به نام خدا

مبانی هوش محاسباتی

پروژه شبکه‌های عصبی

استاد درس:

دکتر عبادزاده

پاییز ۱۴۰۱

فهرست مطالب

مقدمه

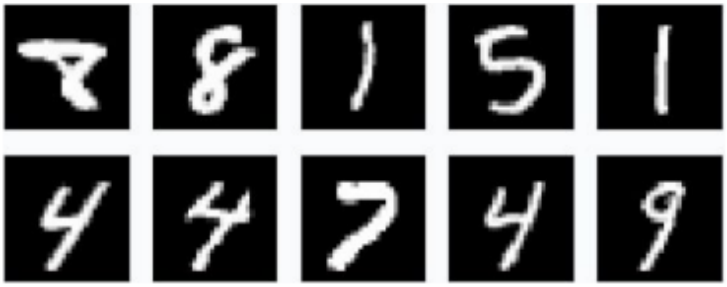
یکی از کاربردهای شبکه‌های عصبی، دسته‌بندی (classification) است. در این پروژه قصد داریم به سراغ دسته‌بندی تصاویر برویم. می‌خواهیم با استفاده از شبکه‌های عصبی مختلف (به طور خاص fully connected networks) مدلی بسازیم که عکس‌هایی را به عنوان ورودی گرفته و دسته‌بندی هر عکس رو تشخیص بدهد.

بعد از پیاده‌سازی شبکه‌ی عصبی fully connected، می‌تونید قسمت امتیازی رو هم انجام بدین، که طی اون با معماری‌های پیچیده‌تر شبکه‌های عصبی آشنا می‌شیم.

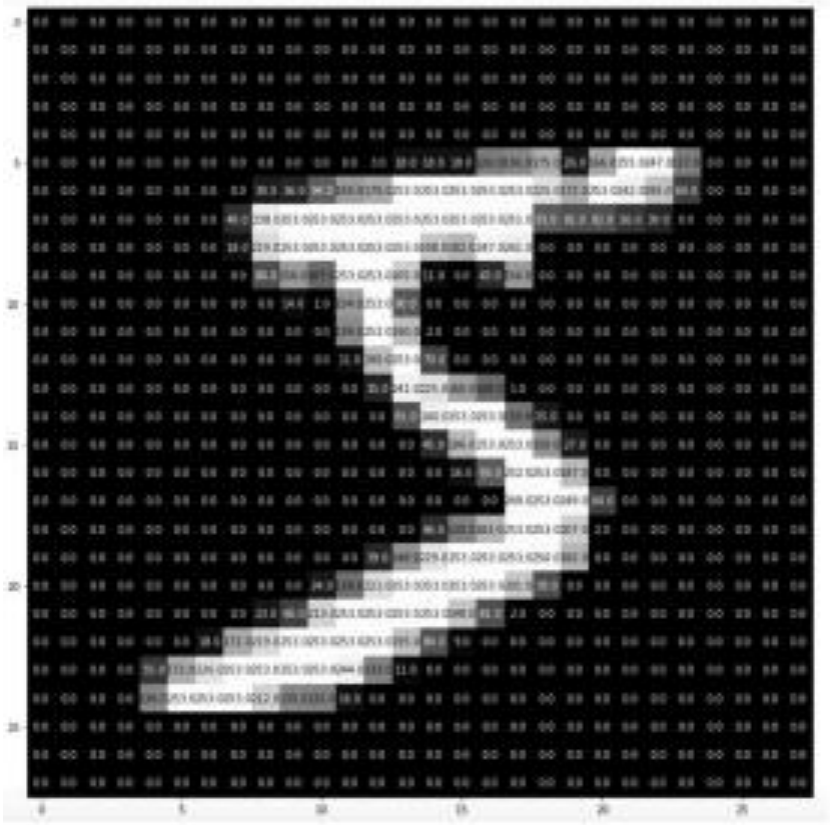
برای انجام این پروژه شما میتوانید با استفاده از google colab و jupyter notebook مراحل را پیاده سازی و نتایج را گزارش کنید. به این منظور، در این لینک می‌توانید با اصول اولیه و کار با jupyter notebook آشنا شوید. همچنین در این لینک می‌توانید آشنایی ابتدایی از google colab بدست آورید.

