشاهده می‌شود که با افزایش تفاوت مقدار دو تحریک (تفاوت در



شکل ۹: زمان واکنش شبکه به ازای مقادیر مختلف تفاوت مقدار دو تحریک

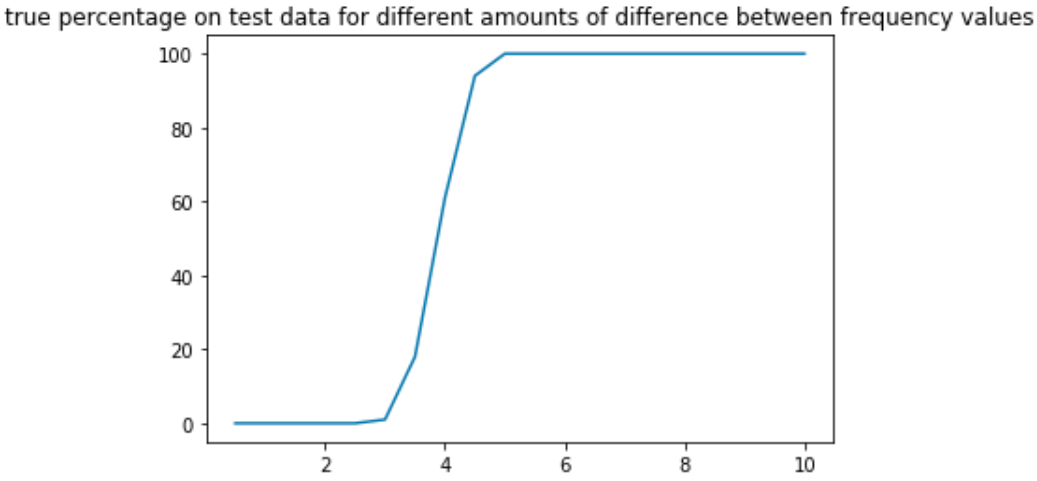


شکل ۱۰: درصد پاسخ صحیح شبکه به دیتای تست برای مقادیر مختلف تفاوت مقدار دو تحریک



شکل ۱۱: نمونه‌ای از ورودی‌های اعمال‌شده به شبکه در تسک حافظه‌ی کاری

فرکانس)، تصمیم‌گیری برای شبکه راحت‌تر شده و درصد پاسخ صحیح بیش‌تر می‌شود.



شکل ۱۲: درصد پاسخ صحیح شبکه به دیتای تست برای مقادیر مختلف تفاوت مقدار دو تحریک

۴ تسک سوم: Eye-Movement Sequence Execution

در این تسک، مدل دنبال کردن حرکت یک نقطه توسط چشم را پیاده‌سازی می‌کنیم. خروجی شبکه باید محل چشم باشد که با دو مختص x و y مدل می‌شود.

۱.۴ تولید دیتاست

توضیحات مربوط به چگونگی تولید دیتاست در مقاله به صورت دقیق داده شده است. دیتاست شامل دو بخش است، یک بخش ۹ ورودی که متناظر با ۹ محل موجود در صفحه است؛ و بخش دیگر ۸ ورودی، که متناظر با یکی از ۸ دنباله‌ی حرکتی موجود است. بنابراین در هر لحظه از زمان مجموعا ۱۷ ورودی به شبکه داده می‌شود. در طول زمان، ۹ ورودی اول ابتدا همگی خاموش هستند، سپس ورودی ۵ که متناظر با شروع اولیه حرکت و محل وسط صفحه است برای مدتی روشن می‌شود؛ در ادامه برای سه بازه‌ی متوالی، در هر بازه سه ورودی روشن هستند که متناظر با محل فعلی نقطه، و دو محل احتمالی بعدی آن هستند. همچنین ۸ ورودی دیگر که تعیین‌کننده‌ی نوع دنباله‌ی اجراشونده هستند (یعنی مشخص می‌کنند کدام‌یک از ۸ دنباله‌ی ممکن در حال اجرا است)، در تمام طول زمان وضعیت ثابتی دارند. ورودی متناظر با دنباله‌ی موجود در تمام طول زمان روشن، و مابقی ورودی‌ها تماماً خاموش هستند.
همچنین خروجی‌های مطلوب نیز مقدار x و y متناظر با محل چشم می‌باشند. شکل ۱۳ نشان‌دهنده‌ی ورودی‌ها و خروجی‌های متناظر با دنباله‌ی شماره ۲ است.

