گزارش کار

پروژه 5 فاز 2 هوش مصنوعی

طنین زراعتی 810197627

نام پروژ: بررسي برخي از مسائل شبكه عصبي به كمك TensorFlow

هدف: تشخیص نژاد افراد با توجه به تصویرشان. 5 نژاد سفید، سیاه، اسیایی،هندی،دیگر داریم.

شرح کلی پروژه:

هر تصویر ابتدا مسطح شده و به صورت بردار به عنوان ورودی به شبکه عصبی داده می شود. هر درایه این بردار (معادل با یک پیکسل تصویر) یک ویژگی برای آن تصویر محسوب می شود. شبکه قرار است بر اساس این ویژگی ها و با ساختن ترکیبات غیرخطی از آن ها، وزن اتصالات بین لایه هایش را طوری تنظیم کند که خروجی آن ضمن داشتن کمترین خطا نژاد افراد را به درستی تشخیص دهد.

داده ها:

داده ها دارای 4 ویژگی: سن، جنیست، نژاد، تاریخ است که هریک چند دسته دارند که در اینجا برای ما نژاد مهم است.

داده ها test و train را با نسبت 20 به 80 جدا میکنیم.

بخش های پروژه:

بخش اول – بررسی و پیش پردازش داده ها:

- از کل داده ها 23705 برای کلاس نژاد سفید 10078 و برای نژاد سیاه 4526 و برای اسیایی3434 هندی برابر 3975 و نژاد های دیگر برابر با 1692 است.

- داده ها را با نسبت 20 برای تست و 80 برای یادگیری جدا میکنیم.

- داده ها را با دو روش بدون نرمال و با نرمال فیت میکنیم.برای حالت بدون نرمال مشاهده میکنیم که خروجی loss , accuracy , F1, precision برابر با nan است و این به دلیل overflow است چون مقدار داده های ما بزرگ است و در relu این مقدار بعد از مدتی چند برابر میشود و همین موجب overflow و بزرگ شدن بیش از حد داده ها شود.به همین دلیل داده ها استاندارد میکنیم (1/255)

بخش دوم – طراحی شبکه:

برای epoch اخر:

```
Epoch 10/10
593/593 [==============================] - 53s
89ms/step - loss: 0.8344 - accuracy: 0.7012 - Recall:
0.6122 - Precision: 0.7786 - F1: 0.6836 - val_loss:
0.8274 - val_accuracy: 0.6986 - val_Recall: 0.6253 -
val_Precision: 0.7717 - val_F1: 0.6898
```

Accuracy برابر با تعداد تشخیص درست .

Accuracy = TP+TN/TP+FP+FN+TN

Precision برابر با دقت نسبت مشاهدات مثبت پیش بینی شده صحیح به کل مشاهدات مثبت پیش بینی شده است.

Precision = TP/TP+FP

Recall برابراست با نسبت تعداد پیش بینی های درست و مثبت به کل پیش بینی های کلاس واقعی.

Recall = TP/TP+FN

F1 برابر است با میانگین وزن دار recall و precision است.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

اطلاعات بدست امده برای بخش دوم و فقط فیت کردن با اطلاعات داده شده برابر است با:

Loss = 0.8274055123329163

Accuarcy = 0.6985868215560913

Recall = 0.6242030262947083

Precision = 0.7702367901802063

F1 = 0.6878976225852966

دقت مدل در ادامه بهتر میشود.

بخش سوم – طبقه بندی داده ها:

تغییراتی که امتحان شده عبارتند از:

افزایش و کاهش تعداد یونیت لایه اول

افزایش و کاهش تعداد بونیت های لایه دوم

افزایش و کاهش لایه ها

تغییر توابع فعال سازی

بهترین نتیجه حاصل شده برای تعداد یونیت ها برابر بودن تعداد یونیت های دو لایه مخفی است. با افزایش بیش از حد اختلاف آنها کاهش کمی در دقت را مشاهده میکنیم (این کاهش خیلی به چشم نمی آید.) برای 1024 یا 100 بودن تعداد یونیت ها بهترین بود.

