به نام خدا

تمرین ششم درس شبکه های عصبی — توحید عابدینی ۹۹۷۲۲۱۵۳

سوال ۱)

Adaptive Neuro Fuzzy Inference System یا ANFIS امروزه در کانون توجه است. دلیل مورد توجه گرفتن زیاد این سیستم توانایی شبیه سازی سری های زمانی غیرخطی در موضوعات مختلف نظیر شبکه های سنسوری بیسیم، مدلسازی سطح رودخانه و... است. همچنین در کارهای متعدد گزارش شده است که کاربرد های محدودی هم در زمینه پیش بینی میزان تقاضای آب شرب شهری دارد. ANFIS یک شبکه عصبی مصنوعی است که با یک سیستم ادراک فازی تلفیق شده است. در این الگوریتم دوگانه (هیبرید) با استفاده از انتشار به عقب خطا، تخمین کمینه مربع، شبکه عصبی قوانین if-then مناسب و توابع عضویت (MFS) برای سیستم ادراک فازی از روی مجموعه داده های موجود به دست می‌آورد.

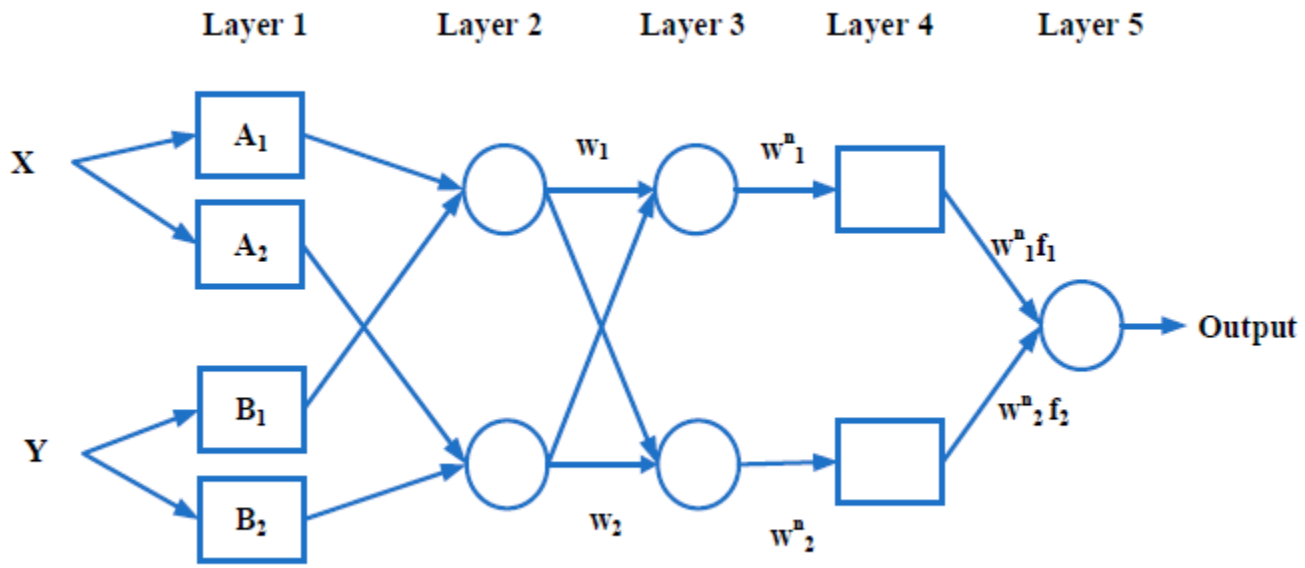
با توجه ذات ترکیبی بودن ANFIS این سیستم قدرت منتج از هر دو مدل اصلی را میتواند داشته باشد که توانایی تخمین توابع غیر خطی است. از نظر محاسباتی سیستم مرتبه اول Takagi-Sugeno فشرده و موثر است پس از این سیستم برای ساخت ANFIS استفاده شده است. همچنین عملکرد ANFIS با انتخاب توابع عضویت مناسب با تعداد مناسب قابل بهبود است.