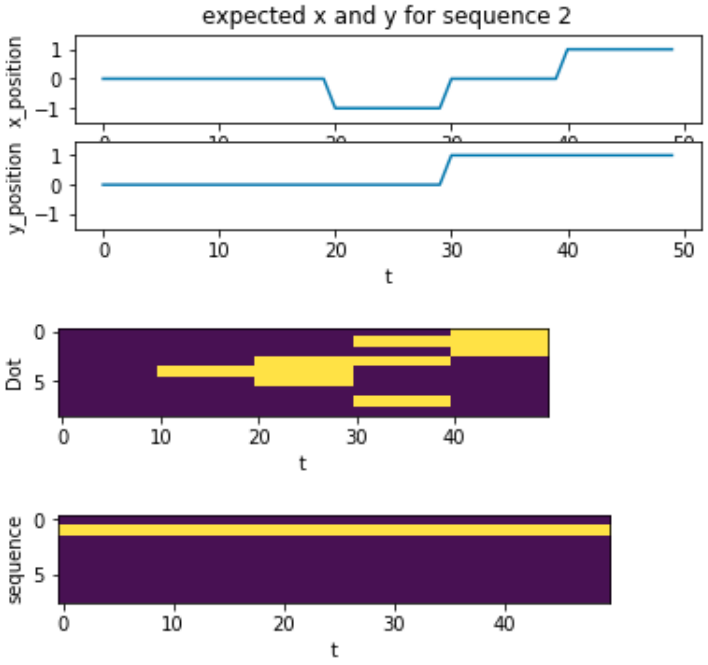
۲.۴ پیاده‌سازی تسک

با اعمال ورودی‌های متناظر به شبکه، آن را آموزش می‌دهیم. دقت کنید که دو ماتریس $9 \times n$ و $8 \times n$ برای ورود به شبکه به هم متصل شده‌اند و یک ماتریس $17 \times n$ را تشکیل داده‌اند. برای بررسی عملکرد شبکه روی داده‌های تست، دو روش مورد استفاده قرار گرفته شده است.

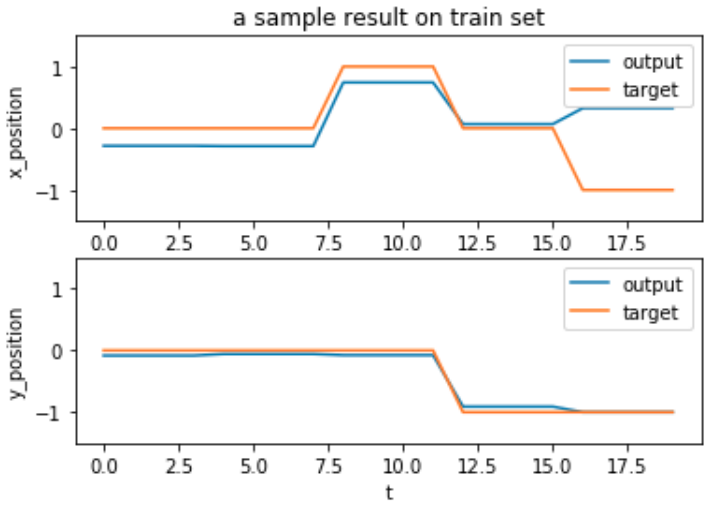
- روش اول:

تمام ۱۷ ورودی موجود در فرایند آموزش به شبکه داده می‌شود. انتظار داریم که خروجی در این حالت، از کیفیت بالایی برخوردار باشد. این وضعیت را به ازای دو حالت بررسی می‌کنیم؛ یک بار تعداد نورون‌های شبکه را کم در نظر می‌گیریم و بار دیگر تعداد نورون‌ها را بیشتر قرار می‌دهیم. در هر حالت، یک نمونه از نتایج شبکه بر روی دیتای آموزش و دیتای تست در این گزارش آورده شده است. شکل‌های ۱۴ تا ۱۷ نشان‌دهنده‌ی این خروجی‌ها هستند.

