"بسمه تعالی"

گزارش پروژه چهارم هوش مصنوعی – زهرا لطیفی – ۹۹۲۳۰۶۹

گزارش ۱: در هنگام تعریف مدل از دو نوع لایه در شبکه عصبی استفاده کردیم Linear و همینطور ReLU . درباره هر کدام از این لایه‌ها تحقیق کنید و توضیح دهید که چرا باید خروجی هر لایه Linear به یک لایه ReLU داده شود. همچنین نقش هر کدام از این لایه‌ها را بیان کنید.

لایه Linear نوعی لایه در شبکه عصبی است که یک تبدیل خطی روی ورودی انجام می‌دهد، مانند ضرب ماتریس در بردار. یک لایه Linear می تواند وزن‌ها و بایاس‌هایی که به بهترین وجه ورودی را به خروجی مورد نظر نگاشت می‌کنند را یاد بگیرد. با این حال، یک لایه Linear نمی‌تواند روابط غیرخطی که ممکن است در داده‌ها وجود داشته باشد را مدل کند. بنابراین، یک لایه Linear اغلب با یک activation function غیرخطی مانند ReLU دنبال می‌شود.

ReLU مخفف rectified linear unit یک تابع فعال سازی ساده اما موثر است که ورودی را در صورت مثبت بودن برابر خزوجی قرار می‌دهد و در غیر این صورت خروجی برابر صفر قرار می‌دهد. ReLU دارای چندین مزیت نسبت به سایر توابع فعال‌سازی مانند sigmoid یا tanh است. ReLU به جهت محاسبات سریع است زیرا شامل هیچ عملیات ریاضیاتی سنگینی مانند نمایی یا تقسیم ندارد. ReLU همچنین مشکل صفر شدن گرادیان توابع را کاهش می‌دهد. این مشکل چه زمانی رخ می‌دهد؟ زمانی که گرادیان توابع فعال‌سازی بسیار کوچک شده و روند یادگیری را کند می‌کند. ReLU دارای گرادیان ثابت ۱ برای ورودی‌های مثبت است، به این معنی که می‌تواند وزن‌ها را سریع‌تر و موثرتر به روز کند. ReLU همچنین شبکه را مقداری sparse می‌کند زیرا برخی از خروجی‌ها را صفر خواهد کرد، که این فرآیند می‌تواند خطر Overfitting را کاهش دهد و generalization را بهبود بخشد.

پس نقش ReLU غیرخطی سازی شبکه است که به آن اجازه می‌دهد تا توابع پیچیده تر را بیاموزد. نقش لایه‌های Linear انجام تبدیل‌های خطی بر روی ورودی است که می‌تواند ابعاد و تنوع featureها را افزایش دهد. با ترکیب لایه‌های Linear و لایه‌های ReLU، یک شبکه عصبی می‌تواند هر تابع پیوسته را با تعداد لایه‌های پنهان کافی تقریب بزند.

گزارش ۲: در فرایند آموزش مدل از Optimizer Adam استفاده کردیم. ابتدا توضیح دهید که چرا نیاز به استفاده از Optimizer داریم و سپس درباره Adam تحقیق کنید و نحوه عملکرد آن را توضیح دهید.

بهینه‌سازی روشی است که پارامترهای یک شبکه عصبی مانند وزن‌ها و بایاس‌ها را با هدف به حداقل رساندن تابع `loss` تنظیم می‌کند. `Optimizer` ها برای آموزش مدل‌های یادگیری عمیق ضروری هستند، زیرا به سرعت بخشی به فرآیند یادگیری و یافتن راه حل بهینه کمک می‌کنند.

`Adam` یک `Optimizer` محبوب است که مزایای دو `Optimizer` دیگر را ترکیب می‌کند: `Momentum` و `RMSprop`. `Momentum` با استفاده از `Moving Average` گیری از گرادیان‌ها، به تسریع فرآیند یادگیری کمک

می‌کند. RMSprop به تطبیق Learning Rate برای هر پارامتر، با استفاده از Moving Average گیری از مجذور گرادیان‌ها کمک می‌کند که باعث می‌شود Learning Rate خیلی بزرگ یا خیلی کوچک نشود.

Adam با محاسبه Moving Average گرادیان نمایی و مجذور گرادیان و استفاده از آنها برای به روز رسانی پارامترها کار می‌کند. Adam همچنین دو هایپرپارامتر β1 و β2 را معرفی می‌کند که نرخ کاهش این Moving Average را کنترل می‌کنند. مقادیر پیش فرض برای β1 و β2 به ترتیب ۰.۹ و ۰.۹۹۹ است. Adam همچنین از مکانیزم اصلاح بایاس برای جلوگیری از مقداردهی اولیه Moving Average با صفر استفاده می‌کند. Adam برای آموزش شبکه‌های عصبی عمیق موثر و کارآمد شناخته شده است، زیرا می‌تواند گرادیان‌های sparse، داده‌های نویزی و اهداف متحرک را مدیریت کند.