همچنین برای هر ورودی، ۳، ۵ و ۷ تابع عضویت در نظر گرفته شده است که در فرآیند آموزش و آزمون استفاده میشود. علاوه بر این ۸ نوع تابع عضویت با اسامی زنگی شکل (gbell)، ذوزنقه ای (trap)، منحنی piشکل (pi)، منحنی گاوسی (Gaussian)، اختلاف دو سیگموید (dsig)، مثلثی (tri)، منحنی گاوسی دو سویه (gauss2) و حاصل ضرب دو سیگموید (psig) آزمایش شده اند. ۲۴ سناریو مختلف شبیه سازی برای پیش بینی میزان تقاضای آب شرب شهری به دست آمد که برای هر کدام ۱۰۰۰ ایپاک آموزش انجام یافته است.

سیستم ANFIS ۵ لایه دارد که به شرح زیر ساختار یافته اند:

عضویت – قوانین – نرمال سازی – تابع – خروجی

شکل زیر برای درک بصری بهتر آورده شده است.



ساختار ANFIS

در شکل بالا X و Y بیانگر ورودی هستند. A2-A1 و B1-B2 متغیر های زبانی را نشان میدهند. نورون های مربع شکل نورون های adaptive ما هستند و باید تغییر یابند (آموزش داده شوند) در حالی که نورون های دایره شکل ثابت هستند ونحوه عملکردشان مشخص است.

۱. لایه اول: هر نورون دو نود تطبیقی دارد که از رابطه زیر به دست می آیند.

$$O_{1,i} = \mu A_i(x)$$

$$O_{1,i} = \mu B_i(y)$$

که در آن μ ها توابع عضویت گفته شده هستند.

۲. لایه دوم: ورودی های لایه قبل در هم ضرب شده و وزن آن نورون را ایجاد میکنند.

$$O_{2,i} = W_i = \mu A_i(x)\mu B_i(y),\ i = 1,2$$

W_i حاصل ضرب ورودی های متناظر از لایه قبل است.

۳. لایه سوم: خروجی لایه دو در این لایه نرمال میشود.

$$O_{3,i} = \overline{w}_i = \frac{w_i}{w_1 + w_2},\ i = 1,2$$

۴. لایه چهارم: یک تابع برای این نورون اعمال میشود تا ورودی های نرمال شده لایه دوم را با هم ترکیب کند.

$$O_{4,i} = \overline{w}_i f_i = \overline{w}_i(p_i x + q_i y + r_i)$$

$p_i - q_i - r_i$ پارامتر های این تابع هستند و آموزش داده میشوند.

این لایه در حقیقت لایه ای است که defuzzification را انجام داده و در $\overline{w_i}$ ضرب میکند. اگر برای تسک کلاس بندی از این شبکه استفاده کنیم در این قسمت شبکه یک بعد دیگر به وزن ها اضافه شده و همین لایه پارامتر های بیشتری خواهد داشت. یعنی یک اندیس جدا برای تعداد خروجی های شبکه ایجاد میکنیم. (برای n خروجی تعداد پارامتر ها در این لایه n برابر خواهد شد.)

۵. لایه پنجم: لایه خروجی است که ورودی های لایه قبل با هم جمع میشوند.

$$O_{5,i} = \sum \overline{w}_i f_i = \sum w_i f_i / \sum w_i\ , i = 1,2$$

سوال ۲) در نوت بوک با اسم Q2 تحویل داده شده است. گزارش جزییات پیاده سازی در اینجا آورده میشود.

با استفاده از لینک موجود در فایل تمرین با کتابخانه skfuzzy این سوال پیاده سازی شده است.