شرح مساله

در این مساله، ما تصاویری سیاه سفید به عنوان ورودی داریم که در هر تصویر یک رقم نوشته شده است؛ مدل ما باید تشخیص بده که اون رقم چیه.



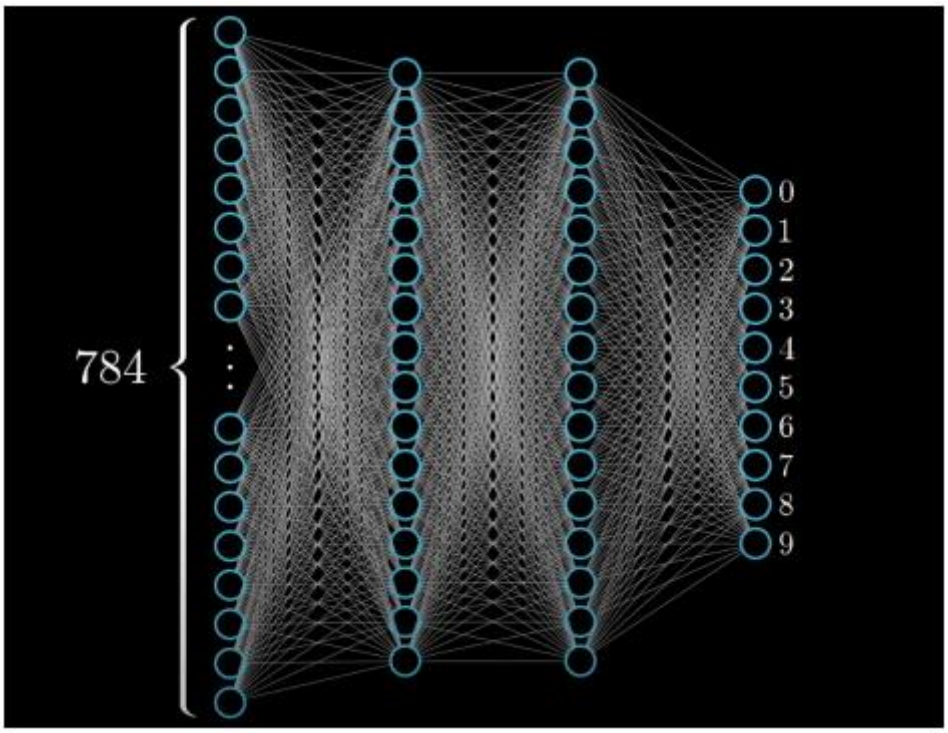
دیتاست مورد استفاده ما MNIST هست که تصاویر اون در ابعاد ۲۸ در ۲۸ هستن. در نتیجه لایه ورودی شبکه عصبی ما ۲۸ x ۲۸ = ۷۸۴ نورون خواهد داشت. این نورون میزان روشنایی اون پیکسل رو به صورت یک عدد int از ۰ تا ۲۵۵ نشون میدن.



این مقادیر رو باید بر ۲۵۶ تقسیم کنیم تا میزان Activation نورون‌های ورودی در بازه‌ی ۰ تا ۱ قرار بگیرن.

با توجه به اینکه مدل ما در نهایت قراره یکی از ۱۰ رقم انگلیسی را تشخیص بده، پس ما برای لایه خروجی به ۱۰ نورون نیاز داریم و اون نورونی که بیشترین Activation رو داره به عنوان رقم تشخیص داده شده مدل ما انتخاب میشه.

برای این شبکه عصبی، دو لایه پنهان یا Hidden Layer در نظر می‌گیریم که هر کدوم ۱۶ نورون دارن. بنابراین ساختار شبکه عصبی ما به شکل زیر میشه:



شبه کد

شبه کد فرایند یادگیری شبکه عصبی طبق روش Stochastic Gradient Descent به صورت زیر هست:

```
Allocate W matrix and vector b for each layer.
Initialize W from standard normal distribution, and b = 0, for each layer.
Set learning_rate, number_of_epochs, and batch_size.
for i from 0 to number_of_epochs:
    Shuffle the train set.
    for each batch in train set:
        Allocate grad_W matrix and vector grad_b for each layer and initialize to 0.
        for each image in batch:
            Compute the output for this image.
            grad_W += dcost/dW for each layer (using backpropagation)
            grad_b += dcost/db for each layer (using backpropagation)
        W = W - (learning_rate × (grad_W / batch_size))
        b = b - (learning_rate × (grad_b / batch_size))
```