با افزایش بیش از حد تعداد لایه ها overfitting رخ میدهد که از اختلاف مقادیر recall با precision و accuracy قابل مشاهده است که البته F1 که میانگین وزن دار آنهاست افزایش چندانی نیافته.

Tanhh: قبل از نرمال و استاندارد کردن داده ها تابع فعال ساز tanh مقادیر nan بدست آمده در مدل relu را نداشت چراکه به نوعی درون خود این لیمیت گذاری را انجام میدهد ولی این روش خوبی نیست چرا که میزان زیادی از داده ها را برابر هم قرار میدهد (مقادیر بزرگ و بینهایت را برابر 1 میگذارد) که در کلاس بندی به اشتباه عمل میکند.

ولی برای بعد از استاندارد شدن بهتر از حالت بالا عمل میکند ولی به دلیل شیبی که دارد(درنمودار) relu بهتر است که تا حد زیادی همگرایی نزول شیب تصادفی را در مقایسه با sigmoid / tanh تسریع می کند.

Relu در برابر sigmoid از نظر سرعت خیلی بهتر عمل میکند. Derivative ان نیز سریع تر است که این نکته بسیار مهم است چراکه در شبکه عصبی با تعداد زیادی داده و لایه زمان زیادی حالت عادی میگیرد و استفاده از sigmoid این زمان را بیشتر هم میکند.

Leaky Relu بهتر از Relu عمل میکند چراکه دیگر مشکل "dying ReLU" را ندارد که در آن تابع relu برای تمام مقادیر منفی صفر درنظر میگیرد.

قسمت اول:تاثیر optimizer

1. مومنتوم، جهشی است که به شبکه اعمال می شود در صورتی که در هنگام یادگیری به اکسترمم موضعی رسیده باشیم و مدل بیشتر آپدیت نشود. استفاده از آن در حالتی که امکان گیر افتادن مدل در چنین حالتی وجود دارد مفید است زیرا این حالت را از بین می برد.
2. س
3. اگر مقدار مومنتوم بسیار کم باشد، توانایی خارج کردن مدل از اکسترمم موضعی را ندارد.همچنین اگر این مقدار خیلی بالا باشد ممکن است پیشرفت های حاصل شده توسط مدل تا آنجای فرآیند یادگیری را هم به هم بزند. به این صورت که مدل را از اکسترممی (که ممکن است اکسترمم مطلق هم باشد) به اکسترمم محلی دیگر با دقت کمتر منتقل کند.همان طور که مشاهده میکنیم در ابتدا برای هردو مومنتوم 0.9 و 0.5 یک مقدار دقت را حدودا داریم ولی هرچه میگذرد و به اپیاک 10 نزدیک میشویم دقت 0.5 افزایش سریع میابد ولی 0.9 به آرامی زیاد میشود و حتی در مراحلی کاهش هم دارد.
4. با استفاده از Adam (Adaptive Moment Etimation) بدون ثبت مومنتوم و با نرخ یاد گیري برابر بخش 1 به دقت خوبي میرسیم. زیرا این الگوریتم در واقع بهینه شده ی SGDاست،به این صورت که نرخ یادگیری و مومنتوم در هر مرحله با توجه به شرایط کنونی تعیین می شوند. همچنین در به روز رسانی وزن ها میانگینی از نتایج حاصل روی داده ها در مراحل فعلی و قبلی آزمون مهم است و نه فقط نتیجه آخر.