برای فازی کردن ورودی ها با استفاده از قوانین داده شده به ترتیب ۷ ترم فازی برای متغیر ورودی دما و ۵ ترم فازی برای متغیر ورودی رطوبت ایجاد کرده اینم. که با استفاده از اطلاعات زیر میتوان معنی عبارات فازی را متوجه شد. ورودی ها با بازه به طول ۱ به مجموعه universe تبدیل شده اند.

V بسیار – C سرد – M متوسط – H بالا (برای ورودی رطوبت) و گرم (برای ورودی دما)

برای تقسیم بازه بندی هر دو ورودی و خروجی از روش automf و مثلثی استفاده کرده ایم که به نوعی متداول ترین شیوه تقسیم است و با واقعیت هم تطابق خوبی دارد. اسم های دلخواه در یک لیست به این متد داده شده است تا طبق خواسته ما نامگذاری ترم های فازی انجام شود.

برای ساخت قوانین هم از متد Rule استفاده میکنیم. که یک نمونه را با هم میبینیم:

```
ctrl.Rule((temperature['C'] | temperature['M']) & humidity['VL'], speed['L'])
```

ورودی اول شرط ماست که با استفاده از عملگر های منطقی | (OR) و & (AND) شروط را ترکیب میکنیم و در ورودی دوم خروجی را مشخص میکنیم.

ترم فازی متغیر مربوطه داخل کروشه با اسم آن داده میشود. در این حالت میگوییم اگر دما سرد یا متوسط باشد و رطوبت بسیار بالا باشد سرعت چرخش پنکه کم شود.

و در نهایت پس از تعریف قوانین، کنترلر را ساخته و سپس ورودی های گفته شده سوال را داده و محاسبات را توسط کنترلر طراحی شده انجام میدهیم و خروجی سیستم را دریافت میکنیم.

سوال ۳) در نوت بوک با اسم Q3 تحویل داده شده است. گزارش جزییات پیاده سازی در اینجا آورده میشود.

ورودی اول ما که همان خروجی اول استیت محیط است مکان ماشین میباشد. این ورودی را به ۶ قسمت با نامهای X1 X2 X3 X4 X5 X6 تقسیم میکنیم. به صورت auto و مثلثی با بازه های مساوی. دلیل اینکه ۶ قسمت انتخاب شد این بود که مکان ابتدایی شروع بازی که حدودا 0.45- است در مرکز تقریبی یکی از این ترم های فازی قرار بگیرد.

به همین ترتیب ورودی دوم که سرعت است را هم به ۵ قسمت با نامهای H – M (medium) – L – VL (Very Low) (High) و VH (Very High) تقسیم میکنیم که نحوه تقسیم همانند ورودی اول به صورت اتو و برابر است.

سپس قوانین محیط را ایجاد میکنیم:

سیاست کلی طبق توضیح سوال این است که ماشین در صورتی که در ابتدای محیط است و از سمت دیگر آمده با قدرت بیشتری به سمت راست فشار داده شود تا به صورت پاندومی مجددا به سمت دیگر رفته تا رفته رفته دامنه حرکت بیشتر شود.

و اگر در نقطه شروع بود (منطقه X3) اگر سرعت منفی است (کم یا خیلی کم) نیروی مثبت خیلی زیاد وارد شود و برعکس.

قوانین با پیروی از سیاست فوق به صورت زیر نوشته شده اند.

```
ctrl.Rule((x['X1'] | x['X2']) , action['VH']),
ctrl.Rule((x['X3']) & (speed['L'] | speed['VL']), action['VL']),
ctrl.Rule((x['X3']) & (speed['H'] | speed['VH']), action['VH']),
ctrl.Rule((x['X4']) & (speed['L'] | speed['VL']), action['VL']),
ctrl.Rule((x['X4']) & (speed['H'] | speed['VH']), action['VH']),
ctrl.Rule((x['X5']), action['VH']),
```