ایده‌ی این روش اینه که به جای اینکه در هر مرحله از یادگیری مدل، بیایم و با کل داده‌های مجموعه train کنیم، می‌تونیم در هر پیمایش، داده‌ها رو به بخش‌هایی تحت عنوان mini-batch تقسیم کنیم، گرادیان مربوط به هر سمپلِ اون mini-batch رو بدست بیاریم، و در نهایت میانگین اون‌ها رو به دست بیاریم و بعد تغییرات رو اعمال کنیم. این کار باعث می‌شه که محاسبات در هر پیمایش کم‌تر بشه و سرعت همگرایی افزایش پیدا کنه.

تعداد سمپل‌هایی که در هر مرحله باهاشون کار می‌کنیم رو بهش می‌گن mini-batch. همچنین، به هر دور که تمامی mini-batchها (و در نتیجه تمامی سمپل‌ها) پیمایش می‌شن، می‌گن epoch (بخوانید ای‌پاک!)

قدم اول: دریافت دیتاست

در قدم اول نیازه که دیتاست پروژه رو از لینکی که بالاتر گذاشتیم دریافت کنید و توی کد خودتون load کنید. این دیتاست شامل ۶۰۰۰۰ نمونه در مجموعه آموزش (Training Set) و ۱۰۰۰۰ نمونه در مجموعه تست (Test Set) هست.

توضیحات مربوط به فرمت فایل‌ها و نحوه خوندنشون به طور کامل در سایت مربوطه گفته شده؛ اما برای اینکه کارتون ساده تر بشه، کد پایتون مربوط به خوندن فایل‌ها رو می‌تونید از اینجا دریافت کنید (فایل‌های مربوط به دیتاست رو کنار کد خودتون بذارید).

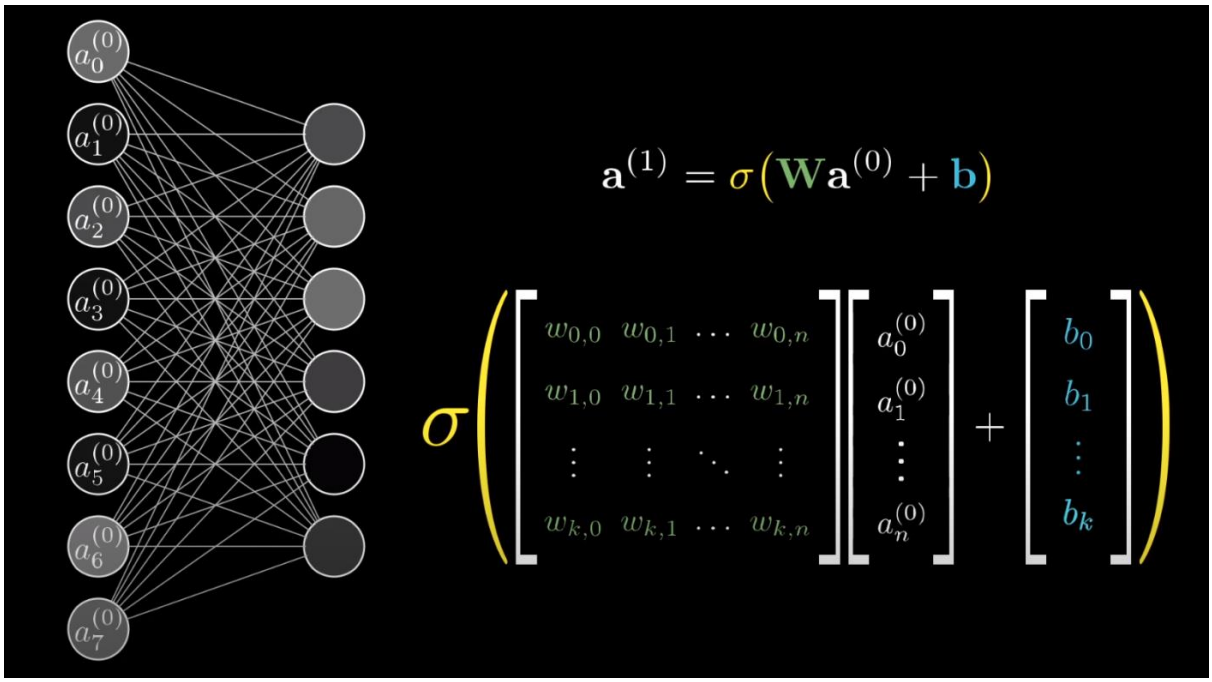
در آخر چند تا عکس رو برای خودتون پلات کنید و لیبل مربوط بهش رو ببینید تا مطمئن شید فایل‌ها به درستی خونده میشن.