قسمت دوم: تاثیر epoch

1. معمولا شبکه را در چند اپیاک تمرین می دهیم به این علت که تعداد داده هایی که داریم برای یادگیری کافی نیست و دقت مطلوب را نمی دهد. اگر تعداد داده ها به اندازه کافی زیاد باشد می توانیم به یک اپیاک هم بسنده کنیم.
2. اگر تعداد اپیاک های تمرین بیشتر از حد نیاز باشد، شبکه روی تمام استثنائات داده های آموزش هم حساس می شود که این اتفاق می تواند دقت مدل روی داده های آزمون را کم کند. به این فرایند overfiting می گویند. برای حل این مشکل چندین راه حل وجود دارد، به عنوان مثال می توانیم قبل از اینکه مدل به جایی برسد که دقت روی داده های آزمون با بیشتر شدن دقت روی داده های یادگیری کمتر شود فرایند را متوقف کنیم که به این روش early stoppingمی گویند. همچنین می توانیم با اعمال محدودیت هایی روی مقادیر وزن و بایاس، یا وارد کردن مقداری نویز به داده های یادگیری، یا حذف کردن برخی از مقادیر حاصل شده در خروجی یک لایه قبل از رسیدن به ورودی لایه بعد (به مجموعه این کارها regularization می گویند.) جلوی این اتفاق را بگیریم.

در این جا مشاهده میکنیم که با 20 epoch مدل بهتر فیت شده و هنوز به مرحله overfitting نرسیده است.

قسمت سوم: تاثیر loss function

این تابع برای محاسبه خطا در تخمین کمیت های پیوسته مناسب است، نه برای مسائلی مانند دسته بندی. چرا که در این مسائل خروجی گسسته است و در نتیجه مقدار خطا هم مقداری گسسته خواهد بود که مشتق پذیر نیست. بنابراین همانطور که مشاهده می شود یادگیری انجام نمی گیرد و دقت ثابت می ماند.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(ObservedVali - PredictedVali)^2$$

با توجه به خروجی بدست آمده مشاهده میکنیم که دقت در طی 20 epoch تغییر چندانی نمیکند همچنین مقدار مناسبی نیز ندارد

قسمت چهارم: regularization

1. به طور کل Regularizationروش هایی است که برای جلوگیری از overfittingاستفاده می شود.
   ساده ترین و رایج ترین روش آن اضافه کردن پنالتی به loss function متناسب با سایز وزن ها در مدل است.
2. Drop out یک روش regularization است حجم زیادی از داده های train را به صورت موازی و با معماری های مختلف انجام میدهد. در هنگام یادگیری تعدادی از لایه های خروجی به طور رندوم در نظر گرفته نمیشوند(dropped out) این عمل باعث میشود تا آن لایه بنظر لایه ای با تعداد نود متفاوت بیاید که در نتیجه هر اپدیت از لایه ها در زمان یادگیری با "view" متفاوتی از لایه اولیه اجرا میشود(تا جلوی تمرین اضافه و اورفیت شدن را بگیرد).
3. مدل regression که از L_1 استفاده میکند را lasso regression و مدلی که از L_2 استفاده میکند ridge regression نام دارد و تفاوت انها درمفهوم پنالتی است.
   Ridge regression مقدار مربع coefficientها (ضریب ها) را به عنوان پنالتی به loss function اضافه میکند درحالی که L_1 مقدار قدرمطلق coefficientها(ضریب) را در loss function اضافه میکند.
4. همانطور که گفتیم ، در drop out برخی مقادیر را drop می کنیم. با این کار ما داده های خود را بسیار ساده می کنیم.ولی این خوب نیست. اگرچه ما noise را کاهش دادیم اما در پایان برای داده های آزمایش می توانیم ببینیم که میزان loss افزایش یافته و دقت آن کاهش یافته است.
5. اما در مورد L2 ، مشکل فوق را نیز می توانیم ببینیم اما این چندان بد نیست و در نتیجه، نتیجه بسیار خوبی گرفتیم و این به دلیل کاهش noise و تغییرات بزرگ در وزن است که با

regularization ایجاد شده است.همچنین L_2 جلوی زیاد شدن بایاس ها را می گیرد ولی نمی گذارد به صفر برسند.

$$L(x,y) \equiv \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n} \theta_i^2$$

L2 Regularization