در منطقه X5 هم که انتهای مسیر است با نیروی ماکسیمم سعی در به جلو راندن ماشین داریم تا به هدف برسد.
در منطقه X4 هم مشابه منطق گفته شده بسته به اینکه ماشین از چه سمتی آمده است (با علامت سرعت متوجه میشویم که در ترم های فازی L,VL علامت منفی و مقدار بزرگ بوده و در ترم های H,VH علامت مثبت و مقدار هم بزرگ میباشد.) سعی در وارد کردن نیرو در سمت مخالف داریم (پس از صفر شدن سرعت و شروع به برگشت به محل آمده) تا رفته رفته دامنه حرکت مان بزرگ تر شود.)
با تحلیل نمودار پاداش ها میبینیم که پاداش در نقاط غیر هدف تفاوت بسیار زیادی با پاداش در آن نقطه دارد. و در نقاط غیر هدف به نوعی تناسب با میزان انرژی مصرف شده (نیروی وارد شده) در آن استپ دارد. و میتوان گفت که رابطه معکوس با انرژی مصرف شده در آن گام دارد (به صورت جبری و بدون علامت – یعنی فقط میزان انرژی مهم است).
یعنی در صورتی که به هدف برسیم یک پاداش بزرگ (۱۰۰) به پاداش اولیه (صفر) اضافه میشود و در نهایت انرژی مصرف شده در آن گام از آن کم میشود. اگر هم به هدف نرسیم که فقط انرژی مصرفی از مقدار اولیه (صفر) کم میشود.

سوال ۴) در نوت بوک با اسم Q4 تحویل داده شده است. گزارش جزییات پیاده سازی در اینجا آورده میشود.

در این سوال ابتدا متغیر های محیط یک تغییر کوچک کرده اند و عوض sin و cos زاویه را به رادیان تبدیل میکنیم و از آن استفاده میکنیم.

برای حل سوال از یادگیری تقویتی عمیق استفاده شده است و با استفاده از الگوریتم Deep Deterministic Policy Gradient (DDPG) که دو مدل actor و critic دارد. که هر یکی بر مبنای شبکه عصبی عمیق هستند. در دید سطح بالا این مدل را اینگونه میتوان تعریف کرد که actor اقدام به انجام عمل در محیط میکند و critic به actor میگوید که تا چه حدی عملکردش خوب بوده است و با استفاده از این بازخورد به بهبود خود و بهینه سازی میپردازد. رویکرد آموزش actor بر اساس policy gradient است.

یک بافر هم داریم که با یک سایز مشخص (در اینجا ۵۰۰۰۰) مشاهدات را ذخیره میکند و در صورتی که پر شود با مشاهدات جدید جایگزین میشود و با batch سایز خود نمونه تصادفی از بین ۵۰۰۰۰ تا انتخاب کرده و آموزش و بروزرسانی این دفعه را با استفاده از آنها انجام میدهد. با این کار نمونه های ما وابستگی و correlation کمتری دارند و یادگیری ما بایاس دار نخواد بود و قابلیت تعمیم بهتری خواهد داشت. ورودی مدل actor مشاهدات از محیط خواهد بود و ورودی مدل critic مشاهدات به همراه عمل انتخاب شده توسط actor خواهد بود تا بتواند قضاوت خوبی در رابطه با عمل انتخاب شده داشته باشد. معماری مدل های مذکور هم قابل بررسی و تغییر است و باید بسته به پیچیدگی مساله انتخاب شود تا underfit/overfit نشود.
در شکل زیر شبه کد این الگوریتم را با هم مشاهده میکنیم.

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

که در ابتدا به صورت تصادفی وزن ها را مقدار دهی میکنیم. بافر هم در ابتدا مقدار دهی میشود (با صفر) و به مرور مشاهدات را در خود ذخیره میکند.