قدم دوم: محاسبه خروجی (Feed Forward)

همانطور که می‌دانید، برای محاسبه‌ی خروجی از روی ورودی در شبکه‌های عصبی، در هر لایه عملیات زیر انجام می‌شود:

$$a^{(L+1)} = \sigma\left(W^{(L+1)} \times a^{(L)} + b^{(L+1)}\right)$$

در نتیجه در پیاده‌سازی شبکه عصبی، برای وزن‌های بین هر دو لایه، یک ماتریس k در n نظر می‌گیریم که k، تعداد نورون‌های لایه بعدی و n، تعداد نورون‌های لایه‌ی فعلی است. در نتیجه هر سطر ماتریس W، وزن‌های مربوط به یک نورون خاص در لایه‌ی بعدی است. همچنین برای بایاس‌های بین هر دو لایه نیز، یک بردار جداگانه در نظر گرفته می‌شود که ابعاد آن برابر با تعداد نورون‌های لایه بعدی است.



در این قدم از پروژه، ۱۰۰ داده (داده‌های ۱۰۰ تا عکس) مجموعه train را جدا کنید و پس از مقداردهی اولیه‌ی ماتریس وزن‌ها با اعداد تصادفی نرمال و بایاس‌ها به صورت بردارهای تماما صفر، خروجی مربوط به این ۱۰۰ داده را محاسبه کنید. محاسبه خروجی را باید به طریقی که بالاتر گفتیم (یعنی به صورت ضرب و جمع ماتریسی/برداری و اعمال تابع سیگموید) انجام دهید. در انتها در لایه آخر، نورونی که بیشترین مقدار را دارد به عنوان تشخیص مدل در نظر گرفته می‌شود که در واقع معادل دسته‌ی مربوط به آن نورون می‌باشد.

سپس دقت (Accuracy) مدل که معادل است با تعداد عکس‌هایی که به درستی تشخیص داده شده تقسیم بر تعداد کل عکس‌ها، را گزارش کنید. با توجه به اینکه هنوز فرآینده یادگیری طی نشده و مقداردهی‌ها تصادفی بوده، انتظار می‌رود دقت در این حالت، به طور میانگین به ۱۰ درصد نزدیک باشد.

توجه: حتما برای کار با ماتریس‌ها، از NumPy استفاده کنید.

قدم سوم: پیاده‌سازی BackPropagation

همانطور که می‌دانید، فرآیند یادگیری شبکه‌ی عصبی به معنی مینیمم کردن تابع cost است:

$$Cost = \sum_{j=0}^{n_L-1} (a_j^{(L)} - y_j)^2$$

این کار به کمک روش Gradient Descent انجام می‌شود که در آن با بدست آوردن مشتقات جزئی تابع Cost نسبت به تمامی پارامترها (یعنی همان گرادیان)، تغییرات مورد نظر بر روی پارامترها را انجام می‌دهیم:

$$(W‘b) = (W‘ b) - \alpha\nabla Cost$$

بدست آوردن این مشتق‌ها به کمک backpropagation انجام می‌شود.

