شبکه های عصبی

بخش اول)

قسمت اول: ابتدا کتابخانه های لازم را import می کنیم، سپس توابع را تعریف می کنیم.

تابع generate_random_points به تعداد نقطه هایی که به عنوان پارامتر می گیرد، نقطه تصادفی روی تابعی که می خواهیم تقریبش زنیم تولید می کند. از این نقاط به عنوان داده های آموزشی استفاده خواهیم کرد.

تابع create_model بر اساس تعداد لایه ها و تعداد نورون های هر لایه و هم چنین توابع فعالیت لایه های میانی (relu) و لایه آخر (همان تابع همانی است) مدل را طراحی می کند. Train_model هم با استفاده از بهینه ساز adam و معیار خطای mse (میانگین مربعات خطا) شبکه را آموزش می دهد.

Evaluate_model ابتدا در یک بازه شروع و پایان به تعداد num_points نقطه آزمایشی تولید کرده و مقدار هر نقطه را در تابع اصلی و تابعی که شبکه پیشبینی کرده مقایسه می کند. مقداری تلرانس برای اختلاف این دو مقدار در نظر گرفته ایم. در آخر تابع اولیه و تابع تقریب زده شده را در کنار هم رسم کرده و دقت را بر می گردانیم.

برای چهار تابع مدل را ارزیابی می کنیم :

1) Sin(x)

به دلیل اینکه این تابع، متناوب و نسبتا پیچیده است، دامنه آن را گسترده در نظر گرفته و تعداد نقاط زیادی برای آن در نظر می گیریم، افزایش تعداد نورونها و لایه ها تا حدی می تواند مدل بهتری تخمین بزند اما از یک جایی به بعد زائد بوده و ممکن است حتی باعث overfitting بشود. تابع فعالیت relu هم عموما برای لایه های میانی استفاده می شود، استفاده از توابع فعالیت دیگر تاثیر چندانی روی دقت ندارد، تعداد چرخه ها هم با آزمون و خطا به دست آمده، کاهش آن ممکن است باعث کاهش دقت بشود و افزایش آن هم تاثیر چندانی ندارد، هر چند برای توابع مختلف باید مقایسه و ارزیابی بشود. نسبت تقسیم داده ها به دو قسمت آموزش و ارزیاب هم معیار موثری است.

معمولا این نسبت را 0.2 در نظر می گیرند.

در تست مشاهده می کنیم که با وجود اینکه در بازه آموزش مدل عملکرد بهتری داشته است، در بازه ای گسترده تر از دامنه اولیه نتوانسته خوب عمل کند و مقادیر دیگری پیش بینی کرده است. این نشان می دهد که برای این طور توابع فقط می توان انتظار این را داشت که در دامنه ای که مدل آموزش داده شده است بتوان نتایج خوبی گرفت.

2) Log(x)

این تابع چون خیلی پیچیده نیست تعداد نقاط کمتری برای آن در نظر می گیریم و حتی مشاهده می کنیم در دامنه ای بسیار گسترده هم (1.5 برابر بازه آموزش) مدل بسیار خوب عمل کرده است.

3) $X^3 + 3x^2 + 1$

این تابع هم تقریبا در بازه آموزش عملکرد خوبی داشته است ولی به دلیل صعود زیادی که در دامنه های گسترده تر

دارد، ممکن است در این دامنه ها، مدل خوب عمل نکند.

4) $X^4 - 5x^2 + 4$

به دلیل تغییر رفتار زیاد، نقاط زیادی را برای آموزش در نظر گرفته ایم(چون تابع برد گسترده ای دارد، تغییر رفتار آن زیاد مشهود نیست). مشاهده می شود با وجود اینکه تلرانس بالایی برای تست در نظر گرفته ایم، به دلیل شیب بسیار زیاد تابع در دامنه گسترده تر از دامنه آموزش، مدل خوب نمی تواند تابع را در این نقاط پیش بینی کند؛ ولی در سایر نقاط پیش بینی خوبی دارد.

