تمرین سری چهارم (سری دوم بخش دکتر خلج) درس زیست شناسی مصنوعی – گزارش تمارین عملی

امین سعیدی

۴۰۰۲۱۱۵۷۹

سوال اول

توضح کلی کد:

کد زده شده برای این سوال حاوی توابع زیر است که هرکدام توضیح داده خواهند شد. این کد با نام Net_motifs.py همراه با فایل های دیگر ارسال شده است.

۱. Data_preparation
این تابع با گرفتن نام فایل ورودی، آن را میخواند و به عنوان خروجی یک لیست برمیگرداند که هر عنصر آن یک سطر فایل است.

۲. Prepare_network
در این تابع لیست خوانش فایل گرفته میشود و سپس به عنوان خروجی یک دیکشنری داده میشود که در آن کلید ها اپرن ها و مقادیر دیکشنری ارتباطات و نحوه این ارتباطات است.

۳. Generating_random_network
این تابع تعداد یال ها و راس های یک شبکه را میگیرد و یک شبکه تصادفی با آن تعداد یال و راس تولید میکند و به شکل یک لیست خروجی میدهد، مطابق با تابع یک، خروجی این تابع برای گرفتن شکل دیکشنری باید به تابع prepare_network داده شود

۴. NOA_network_generator
این تابع، یک شبکه کامل را میگیرد و از آن ارتباطات autoregulation را حذف میکند.

۵. Output_generator
در این تابع ابتدا ۱۰۰۰ شبکه تصادفی ساخته میشود و تعداد الگوهای هریک شمرده میشود، تا با میانگین و واریانس گیری و انجام تست اماری، مشخص شود که آیا یک الگو موتیف هست یا خیر. خروجی این تابع یک دیکشنریست که کلید های آن الگوهای مختلف و مقادیر ان به ترتیب میانگین و واریانس تعداد ان الگو در شبکه های تصادفی و همچنین مقادیر پی استخراج شده از تست های آماری هستند.

۶. Finding_NAR
این تابع با گرفتن یک شبکه کامل به صورت دیکشنری، تعداد NAR های آن شبکه را میشمرد.

۷. Finding_PAR
این تابع با گرفتن یک شبکه کامل به صورت دیکشنری، تعداد PAR های آن شبکه را میشمرد.

۸. Finding_DPFL
این تابع با گرفتن یک شبکه بدون autoregulation به صورت دیکشنری، تعداد DPFL های آن شبکه را میشمرد.

۹. Finding_DNFL
این تابع با گرفتن یک شبکه بدون autoregulation به صورت دیکشنری، تعداد DNFL های آن شبکه را میشمرد.

۱۰. Finding_FAN_outs

این تابع با گرفتن یک شبکه بدون autoregulation به صورت دیکشنری، تعداد FAN_outs های آن شبکه را میشمرد.

۱۱. Finding_Cascades

این تابع با گرفتن یک شبکه بدون autoregulation به صورت دیکشنری، تعداد Cascades های آن شبکه را میشمرد.

۱۲. Finding_FFL

این تابع با گرفتن یک شبکه بدون autoregulation به صورت دیکشنری، تعداد FFL های آن شبکه را میشمرد.

در انتهای این کد نیز، با استفاده از توابع بالا، ابتدا شبکه ها از روی فایل های ورودی ساخته میشوند و سپس تعداد الگوهای گفته شده در شبکه اصلی شمرده میشوند و سپس دیکشنری نهایی تولید میشود تا از ان نتیجه گیری شود که کدام الگو ها موتیف هستند.