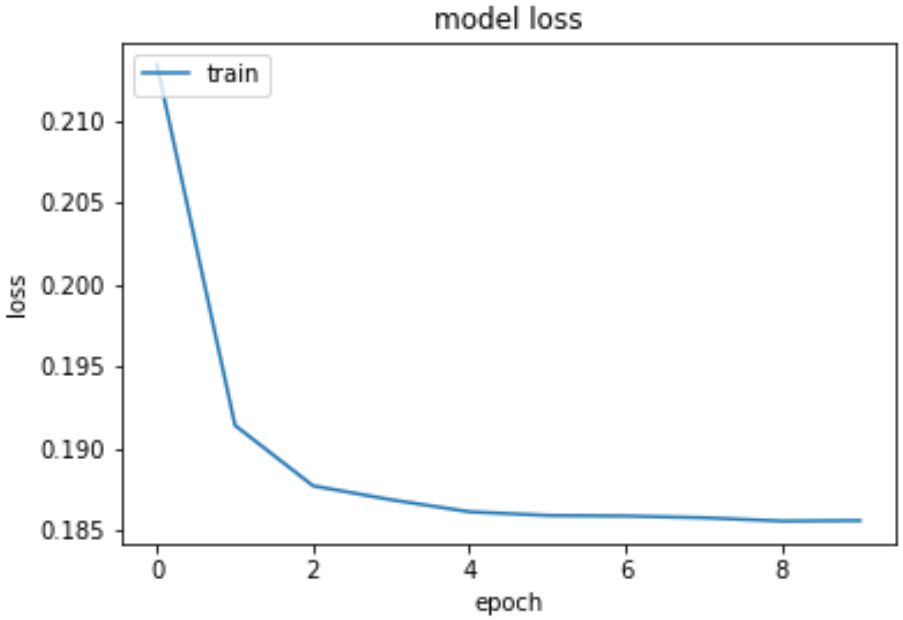
در این قدم از پروژه، شبه‌کدی که بالاتر گفته شد را به طور کامل پیاده‌سازی کنید. مجموعه train را، همان ۱۰۰ داده که در مرحله‌ی قبل گفته شد، در نظر بگیرید. Hyperparameter ها را هم به صورت زیر قرار دهید:

batch_size = 10, learning_rate = 1, epoch = 20

برای بدست آوردن گرادیان‌ها، ماتریس‌هایی و بردارهایی به ابعاد همان w و b و a ها در نظر بگیرید و با for زدن روی درایه‌ها، مشتق جزئی cost نسبت به آن عنصر را بدست آورید.

در پایان این مرحله، دقت مدل و زمان اجرای فرآیند یادگیری را برای همان ۱۰۰ داده گزارش کنید. با توجه به اینکه تعداد epoch و داده‌های آموزشی کم هستند، انتظار می‌رود ر پایان فرآیند یادگیری، دقت مدل در این حالت، به طور میانگین حدود ۲۵-۵۰ درصد باشد. اگر زمان اجرا برایتان معقول بود می‌توانید به ازای تعداد epoch بیشتر هم، کدتان را تست کنید و نتایج را گزارش کنید.

همچنین میانگین cost نمونه‌ها را در هر epoch محاسبه کنید و در آخر پلات کنید. انتظار می‌رود که این میانگین‌ها، در هر epoch کاهش پیدا کند و در نتیجه نمودار نهایی شبیه به نمودار زیر باشد:



توجه: اگر این سیر نزولی در cost دیده نشود، به احتمال زیاد مشکلی در پیاده‌سازی الگوریتم وجود دارد.

قدم چهارم: Vectorization

تا اینجا تنها با ۱۰۰ داده اول دیتاست کار کردیم چون زمان اجرای فرآیند آموزش فعلی خیلی زیاد است و برای یادگیری شبکه بهینه نیست. برای رفع این مشکل، از مفهمومی تحت عنوان vectorization استفاده می‌کنیم. این مفهوم به این معنی است که در عوض استفاده از for بر روی درایه‌ها، عملیات مدنظر را به شکل عملیات ماتریسی (ضرب و جمع ماتریسی و برداری، ضرب داخلی، ترانهاده کردن و اعمال توابع روی تک تک عناصر ماتریس‌ها) پیاده‌سازی کنیم.

این کار سبب می‌شود تا زمان اجرای کد به میزان قابل توجهی کمتر شود. دلیل اصلی این تسریع در محاسبات این است که عملیات ماتریسی به خوبی می‌توانند موازی‌سازی شوند و به صورت چندهسته‌ای اجرا شوند. همچنین پردازنده‌ها دستورالعمل‌هایی مخصوص کار کردن با داده‌های بزرگ و برداری دارند که به طور کاراتر اجرا می‌شوند.

در مرحله اول feedforward الگوریتم را از اول به صورت vectorized پیاده‌سازی کردیم. حال در این مرحله باید backpropagation را هم به صورت vectorized کنید. در پایان این مرحله انتظار می‌رود که محاسبه‌ی مشتقات جزئی هر لایه (یعنی مشتقات نسبت به W و b و aها) بدون for زدن انجام شوند.