سپس به تعدادی که تعیین کرده ایم اپیزود داریم و در هر اپیزود تعداد مشخص (یا نا مشخص و تا زمانی که اپیزود به اتمام برسد با شرط done) قدم برمیداریم و هر بار بافر و مدل ها را آپدیت میکنیم (با روابط موجود در عکس). برای استیبل کردن آموزش و اینکه با استفاده از روابط bellman یادگیری را انجام میدهیم و بروزرسانی میکنیم از دو شبکه target برای هر مدل استفاده شده است. مقادیر پاداش و اعمال ما با استفاده از شبکه های target در فرآیند آموزش محاسبه شده و یک تابع هزینه بین خروجی این شبکه (که در معادلات bellman) جایگذاری شده و خروجی داده است و خروجی مدل متناظر (actor برای اعمال و critic برای نقد کردن عمل) محاسبه شده و بروز رسانی انجام میشود.

با مطالعه شبه کد بالا میتوان به جزییات بیشتر پی برد.

پاداش ها در این محیط به این با رابطه ی زیر به دست می‌آید:

reward: -(theta^2 + 0.1*theta_dt^2 + 0.001*action^2)

که زاویه بین pi و pi- و عمل بین 2- و 2 و theta_dt هم سرعت زاویه ای است که بین 8- و 8 است.

هدف این محیط بالا نگه داشتن پاندول (به صورت ثابت) است که پاداش صفر خواهد داشت (بیشترین پاداش) و در بقیه حالات پاداش منفی داریم و در pi بیشترین فاصله از نقطه تعادل را داریم که بیشینه پاداش منفی میتواند در آنجا رخ دهد.

با بررسی نمودار متوجه میشویم که هر چه به نقطه تعادل نزدیک شویم (با سرعت کمتر در آنجا) پاداش بیشتر خواهد بود.

و اگر در یک اپیزود زیاد در پایین بمانیم پاداش منفی زیادی خواهیم گرفت. با شروع حرکت به سمت نقطه تعادل رفته رفته پاداش بیشتر میشود تا در نهایت و در نقطه تعادل به مقدار بیشینه برسد.

سوال ۵) در نوت بوک با اسم Q5 تحویل داده شده است. گزارش جزییات پیاده سازی در اینجا آورده میشود.

در نوت بوک آپلود شده شبکه عصبی ANFIS برای تمامی دیتاست ها با هر دو نوع رگرسیون و کلاس بندی آموزش داده شده و توسط PSO بهینه سازی شده است.

در ابتدا پارامتر های شبکه که در نورون های adaptive لایه اول و لایه چهارم شبکه هستند به صورت تصادفی وزن دهی شده اند و سپس به بهینه سازی این پارامتر ها پرداخته شده است.

معماری شبکه دقیقا مطابق با مقاله پیاده سازی شده است. که در سوال اول به تفصیل گزارش داده شده است.

تعداد ترم های فازی برای هر ورودی به صورت یکسان و برابر با ۳ ترم در نظر گرفته شده است. که میتوان با هر عدد قابل قبول دیگری شبکه را طراحی کرد.

برای آموزش ابتدا دیتاست را به دو بخش آزمون و آموزش تقسیم کرده (به صورت two-way split و با نسبت ۲۰ – ۸۰ که قابل تغییر است) و با استفاده دیتاست آموزش و الگوریتم PSO به آموزش شبکه پرداخته و سپس معیار های هزینه (MSE برای رگرسیون و Cross Entropy برای کلاس بندی) برای هر دو دیتاست آموزش و آزمون برای بررسی کیفیت آموزش شبکه و پارامتر های به دست آمده گزارش میشود.