قسمت دوم: ابتدا کتاب خانه ها را import کرده و توابع را تعریف می کنیم.

تابع extract_points تصویر را باینری می کند.

تابع generate_random_points برخی از داده ها را به صورت تصادفی به عنوان داده آموزشی می گیرد، البته اینجا چون تعداد داده ها کم است، از همه داده ها استفاده می کنیم.

تابع drop_noisy_points سعی می کند داده های نویزی را حذف کند، بدین صورت که اگر مولفه y نقطه ای، اختلاف زیادی با مولفه y پیش از خود داشت، حذف می شود. (این اختلاف نسبت به آستانه داده شده سنجیده می شود و اگر آستانه کمتر باشد، سخت گیری بیشتر است اما ممکن است برخی از داده های اصلی هم حذف شوند، به خصوص اگر تابع دارای نقاط پرش ناگهانی و ناپیوستگی های بزرگ باشد.

روش کار بدین صورت است که از تابع عکس گرفته و سپس تصویر تابع را خوانده و آن را باینری می کنیم. سپس مولفه های x و y نقاطی از تصویر را که مقدار صفردارند (نقاط سیاه تصویر) را در یک آرایه نامپای ذخیره می کنیم. قدم بعد این است که شکل به دست آمده را تبدیل به تابع می کنیم یعنی از x های مشترک فقط یکی را نگه می داریم.

تابع create_model بر اساس ابرپارامترها مدل را طراحی می کند. تابع train_model هم مدل را آموزش می دهد. چون تقریب این تابع کار پیچیده ای است، نرخ یادگیری را متغیر قرار می دهیم، بر اساس این فرمول محاسبه می شود:

```
def decayed_learning_rate(step):
    return initial_learning_rate * decay_rate ^ (step / decay_steps)
```

checkpoint هم بهترین مدل را بر اساس ارزیابی ضرر ذخیره می کند.

Evaluate_model هم بر اساس حد آستانه، مدل را ارزیابی می کند.

عملیات آموزش و ارزیابی را انجام می دهیم و می بینیم که در 1000 چرخه مدل خوب آموزش دیده است.

در بخش دوم کمی نویز به داده ها اضافه می کنیم (بدین صورت که مولفه y برخی داده ها را به صورت تصادفی عوض می کنیم) و سپس داده های نویزی شده را حذف نویز می کنیم. مشاهده می کنیم که تابع رفع نویز در برخی نقاط خوب عمل نکرده و نقاطی را از دیتاست اصلی را حذف کرده است. سپس این مدل را با دیتاست حذف نویز شده و دیتاست اولیه مقایسه می کنیم و نمایش می دهیم. دقت این مدل نسبت به مدل اولیه اندکی کمتر است.

بخش دوم)

ابتدا کتابخانه های لازم را import می کنیم. سپس دیتاست را لود می کنیم. دیتاست cifar10 شامل 60000 عکس از 10 کلاس مختلف هست. برخی از عکس های این دیتاست را با کلاسشان مشاهده می کنیم. کلاسها به شرح زیرند.

```
['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

مدل را تعریف می کنیم و داده ها و لیبلشان را (پس از one_hot encoding) به مدل می دهیم. مدل را طوری تعریف می کنیم بهترین نتیجه را بر اساس ضرر ذخیره کند.

پس از آموزش مدل، نمودار های دقت و ضرر را رسم می کنیم. مشاهده می کنیم که پس از چرخه حدودا 100 مدل دچار Overfitting شده است. بهترین درصد بدست آمده هم 55 درصد است که در چرخه 271 بدست آمده است، البته معیار ما بهترین Loss است که حدود 1.4 است و داده ها را بر اساس آن ارزیابی می کنیم. در شبکه های پرسپترون چندلایه ای هم که برای بخش بندی این دیتاست آموزش دیده اند، دقت بین 50 و 60 است.

در آخر برخی داده ها را با لیبل پیش بینی شده و هم چنین لیبل درستشان مشاهده می کنیم.