الف:

Pattern Name	Count
Negative Autoregulation	45
Positive Autoregulation	17
Double-positive Feedback loop	0
Double-negative Feedback loop	0
Fan-outs	113
Cascades	202
FFL	15

ب)

Pattern Name	Count	Random-networks-mean	Random-networks-var	t-test(p-value)
Negative Autoregulation	45	0.895	0.8548	**0**
Positive Autoregulation	17	0.896	0.83	**0**
Double-positive Feedback loop	0	1.252	2.12	**~0**
Double-negative Feedback loop	0	1.253	2.023	**~0**
Fan-outs	113	97.504	39.42	**0**
Cascades	202	348.785	762.63	**0**
FFL	15	0.463	0.465	**0**

از آنجایی که مقادیر-پی برای هریک از تست های یک طرفه بالا بسیار کوچک است، لذا میتوان نتیجه گرفت که آن میانگینی که از شبکه های رندم بدست آمده (یعنی ستون سوم جدول بالا) میتواند تخمین خوبی باشد از این که اگر یک شبکه به صورت تصادفی ایجاد شود، چه تعداد از هر یک الگوها خواهد داشت. بنابراین میتوان دید که سه الگوی ***NAR,PAR,FFL*** حداقل چندین برابر مقدار تصادفی هستند و لذا میتوان نتیجه گرفت که این سه الگو بنابر تکامل به این میزان در ***e-coli*** موجود هستند و لذا میتوان آن ها را موتیف در نظر گرفت.

سوال دوم

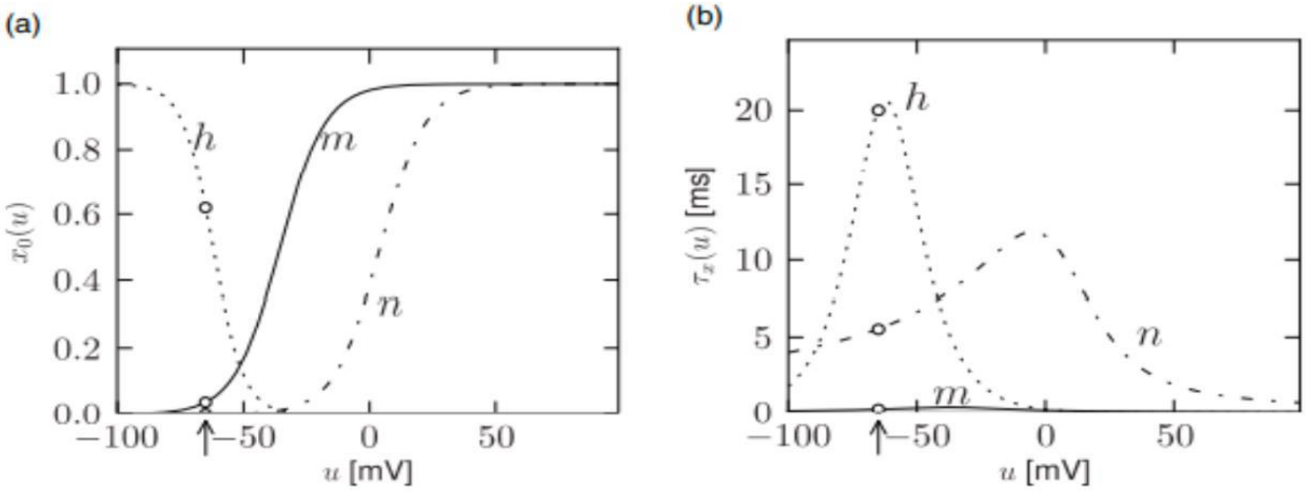
الف:

$$C_m \frac{du}{dt} = -\bar{g}_{Na}(+m)^3 h\left(u - V_{(Na)^+}\right) - \bar{g}_K(+n)^4\left(u - V_{(K)^+}\right) - \bar{g}_L(u - V_L) + I_{ext}$$

$$\frac{dm}{dt} = -\frac{1}{\tau_m}\left(m - m_0(u)\right)$$

$$\frac{dn}{dt} = -\frac{1}{\tau_n}\left(n - n_0(u)\right)$$

$$\frac{dh}{dt} = -\frac{1}{\tau_h}\left(h - h_0(u)\right)$$



فرض اول)

با توجه به نمودار a بالا میتوان مشاهده کرد که مقیاس زمانی دینامیک متغیر گیت m بسیار سریع‌تر از متغیرهای n و h است. مقیاس زمانی m در مقایسه با ثابت زمانی غشا $\tau = \mathrm{C/gL}$ یک غشای غیرفعال سریع است، که مشخصه تکامل ولتاژ u در زمانی که همه کانال‌ها بسته هستند، است. مقیاس زمانی نسبتا سریع m نشان می دهد که ما ممکن است m را به عنوان یک متغیر آنی در نظر بگیریم. بنابراین متغیر m در معادله جریان یونی مدل هوچکین-هاکسلی را می توان با مقدار حالت پایدار آن که در زیر آمده است جایگزین کرد:

$$(I): m(t) \to m_0[u(t)]$$

این همان چیزی است که آن را quasi steady state می نامیم که به دلیل separation of time scales بین متغیرهای سریع و آهسته امکان پذیر است.