برای مثال، کد زیر که برای محاسبه‌ی گرادیان برای وزن‌های لایه آخر:

```
for j in range(10):
    for k in range(16):
        grad_W3[j, k] += a2[k, 0] * sigmoid_deriv(z3[j, 0]) * (2 * a3[j, 0] - 2 * y[j, 0])
```

را می‌توانید به صورت زیر بنویسید:

```
grad_W3 += (2 * sigmoid_deriv(z3) * (a3 - y)) @ (np.transpose(a2))
```

(علامت @ برای ضرب ماتریسی است).

یا محاسبه گرادیان برای نورون‌های لایه یکی مانده به آخر به شکل زیر است:

```
grad_a2 = np.zeros((16, 1))
for k in range(16):
    for j in range(10):
        grad_a2[k, 0] += W3[j, k] * sigmoid_deriv(z3[j, 0]) * (2 * a3[j, 0] - 2 * y[j, 0])
```

که به صورت زیر vectorized می‌شود:

```
grad_a2 = np.transpose(W3) @ (2 * sigmoid_deriv(z3) * (a3 - y))
```

سایر عبارات را هم مشابه حالات توضیح داده شده، vectorized کنید.

در پایان این مرحله، انتظار می‌رود که کدتان در مدت زمان خیلی کمتری نسبت به مرحله‌ی قبل اجرا شود. در نتیجه تعداد epoch را افزایش دهید به عدد ۲۰۰ و دقت مدل نهایی، زمان اجرای فرآیند یادگیری و همچنین پلات cost در طی زمان را گزارش کنید.

قدم پنجم: تست کردن مدل

حال که الگوریتم را تا حد خوبی بهینه کرده‌ایم، می‌تونیم بر روی کل داده‌ها، train را انجام دهیم.
هایپرپارامترها را به شکل زیر تغییر دهید:

batch_size = 50, learning_rate = 1, epoch = 5

در پایان این مرحله، دقت مدل را برای مجموعه‌ی train و همچنین برای مجموعه test گزارش کنید.
همچنین همانند قبل میانگین Cost را نیز پلات کنید.

اگر پیاده‌سازی درست انجام شده باشد، انتظار می‌رود که دقت مدل برای train و test حدود ۹۰ درصد باشه.

بخش تحقیقی امتیازی

سوال اول

هدف از داشتن مجموعه داده‌ی اعتبارسنجی (Cross Validation) چیست و چه فرقی با مجموعه داده‌ی تست (Test Set) دارد؟

سوال دوم

دو ورژن پیشرفته گرادیان کاهشی ، گرادیان کاهشی دسته‌ای[۱] ، گرادیان کاهشی تصادفی[۲] و گرادیان کاهشی دسته‌ای کوچک[۳] نام دارد. در مورد هر کدام تحقیق کنید و مزایا و معایب هرکدام را بیان کنید.

سوال سوم

آموزش شبکه های عصبی عمیق با ده ها لایه چالش برانگیز است زیرا می توانند به وزن های تصادفی اولیه و پیکربندی الگوریتم یادگیری حساس باشند. نرمال‌سازی دسته‌ای[۴]، تکنیکی برای آموزش شبکه‌های عصبی بسیار عمیق است که ورودی‌های یک لایه را برای هر مینی بچ استاندارد می‌کند. این امر باعث تثبیت فرآیند یادگیری و کاهش چشمگیر تعداد دوره های آموزشی مورد نیاز برای آموزش شبکه های عمیق می شود. در مورد این تکنیک تحقیق کرده و آن را شرح دهید.

---

۱ Batch Gradient Descent
۲ Stochastic Gradient Descent
۳ Mini-batch Gradient Descent
۴ Batch normalization

سوال چهارم

در شبکه‌های کانولوشنی (CNN) وظیفه لایه Pooling را مختصر توضیح دهید و روش‌های مورد استفاده آن را نام ببرید؛ در انتها با ذکر دلیل توضیح دهید چرا برای مسائل دسته بندی ای که با عکس سر و کار دارد، شبکه‌های کانولوشنی بر شبکه‌های ساده ارجحیت دارند؟

بخش کد امتیازی

هدف در این بخش آشنایی و استفاده از یکی از پلتفرم‌های محبوب توسعه هوش مصنوعی و شبکه‌های عصبی یعنی پایتورچ[۵] می‌باشد. در این بخش هم‌چنین به صورت اختیاری نیز می‌توانید از سرویس گوگل کولب[۶] نیز برای اجرای مدلتان استفاده کنید.