از کتابخانه pyswarms برای بهینه سازی PSO استفاده شده است. که پارامتر های آن به صورت زیر است:

که w نشانگر اینرسی وزن ها و c1 و c2 هم ثابت های شتاب هستند که چگونگی حرکت ذرات را تعیین میکنند. پارامتر iter نشانگر تعداد تکرار و particle هم به معنای تعداد ذرات در الگوریتم است.

```
# Set-up hyperparameters
options = {'c1': 0.5, 'c2': 0.3, 'w':0.9}
```

```
cost, pos = run_PSO_on_network(network, n_particles=100, iters=10)
```

به دلیل کند بودن اجرا و محدودیت سخت افزاری روی سیستم شخصی تعداد تکرار ها روی عدد ۱۰ و تعداد particle ها روی عدد ۱۰۰ تنظیم شده است. میتوان برای بررسی امکان آموزش بهتر روی تعداد تکرار بالا تر هم الگوریتم را اجرا کرد.

هزینه (عملکرد) الگوریتم و (معیار دقت برای تسک رگرسیون) برای حالت ابتدایی یا پارامتر های تصادفی (بین ۰ و ۱) هم گزارش شده است تا نشان دهد الگوریتم بهینه سازی تا چه حدی کارساز بوده است.

در شکل زیر نتیجه اجرای الگوریتم بر روی دیتاست stock یه یک تسک رگرسیون است آورده شده است.

```
cost, pos = run_PSO_on_network(network, n_particles=100, iters=10)
```

```
2022-01-12 05:00:32,265 - pyswarms.single.global_best - INFO - Optimize for 10 iters with {'c1': 0.5, 'c2': 0.3, 'w': 0.9}
pyswarms.single.global_best: 100%|
|10/10, best_cost=0.00703
2022-01-12 05:03:10,657 - pyswarms.single.global_best - INFO - Optimization finished | best cost: 0.007034484250286584, bes
t pos: [-0.18523599  0.39525363  0.36044681  0.64968409  0.08335652 -0.30749874
  0.04620737  0.12230104  0.65568481  0.17458391 -0.12110144  0.18570054
  0.37792509  0.07633825  0.21825258 -0.34777294  0.66558111  1.20971793
  0.43717829  0.55777829  1.10947873  0.11336077  0.37063121 -0.1081405
  0.21858808  0.0251866   0.59694481  0.32052406  1.25376305  0.20198665
  0.47344497  0.44563173  0.36493497  0.45099722  0.02210589  0.08023556
  0.44114211 -0.03350152  0.42665905  0.785043    0.56248504  0.33389951
  0.83495996  0.14155413  0.49893157  0.64485829  1.04824707 -0.00953864
 -0.22497878  0.26148151  0.87003354]
```

```
reult_printer(pos, init_train, init_test)
```

```
Train loss: 0.007034484250286584, Init train loss: 1.3406646332954915
Test loss: 0.010499653358208613, Init test loss: 1.4021133707936864
```

میتوان با مشاهده عکس بالا متوجه شد که آموزش تا چه حدی تاثیر گذار بوده است و اینکه آیا مدل قابلیت تعمیم بالا دارد یا خیر.

میتوان با بررسی حالت های مختلف نسبت تقسیم دیتاست و همچنین پارامتر های الگوریتم PSO و استفاده از توابع تعیین عضویت

فازی دیگر (که در اینجا به دلیل راحتی و محدودیت نداشتن از gbell استفاده شده است) اجرا های مختلف از شبکه و الگوریتم گرفت و نتایج را مشاهده کرد.

به دلیل نحوه عملکرد الگوریتم PSO تمامی وزن ها توسط تابع parameters_wrapper، concat شده اند و در تابع suitable_form_of_network_as_function عمل بازگردانی به شکل اصلی و اجرای شبکه روی دیتای داده شده انجام شده است.

با استفاده از تابع predict هم میتوان خروجی شبکه (پیش بینی ها و معیار های مربوط به تسک) را برای هر دیتافریم دلخواه محاسبه کرد.

منبع:

https://www.matec-conferences.org/articles/matecconf/pdf/2016/26/matecconf_mmme2016_02019.pdf

منابع دیگر در ابتدای نوت بوک آورده شده اند. که بیشتر منابع کد زنی هستند.