فرض دوم)

با توجه به نمودار a بالا می بینیم که ثابت های زمانی $\tau_n(u)$ و $\tau_h(u)$ دینامیک مشابهی بر روی ولتاژ u دارند. علاوه بر این، نمودارهای$n_0(u)$ و $1 - h_0(u)$ که در نمودار b مشاهده میشوند نیز مشابه هستند. این نشان می دهد که ما میتوانیم دو متغیر n و $(1 - h)$ را با یک متغیر w تقریب بزنیم. برای اینکه روال کمی کلی تر باشد، از تقریب خطی زیر استفاده می کنیم:

$$(b - h) \approx an \rightarrow where\ a\&b\ are\ some\ constant$$

$$\rightarrow w = b - h = an$$

$$\rightarrow (II): h = b - w$$

$$\rightarrow (III): n = \frac{w}{n}$$

$$\rightarrow (I)\&(II)\&(III) \rightarrow C\frac{du}{dt} = -\bar{g}_{Na}\left(m_0(u)\right)^3(b - w)\left(u - V_{(Na)^+}\right) - \bar{g}_K\left(\frac{w}{a}\right)^4\left(u - V_{(K)^+}\right) - \bar{g}_L(u - V_L) + I_{ext}$$

$$\rightarrow \frac{du}{dt} = \frac{1}{\tau}[F(u.w) + RI] \rightarrow where: R = g_L^{-1}\ \&\ \tau = RC$$

$$\rightarrow according\ to\ (I)\&(II)\&(III)$$

$$\rightarrow\ The\ m\ equation\ has\ disappeared\ since\ m\ is\ treated\ as\ instantaneous.\ Also:$$

$$\rightarrow\ Instead\ of\ the\ two\ equations\ for\ n\ and\ h, we\ are\ left\ with\ a\ single\ effective\ equation:$$

$$\rightarrow according\ to\ (I)\&(II)\&(III)$$

$$\rightarrow \frac{dw}{dt} = \frac{1}{\tau_w}G(u.w) \rightarrow where\ \tau_w is\ a\ parameter\ and\ G\ a\ function\ that\ interpolates\ between \frac{dn}{dt} and \frac{dh}{dt}$$

$$\rightarrow So: two\ above\ highlighted\ equations\ define\ a\ general\ two - dimensional\ neuron\ model$$

$$\rightarrow w = -\tan\alpha * n_0(u_{rest}) - z_1 \sin\alpha$$

$$\rightarrow where: \tan\alpha = \frac{\frac{dh_0}{du}}{\frac{dn_0}{du}}$$

$$\rightarrow where: \frac{dz_1}{dt} = -\cos\alpha * \frac{z_1\cos\alpha + n_0(u_{rest}) - n_0(u)}{\tau_n(u)} - \sin\alpha * \frac{z_1\sin\alpha + h_0(u_{rest}) - h_0(u)}{\tau_h(u)}$$

ب:

در ابتدا لازم است تا ذکر کنم که برای انجام این تمرین ابتدا در اینترنت گشتی زدم و به این لینک برخورد کردم که کامل سوال خواسته شده از ما را پیاده سازی کرده بود. با توجه به کمبود وقت و فشردگی دروس در انتهای ترم تصمیم گرفتم تا برای این تمرین مستقیما از این منبع استفاده کنم. اما خب از آنجایی که هدف اصلی تمرین یادگیری بیشتر مباحث است، سعی کردم تا کامل متوجه منطق و روند کد شوم تا یادگیری اتفاق بیافتد. در ادامه توضیحات مربوط به کد ارسال شده آورده شده است. **به همراه این گزارش نیز نمودار های تعداد کمی شبیه سازی مرتبط با ولتاژ خروجی به ازای ورودی های مختلف آورده شده است.**