مشاهده می‌شود که وقتی تعداد نورون‌ها زیاد می‌شود، نتایج قابل قبولی روی هر دو داده‌ی آموزش و تست به دست می‌آید.



شکل ۱۳: یک نمونه ورودی و خروجی مطلوب شبکه (متناظر با دنباله‌ی شماره ۲)

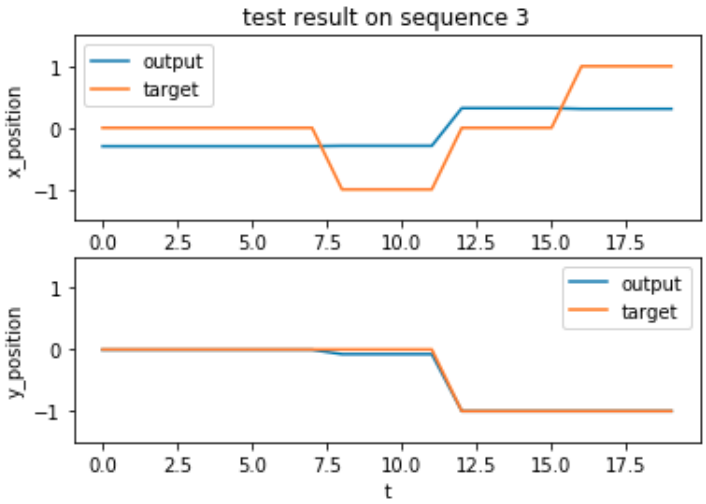


شکل ۱۴: یک نمونه از خروجی‌های شبکه با تعداد نورون‌های کم بر روی داده‌ی آموزش

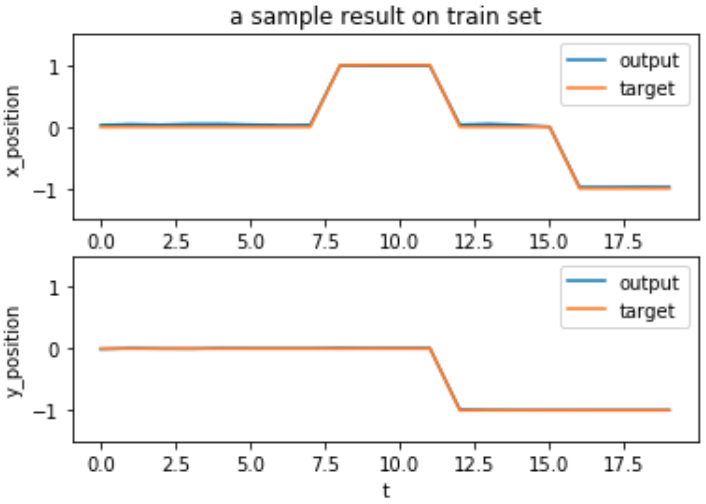
- روش دوم:

در این حالت تنها ورودی‌های مربوط به مکان (ورودی‌های ۱ تا ۹) به شبکه داده می‌شود و دیگر نوع دنباله مشخص نمی‌شود. دو نمونه ورودی از نتایج حاصل بر روی دیتای آموزش و تست (برای شبکه با تعداد نورون‌های کافی) در شکل‌های ۱۸ و ۱۹ قابل مشاهده است.

