دانشگاه تهران
دانشکده‌ی مهندسی برق و کامپیوتر

پروژه‌ی پنجم هوش مصنوعی

شبکه‌های عصبی

مهلت تحویل: جمعه شب، ۱۳ دی

طراحان:

احمد پوری‌حسینی

محمدرضا طیرانیان

۱. مقدمه

در این پروژه شما قرار است با پیاده‌سازی بخش‌های پایه‌ای یک شبکه‌ی عصبی، شناخت عمیق‌تری را نسبت به آن‌ها کسب کنید. از آنجایی که پیاده‌سازی صفر تا صد یک شبکه‌ی عصبی جا را برای ابهامات و تفاوت‌های زیادی باز می‌گذاشت و عادلانه بودن روند ارزیابی را نیز دشوار می‌کرد، یک اسکلت کلی از پیاده‌سازی یک شبکه‌ی عصبی در اختیار شما قرار گرفته، که شما باید بخش‌هایی از آن را کامل کنید. پس از کامل کردن این پیاده‌سازی باید با استفاده از API موجود چند مسئله‌ی بیان شده در قالب چندین مجموعه‌داده را حل کنید، و پس از اطمینان از عملکرد درست شبکه، چند مفهوم را روی شبکه پیاده‌سازی کرده و بیازمایید.

۲. کتابخانه‌ها و نکاتی در مورد اجرای برنامه:

برای انجام این تمرین، شما مجاز به استفاده از کتابخانه‌های numpy، matplotlib، و pandas هستید. با توجه به اینکه قرار است در این تمرین خودتان شبکه‌ی عصبی را پیاده‌سازی بکنید، شما مجاز به استفاده از هیچ کتابخانه‌ی دیگری، از جمله کتابخانه‌های یادگیری‌ماشین مثل scikit-learn، PyTorch، Tensorflow یا Keras نیستید. همچنین توجه کنید که برای اجرای کدهای آماده‌شده برای شما، باید از نسخه‌ی پایتون ۳ استفاده بکنید. (اگر ورژن پایتون شما متفاوت است،از virtualenv ها کمک بگیرید!) همچنین توجه کنید که بخش‌های پایانی این صورت پروژه نیاز به اجراهایی خواهند داشت که نسبت به چیزی که احتمالا به آن عادت دارید طولانی مدت تر خواهد بود. هر اجرا احتمالا حدود ۳-۴ دقیقه به طول بینجامد و نیاز خواهید داشت چندین بار آن‌ها را تکرار کنید.

۳. پیاده‌سازی شبکه‌ی عصبی:

در فایل neural_net.py بدنه‌ی اصلی یک شبکه‌ی عصبی آورده شده است. سه کلاس Input، PerformanceElem، و Neuron هر سه پیاده‌سازی‌های ناقصی از توابع زیر را دارند که باید توسط شما کامل شوند:

```
def output(self)
def dOutdX(self, elem)
```

تابع **output(self)**:

این تابع، خروجیِ هر کدام از المان‌های شبکه‌ی عصبی را تولید می‌کند. در این تابع باید از activation function سیگموید (لاجیستیک) استفاده کنید.

تابع **dOutdX(self, elem)**:

این تابع مشتق جزئی خروجی را نسبت به المان وزنی که به عنوان ورودی داده شده محاسبه می‌کند. از این مقدار برای بروزرسانی وزن‌های شبکه استفاده خواهد شد. البته در این پیاده‌سازی، به جای مفهوم loss از مفهوم performance استفاده شده که همان عکس loss است. یعنی هرچه performance بالاتر باشد بهتر است. در نتیجه فرمول بروزرسانی وزن‌های شبکه به شکل زیر خواهد بود:

```
wi' = wi + rate * dP / dwi
```

که در آن P همان مقدار Performance است.

توجه کنید که المان ورودی در این تابع، همواره یک وزن خواهد بود. شما باید فکر کنید که چطور می‌توان این تابع را با استفاده از فراخوانی‌های بازگشتی توابع dOutdX و output روی ورودی های شبکه یا سایر وزن‌ها به دست آورد. برای این پیاده‌سازی شما باید از قانون زنجیره‌ای[1] در مشتق گیری استفاده کنید.

برای مثال برای یک Performance Element با نام P، پیاده‌سازی تابع dOutdX می‌تواند به صورت زیر باشد: (در اینجا o خروجی نورونی است که که مستقیما به P متصل شده)

```
dP / d(w) = dP / do * do / dw = (d - o) * o.dOutdX(w)
```

در مورد Neuron ها باید به دقت فکر کنید که شرط پایان تابع بازگشتی چه خواهد بود. توجه کنید که این مدل پیاده‌سازی متفاوت از مدل متداول محاسبه‌ی دلتاهای کوچک به ازای هر نورون است که در کلاس تدریس شد.