گزارش ۳: در فرایند آموزش مدل از Loss MSE استفاده کردیم. ابتدا توضیح دهید که چرا نیاز به Loss داریم سپس درباره MSEتحقیق کنید و نحوه عملکرد آن را توضیح دهید.

Loss معیاری است که نشان می‌دهد یک شبکه عصبی چقدر می‌تواند خروجی desired از یک ورودی مشخص را پیش بینی کند. Loss همچنین Error یا Cost نامیده می‌شود و برای ارزیابی عملکرد یک مدل استفاده می‌شود. در واقع Loss با مقایسه خروجی desired با خروجی واقعی با استفاده از Loss Function محاسبه می‌شود. Loss Function یک فرمول ریاضی است که این اختلاف را کمّی می‌کند. انواع مختلفی از توابع Loss وجود دارد و هر کدام مزایا و معایب خاص خود را دارند. انتخاب Loss Function به نوع مسئله، نوع خروجی و الگوریتم بهینه‌سازی بستگی دارد.

MSE مخفف میانگین مربعات خطا است و یکی از متداول ترین توابع برای مسائل Regression است که در آن خروجی یک مقدار پیوسته است. MSE میانگین اختلاف مجذور بین خروجی پیش بینی شده و خروجی واقعی را محاسبه می‌کند. فرمول MSE را می‌توان به صورت زیر نوشت:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

که n تعداد نمونه‌ها، $\hat{y}i$ خروجی پیش‌بینی‌شده، و yi خروجی واقعی است.

MSE برای خطاهای بزرگ بیشتر از خطاهای کوچک penalty تعیین می‌کند، چرا که خطا قبل از میانگین گیری مجذور می‌شود. این به این معنی است که MSE به نقاط پرت حساس است. MSE همچنین دارای یک تفسیر هندسی واضح است، زیرا معادل فاصله اقلیدسی بین خروجی پیش بینی شده و خروجی واقعی است. محاسبه و مشتق‌گیری از MSE آسان است، که آن را برای الگوریتم های بهینه سازی مبتنی بر گرادیان مناسب می‌کند.

با این حال، MSE دارای معایبی نیز می‌باشد:

- MSE را می‌توان تحت تأثیر مقیاس خروجی دانست، زیرا مقادیر بزرگتر منجر به خطاهای بزرگتر می شود. بنابراین، مهم است که قبل از استفاده از MSE، خروجی را نرمالایز یا استاندارد کنیم.

- MSE می‌تواند منجر به واریانس بالای گرادیان شود، زیرا خطا می‌تواند بسته به ورودی بسیار متفاوت باشد. پس درواقع می‌تواند فرآیند یادگیری را ناپایدار و کُند کند.

- MSE می‌تواند باعث شود که مدل به سمت میانگین بایاس پیدا کند، زیرا توزیع خروجی را در نظر نمی‌گیرد که می‌تواند منجر به Underfitting و Generalization ضعیف شود.

گزارش ۴: در فرایند آموزش مدل عددی تصادفی تولید می‌کنیم و با مقایسه این عدد با Epsilon تصمیم می‌گیریم که Action بعدی چه باشد. توضیح دهید که چرا به چنین فرایندی نیاز داریم و انجام این کار چه تاثیری روی آموزش ما خواهد داشت. همچین بیان کنید که با کم یا زیاد کردن Epsilon چه اتفاقی خواهد افتاد.

این فرآیند epsilon-greedy exploration نامیده می‌شود و یک تکنیک رایج برای ایجاد تعادل بین exploration و exploitation در یادگیری تقویتی است. exploration به معنای آزمایش اقدامات مختلف برای کشف پیامدهای آنها است، در حالی که exploitation به معنای انتخاب بهترین اقدام بر اساس دانش فعلی است. epsilon-greedy exploration با انتخاب یک عمل تصادفی با احتمال اپسیلون و انتخاب بهترین عمل با احتمال 1-epsilon کار می‌کند.

دلیل اینکه ما به چنین فرآیندی نیاز داریم، جلوگیری از گیرکردن مدل در یک local optimum یا یک سیاست غیربهینه است. اگر مدل فقط از دانش فعلی خود بهره برداری کند، ممکن است اقدامات بهتری را که هنوز امتحان نکرده است از دست بدهد. از سوی دیگر، اگر مدل فقط به صورت تصادفی کاوش کند، ممکن است زمان و منابع را برای اقداماتی که به وضوح ضعیف هستند تلف کند. epsilon-greedy exploration به مدل اجازه می‌دهد تا بین این دو هدف تعادل برقرار کند و از موفقیت‌ها و شکست‌هایش درس بگیرد.