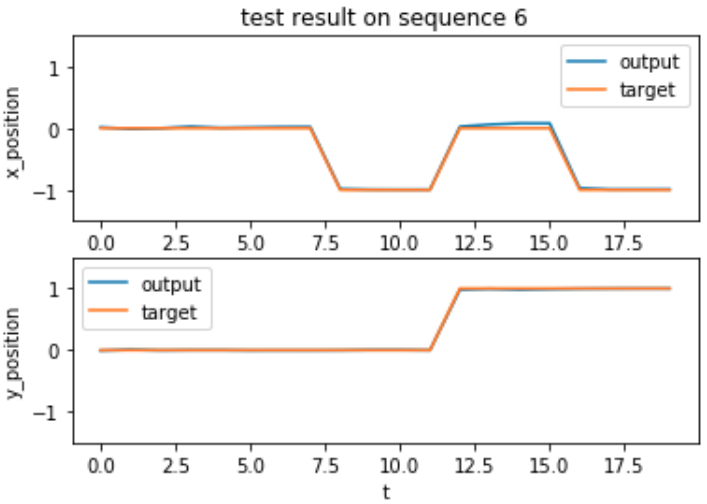
مشاهده می‌شود که در این حالت، مجدداً نتایج مربوط به دیتای آموزش خوب است (که در واقع بخش مربوط به دیتای آموزش با روش اول تفاوتی ندارد)، اما بخش مربوط به دیتای تست به خوبی قبل نیست و شبکه بعضاً در تشخیص بعضی حرکات دچار اشتباه می‌شود و تصمیم نادرست می‌گیرد.



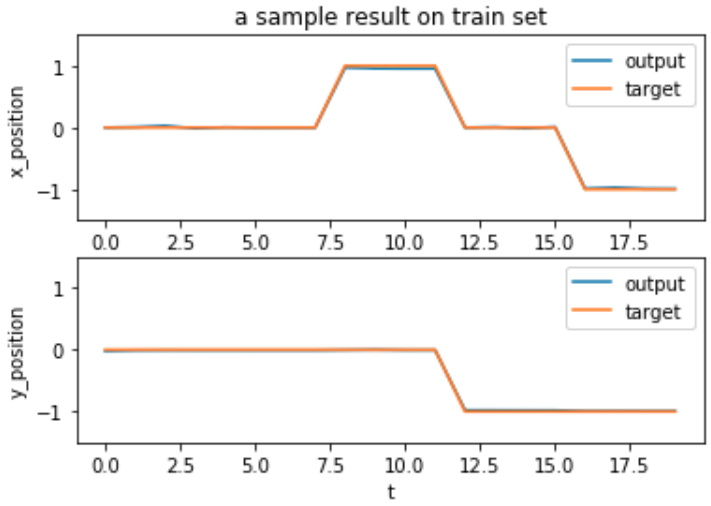
شکل ۱۵: یک نمونه از خروجی‌های شبکه با تعداد نورون‌های کم بر روی داده‌ی تست



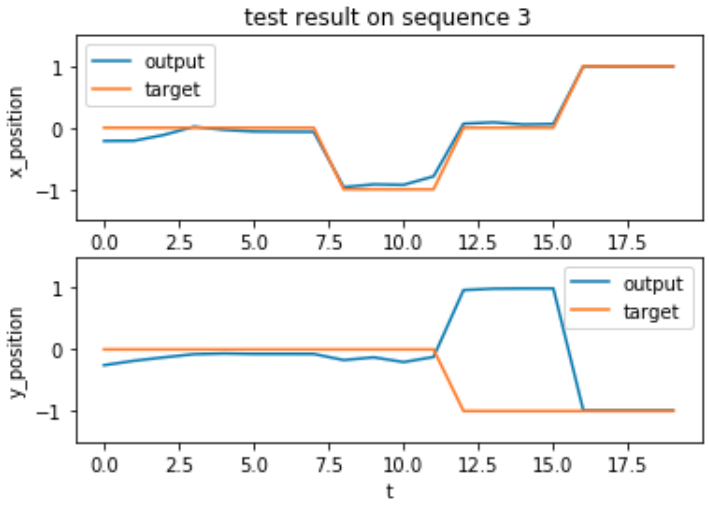
شکل ۱۶: یک نمونه از خروجی‌های شبکه با تعداد نورون‌های زیاد بر روی داده‌ی آموزش



شکل ۱۷: یک نمونه از خروجی‌های شبکه با تعداد نورون‌های زیاد بر روی داده‌ی تست



شکل ۱۸: یک نمونه از خروجی‌های شبکه بر روی داده‌ی آموزش بدون اعمال ورودی مربوط به نوع دنباله



شکل ۱۹: یک نمونه از خروجی‌های شبکه بر روی داده‌ی تست بدون اعمال ورودی مربوط به نوع دنباله