در فایل documentation.pdf، کلاس‌های موجود داخل کد به شکل مفصل تر توضیح داده شده اند. توصیه می‌شود قبل از ادامه دادن، آن را مطالعه کنید. همچنین پیشنهاد می‌کنیم برای درک بهتر فرآیند back-propagation به توابع train و test در فایل neural_network.py نگاه بیندازید.

۴. تست کردن شبکه:

بعد از اتمام پیاده‌سازی توابع، می‌توانید با استفاده از اسکریپت پایتون neural_net_tester.py کار خود را تست کنید. با اجرای دستور زیر مطمئن خواهید بود که برنامه‌ی شما برای کاربردهای ساده‌ای مثل AND و OR کار خواهد کرد:

```
Python neural_net_tester.py simple
```

[1] chain rule

برای دریافت نمره‌ی این بخش حتما باید شبکه‌ی شما به دقت ۱۰۰ درصد روی این دو مجموعه داده برسد.
برای ساختن شبکه‌های عصبی بخش های بعدی، می‌توانید از کد داخل تابع `make_neural_net_basic` که شبکه‌ی عصبی این بخش را پیاده‌سازی کرده الگو بگیرید.

قواعد نام‌گذاری

هنگام نام‌گذاری المان‌هایی که طراحی می‌کنید حتما از قواعد زیر پیروی کنید:

۱) ورودی‌ها به صورت 'i' + input_number باشند. Input_number باید از یک شروع شود. مثلا: i1, i2. از i0 برای ورودی‌های با مقدار ثابت (که ضریب آن‌ها نقش bias را در شبکه دارند) از مقدار 1- استفاده کنید.

۲) برای وزن‌ها به صورت 'w' + from_id + to_id نام‌گذاری کنید. مثلا برای وزنی که که از ورودی شماره یک به نورون A می‌رود w1A مناسب است. یا برای وزنی که از نورون A به B می‌رود کافی است wAB را به عنوان نام انتخاب کنید.

۳) برای اسم نورون‌ها باید یک حرف از حروف الفبا اختصاص دهید. به این صورت که نزدیک‌ترین نورون به ورودی A نام می‌گیرد و نورون‌های دورتر B و ... . اگر دو نورون فاصله‌ی یکسانی با ورودی دارند به هر صورت که مایل هستید ترتیبی را قائل شوید.

۵. Finite Difference

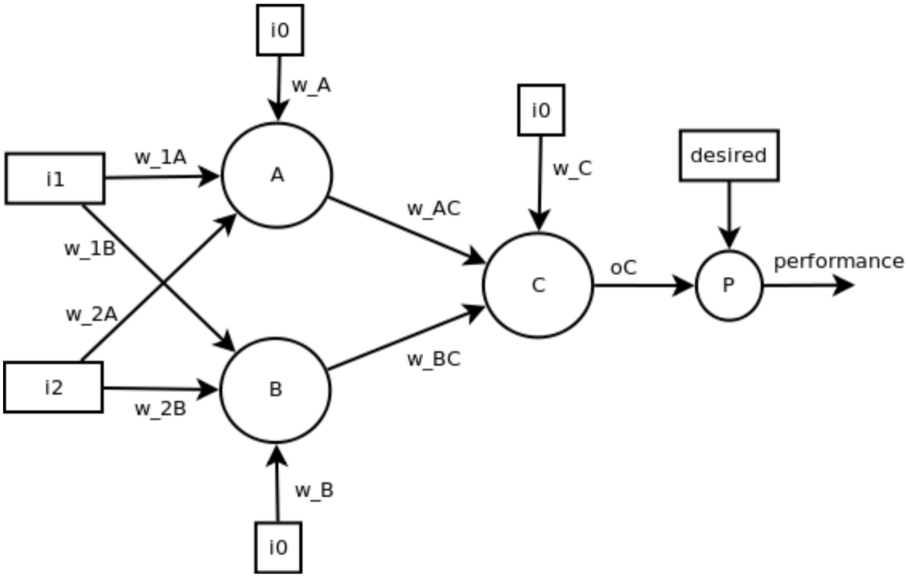
رسیدن به دقت ۱۰۰ درصد روی مسائل ساده‌ی OR و AND لزوما به معنای درست بودن کامل پیاده‌سازی نیست. برای اینکه از درستی پیاده‌سازی که در بالا داشته اید تا حد خوبی مطمئن بشوید و همچنین اگر که پیاده‌سازی درست نیست تکنیکی برای debug کردن آن داشته باشید، باید در این بخش از متد finite difference برای تخمین زدن مشتق وزن‌ها استفاده کرده، و بعد آن را با مقدار حاصل از توابعی که نوشته‌اید مقایسه کنید. فرمول مورد استفاده در این متد برای تقریب زدن مشتق به شکل زیر است:

$$\acute{f}(x) = \frac{f(x+\epsilon) - f(x)}{\varepsilon}$$

که در آن مقدار ε باید برابر مقدار بسیار کوچکی باشد (مثلا ۱۰ به توان منفی ۸). برای کاربرد ما، x هر کدام از وزن‌های شبکه خواهد بود، و تابع f، تابعی است که خروجی PerformanceElem را نمایش می‌دهد. شما باید در این بخش تابعی بنویسید که با گرفتن یک شبکه، روی وزن‌های آن iterate بکند، و به ازای هر وزن، مقدار مشتق PerformanceElem را نسبت به آن وزن یک بار از متد تخمین بالا، و یکبار از طریق تابع dOutdX محاسبه کرده و برابری (تقریبی) آن‌ها را چک بکند و در نهایت اگر همه برابر بودند مقدار True برگرداند. در نوشتن این تابع حواستان به خالی کردن cache شبکه بین دو محاسبه باشد.

۶. پیاده‌سازی شبکه عصبی دولایه‌ای

حال از API ای که کامل کرده‌اید استفاده کنید و یک شبکه‌ی عصبی دولایه بسازید که مانند شکل زیر باشد:



این کار را داخل تابع make_neural_net_two_layer() در داخل فایل neural_net.py انجام بدهید. شبکه عصبی شما باید قادر باشد دیتاست‌های کمی سخت تر مثل NOT EQUAL (XOR) یا EQUAL را طبقه‌بندی کند.

برای وزن‌های این شبکه مقادیر اولیه‌ای به صورت رندم در نظر بگیرید. دقت کنید که برای تکرارپذیر بودن تست‌ها مقدار seed را قبل از هر چیزی تعیین کنید و بعد با استفاده از تابع random_weight مقدار وزن‌های مورد نیاز را تولید کنید:

```
seed_random()
wt = random_weight()
...use wt...
wt2 = random_weight()
...use wt2...
```

برای تست این شبکه دستور زیر را استفاده کنید:

```
python neural_net_tester.py two_layer
```

شبکه‌ی شما باید برای دیتاست neural_net_data.harder_data_set به دقت ۱۰۰ درصد برسد.

۷. کشیدن ناحیه‌ی تصمیم‌گیری:

در این بخش شما باید تابعی بنویسید که با دریافت یک شبکه‌ی عصبی، و یک محدوده از صفحه در قالب یک مربع، ناحیه‌ی تصمیم گیری شبکه را در آن قسمت از صفحه رسم کند. به منظور این کار کافی است که در آن محدوده از صفحه، نقاط زیادی را به شکل یک grid ریزدانه انتخاب کنید و به ازای هر نقطه معین کنید که آیا خروجی شبکه کمتر از ۰.۵ است یا خیر، و اگر جواب مثبت بود آن نقطه‌را به نحوی روی صفحه نمایش بدهید. امضای این تابع باید به شکل زیر باشد:

```
def plot_decision_boundary(network, xmin, xmax, ymin, ymax)
```

۸. بیش‌برازش[۲] و regularization:

گاهی از اوقات، اگر ظرفیت شبکه‌ی عصبی برای یادگیری خیلی بیش‌تر از پیچیدگی داده‌ای باشد که باید یاد بگیرد، ممکن است بیش‌برازش رخ بدهد. بیش‌برازش وقتی رخ می‌دهد که شبکه‌ی عصبی شروع به حفظ کردن تک تک جزییات بی‌اهمیت موجود در داده بکند، و در نتیجه قابلیت تعمیم‌دهی[۳] خود را از دست بدهد. برای آشنایی بیش‌تر با این مفهوم توصیه می‌شود این مطلب را مطالعه کنید. همچنین دقیقه‌ی ۱۲ الی ۲۰ این ویدئو نیز توضیحات کلی خوبی راجع به بیش‌برازش دارد.