تأثیر epsilon-greedy exploration بر یادگیری مدل به مقدار اپسیلون بستگی دارد. اگر اپسیلون خیلی زیاد باشد، مدل بیش از حد exploration انجام می‌دهد و دانش آموخته شده خود را نادیده می‌گیرد. این می‌تواند منجر به یادگیری کند و ناکارآمد شود، زیرا ممکن است مدل به سیاست بهینه همگرا نشود. اگر اپسیلون خیلی کم باشد، مدل بیش از حد exploitation می‌کند و از exploration اجتناب می‌کند. این می‌تواند منجر به یادگیری زودرس و غیربهینه شود، زیرا ممکن است مدل اقدامات بهتری را کشف نکند یا با تغییرات محیط سازگار نشود. بنابراین، انتخاب یک مقدار مناسب از اپسیلون برای عملکرد مدل بسیار مهم است.

یکی از روش‌های رایج برای انتخاب اپسیلون استفاده از decay schedule است، جایی که اپسیلون با مقدار زیادی شروع می‌شود و به تدریج در طول زمان کاهش می‌یابد. این به مدل این امکان را می‌دهد که در ابتدا، زمانی که دانش کمی دارد، بیشتر exploration کند، و زمانی که تسلط بیشتری دارد، بیشتر exploitation کند. روش دیگر برای انتخاب اپسیلون استفاده از dynamic schedule است که در آن اپسیلون به وضعیت محیط یا مدل بستگی دارد. به عنوان مثال، زمانی که مدل غیرقطعی است یا محیط نویزی است، اپسیلون می‌تواند بالاتر باشد و زمانی که مدل قطعی است یا محیط پایدار است، پایین‌تر باشد.

گزارش ۵: در فرایند آموزش مدل توضیح دهید که Q-Targets چه مفهومی دارد. سپس روش محاسبه آن را بیان کنید و شرح دهید که چرا از این فرمول استفاده می‌کنیم.

Q_targets همان Q-value هدفی است که ما می‌خواهیم مدل ما آن را یاد بگیرد. با اضافه کردن یک reward فوری و Q-value حالت بعدیِ تخفیف دار، ضرب در ضریبی که نشان می‌دهد اپیسود تمام شده است یا خیر، محاسبه می‌شود.

فرمول Q_targets از معادله بلمن گرفته شده است که بیان می‌کند که Q-value بهینه برای یک جفت حالت-عمل برابر است با پاداش مورد انتظار به اضافه Q-value بهینه جفت حالت-عمل بعدی. از نظر ریاضی، این را می‌توان به صورت زیر نوشت:

$$Q^*(s,a) = E_{S'}[r + \gamma \max_{a'} Q^*(s', a') | s, a]$$

که در آن Q*(s, a)، Q-value بهینه است، r پاداش، gamma ضریب تخفیف، و Es' انتظار از حالت بعدی است. با این حال، در عمل، ما مقدار Q بهینه را نمی‌دانیم، بنابراین از تقریبی آن استفاده می‌کنیم که با Q(s, a; theta) نشان داده می‌شود، که در آن theta پارامترهای مدل ما است. (مثلا به عنوان وزن یک شبکه عصبی) ما همچنین از یک شبکه هدف جداگانه استفاده می کنیم که با Q(s, a; theta^-) نشان داده می‌شود، که یک کپی از مدل ما است که کمتر به روز می‌شود. این به تثبیت فرآیند یادگیری و جلوگیری از برآورد بیش از حد مقادیر Q کمک می‌کند.

بنابراین، فرمول Q_targets تقریبی به صورت زیر می‌شود:

$$Q_{\text{targets}} = r + \gamma \max_{a'} Q(s', a'; \theta^-) \cdot (1 - d)$$

که در آن d یک نشانگر باینری است که اگر اپیسود تمام شده باشد ۱ و در غیر این صورت ۰ است. ما از این فرمول برای آموزش مدل خود با به حداقل رساندن میانگین مربعات خطا (MSE) بین Q_targets و Q-values پیش بینی شده توسط مدل استفاده می‌کنیم. که معادل همان gradient descent در تابع Loss زیر است:

$$L(\theta) = E_{s,a,r,s',d}[(Q_{\text{targets}} - Q(s,a;\theta))^2]$$

با انجام این کار، ما انتظار داریم که مدل ما یاد بگیرد که تابع Q-value بهینه را تقریب کند و در نتیجه بتواند بهترین عملکرد را در هر حالت انتخاب کند.