توضیحات کد:

کد از دو بخش اصلی تشکیل شده است:

1. **کلاس HHModel**

این کلاس هسته کد این تمرین محسوب میشود که مقادیر داده شده در سوال را برای هریک از Na و K و Kleak مقدار دهی میکند. این کلاس حاوی یک متغیر Vm است که ولتاژ خروجی در هر حالت شی این کلاس را در خود نگه میدارد. نحوه استفاده از یک شی این کلاس به این صورت است که در یک حلقه و با پیشبرد زمان حالات گیت ها آپدیت شده و به کمک حالات گیت ها و رابطه هوچکین-هاکسلی مقادیر ولتاژ خروجی مقدار دهی شود. این کلاس حاوی کلاس و توابع زیر است:

a. کلاس gate

کلاس gate سینتیک و حالت باز کانال را مدیریت می کند و به نوعی برای شبیه سازی سه گیت m و n و h کاربرد دارد که دارای توابع زیر است تا مقادیر گیت ها با زمان آپدیت شوند.

i. تابع Update

ii. تابع setInfiniteState

b. تابع UpdateGateTimeConstants

این تابع با توجه به روابط آورده شده در صورت تمرین مقادیر ثابت زمانی $\alpha_m - \alpha_h - \alpha_h$ و $\beta_m - \beta_h - \beta_h$ را با گرفتن ولتاژ هر استپ آپدیت میکند.

c. تابع UpdateCellVoltage

این تابع با توجه به رابطه اصلی هوچکین-هاکسلی ولتاژ را با محاسبه اجزای رابطه در هر استپ محاسبه میکند.

d. تابع UpdateGateStates

این تابع حالت گیت ها را با استفاده از تابع update کلاس gate آپدیت میکند.

e. تابع Iterate

این تابع به نوعی هسته این کلاس محسوب میشود، چرا که در این تابع به ترتیب توابع b و c و d فراخوانی میشوند تا به ترتیب مقادیر ثابت زمانی، ولتاژ در زمان و همچنین حالت گیت ها آپدیت شوند.

f. تابع inputPulse

این تابع برای تولید انواع ورودی نوشته شده است. این تابع با گرفتن زمان و یک stim مقدار دهی اولیه شده، با توجه به mode و استفاده از کتابخانه scipy یک ورودی تولید میکند. ورودی های در نظر گرفته شده SquarePulse و chrip و unit_impulse و cosine هستند. برای نمونه نیز دو دامنه ۱ و ۲۰ برای این ورودی ها در نظر گرفته شده است.

۲. قسمت main

این قسمت، قسمت اجرایی کد است که در آن ابتدا کلاس HHModel و تعداد نقاطی که برای آن ها میخواهیم ولتاژ خروجی را محسابه کنیم مقدار دهی اولیه میشوند و سپس نام مدل ورودی به عنوان mode مقدار دهی اولیه میشود و پس از آن نیز متغیرهای time و موج ورودی مقدار دهی میشوند. سپس در یک حلقه به تعداد تکرار تعداد نقاط، ولتاژ محاسبه میشود و در یک لیست ریخته میشود تا از آن استفاده شود و نمودارها رسم گردد. در ادامه این قسمت نیز در یک حلقه به ازای mode های مختلف که همان حالت های مختلف ورودیست و به ازای یک دامنه مشخص برای ورودی ها، نمودار های شبیه سازی مربوط به موج ورودی برحسب زمان و ولتاژ بر حسب زمان آورده شده است.

بررسی:

میتوان مشاهده نمود که هربار با تغییرات ناگهانی stim ولتاژ نیز دارای نوسان میشود، همچنین مقادیر بالای دامنه در stim میتواند در ولتاژ ایجاد نوسانات شدید تر کند و همچنین میزان ماکسیمم ولتاژ را نیز تحت تاثیر قرار میدهد.