حال می‌خواهیم یک شبکه‌ی نسبتا بزرگ‌را روی مجموعه‌داده‌ی two moon آموزش بدهیم. شما باید یک شبکه‌ی عصبی با ۲ ورودی در لایه‌ی اول، ۴۰ نورون در لایه‌ی دوم و ۱ نورون خروجی در لایه‌ی سوم طراحی کنید و در تابع make_neural_net_two_moons آن را پیاده‌سازی کنید. سپس آن را در دفعات جداگانه، به تعداد ۱۰۰، ۵۰۰ و ۱۰۰۰ iteration آموزش دهید. سپس با استفاده از تابع test موجود در فایل neural_net.py، میزان دقت شبکه‌را روی داده‌های آموزش و تست به صورت جداگانه اندازه گیری کرده و در یک جدول گزارش دهید. همچنین ناحیه‌ی تصمیم‌گیری را نیز در این ۳ مورد با استفاده از تابع بخش قبل بکشید. علت عددهایی را که مشاهده می‌کنید با توجه به توضیحات داده شده و ناحیه‌ی تصمیمی که می‌بینید توضیح بدهید.

یکی از راه‌حل های جلوگیری از پدیده‌ی بیش‌برازش، استفاده از regularization است. راه‌های مختلفی برای این کار وجود دارد، و یکی از پرکاربرترین آن ها استفاده از l2-norm است که در لینکی که داده شد توضیح آن آمده بود.
شما باید یک کلاس جدید به اسم RegularizedPerformanceElem تعریف کنید که از کلاس PerformanceElem به ارث ببرد، و توابع output و dOutdX آن را به نحو مناسب تغییر دهد تا در خروجی شبکه l2-norm لحاظ بشود. البته در موقع پیاده‌سازی باید توجه داشته باشید که در پیاده‌سازی ما performance وجود دارد، نه loss. در نتیجه عبارت l2-norm را باید با علامت منفی در پیاده‌سازی خود لحاظ کنید. شما در این پیاده‌سازی مجاز هستید سایر توابع یا متغیرهای به ارث رسیده از کلاس PerformanceElem را نیز در صورت نیاز تغییر بدهید.

پس از پیاده‌سازی این کلاس، و با استفاده از آن، آموزش‌های خواسته شده‌ی بالا را مجددا تکرار کنید و تمام موارد خواسته شده را در این حالت نیز محاسبه کرده و در گزارش کار خود بیاورید. آیا مشکل بیش‌برازش حل شده است؟ اگر می‌بینید که یادگیری شبکه دچار مشکل شده، سعی کنید با ضریب لاندا بازی کنید تا مقدار مناسب آن را پیدا بکنید. همچنین توضیح مختصری بدهید که چرا این کار توانسته/نتوانسته جلوی بیش‌برازش را بگیرد.

۹. گزارش کار

گزارش کار در همه پروژه ها باید کامل باشد و تصحیح از روی آن انجام می شود. نمودارها و تحلیل‌هایی که در هر مرحله به دست می‌آید، در آن ضمیمه شده باشد. فرمت نهایی گزارش کار باید pdf یا HTML باشد.

[2] overfitting
[3] generalization

۱۰. مراجع:

اکثر سوال‌های پیاده‌سازی این پروژه با تغییر اندکی از این منبع برداشت شده‌اند:

https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/assignments/MIT6_034F10_lab5.pdf

۱۱. نکات پایانی

- توجه کنید که پیاده‌سازی هر وزن و هر نورون به شکل یک کلاس جدا، باعث می‌شود برنامه‌ی شما به شدت کند باشد و در نتیجه نخواهید توانست کاربردهای به‌روز و عملی شبکه‌های عصبی را با استفاده از این کد پیاده‌سازی کنید. این پیاده‌سازی خاص صرفا به دلیل سادگی فهم و بار آموزشی آن برای شما در نظر گرفته شده است.
- کدهای کامل شده را به همراه تمامی فایل‌های مورد نیاز برای اجرای آن در داخل پوشه‌ی Codes قرار داده، و به همراه گزارش کار خود در قالب یک فایل zip آپلود کنید. مهم است که کدهایی که به دست ما می‌رسد در همان شکل آپلود شده قابل اجرا باشند وگرنه نمره‌ای به شما تعلق نخواهد گرفت.
- برای ما مهم است که حاصل کار خودتان را به ما تحویل دهید. در صورت تقلب برای بار اول به هر دو طرف نمرهی ۱۰۰- تعلق میگیرد و بار دوم معرفی به دانشگاه و ثبت نمره ۰.۲۵ به عنوان تقلب انجام می شود.
- تاخیر به ازای روز اول و دوم هر کدام ۱۰ درصد و روز سوم به بعد هر روز ۱۵درصد خواهد بود. برای مثال سه روز تاخیر ۳۵درصد از نمره دریافتی شما را کم میکند.
- در صورتی که سوالی داشتید در فروم درس مطرح کنید که دیگران هم از جواب آن استفاده کنند.

موفق باشید!