پایتورچ توابع از پیش پیاده شده بسیاری جهت محاسبات عملیاتی نظیر عملیات پردازشی نیز دارد بعلاوه این کتابخانه از پلتفرم پردازشی شرکت اپل برای استفاده از کارتهای گرافیکی مختص این شرکت و همچنین شرکت انویدیا[۷] یعنی کودا[۸] نیز پشتیبانی میکند.

در این بخش شما یک ژوپیتر نوت‌بوک[۹] در کنار دیتاست بخش امتیازی و فایل‌های دیگر پروژه دریافت خواهید کرد که محیط توسعه‌ای شما برای بخش امتیازی و تصحیح خواهد بود.

دیتاستی که در این بخش آورده شده است دیتاست تصاویر MRI میباشد که شامل ۳ کلاس می‌باشند. با توجه به ماهیت تسک و برنامه، ما از یادگیری انتقالی[۱۰] میخواهیم استفاده بکنیم که بیشتر بخش‌های آن نیز به صورت از پیش آماده برای شما نوشته شده است. مدل مورد استفاده در این بخش مدل معروف resnet50 هست که به صورت پیاده‌سازی شده و آموزش دیده در پایتورچ وجود دارد.(مدل resnet50 غیرقابل تغییر است)

یکی از نکاتی که در تسک‌های مختلف یادگیری ماشین مورد توجه است کمبود داده‌های تمرین هست که یکی از راه‌های حل آن برای آموزش مدل‌ها استفاده از افزایش داده‌ها[۱۱] می‌باشد. (این مورد نیز پیاده سازی

---

۵ PyTorch
۶ Google Colab
۷ Nvidia
۸ CUDA
۹ Jupyter Notebook
۱۰ Transfer Learning
۱۱ data augmentation

شده در نوت‌بوک وجود دارد) با استفاده از این تکنیک میشود تعداد داده‌های آموزش مدل را تا یک میزان مناسب زیاد کرد.

شما در این بخش باید لایه‌های عصبی متنوعی را برحسب درسی که در کلاس آموزش دیدید در ادامه‌ی خروجی‌های مدل resnet50 اضافه کنید، نرخ یادگیری، اپتیمایزر، تابع هزینه و دیگر موارد را خودتان از پایتورچ نوشته و بهبود دهید تا مدلتان به بهترین میزان دقت در داده‌های تست برسد.

در نوت‌بوک این بخش کامنت‌های مناسب زیادی برای شما نوشته شده‌اند تا متوجه کد بشوید.

نمره‌دهی بخش به صورت نسبی میان دانشجویان می‌باشد، به این صورت که مدل‌های ذخیره شده (سلول ذخیره سازی در نوت‌بوک توسط ما نوشته شده است و بعد از کار خودتان کافی است از آن استفاده کنید) برروی نمونه‌های تست اختصاصی تدریس‌یاران ارزیابی میشوند و دقت مدل هر دانشجو بدست می‌آید. سپس دقت‌ها رتبه بندی شده و بالاترین دقت(ها) نمره کامل خواهند گرفت و بقیه نیز به صورت نسبی نمره میگیرند.

لینک دیتاست برای قسمت امتیازی در اینجا در دسترس است.

فایل نوت بوک قسمت امتیازی نیز با نام Bonus_Project1 در فولدر آپلود شده قرار دارد.

نکات تحویل پروژه

- موارد تحویلی در فایل گزارشی که باید ارسال کنید در صورت پروژه با رنگ قرمز مشخص شده اند.

- فایل نهایی تحویلی شما برای کدها یک فایل Jupyter notebook **یا** فایل‌های پایتون (.py) به همراه یک فایل pdf که همان گزارش شما است می‌باشد.

- فایل تحویلی شما باید با فرمت zip به صورت:

  ANN_{studentID}.zip باشد (برای مثال ANN_9931000.zip).

- در صورت داشتن هرگونه سوال یا ابهام از طریق ایمیل زیر با ما در ارتباط باشید.

  ci.fall.1401@gmail.com

- مهلت ارسال پروژه تا ساعت ۲۳:۵۵ روز جمعه ۱۱ آذر می‌باشد.

موفق باشید

تیم تدریس یاری درس مبانی هوش محاسباتی