دانشگاه صنعتی امیرکبیر
دانشکده مهندسی کامپیوتر

مبانی هوش محاسباتی
پروژه شبکه‌های عصبی
(Handwritten Digit Recognition)

استاد درس: دکتر عبادزاده

بهار ۱۴۰۰

فهرست

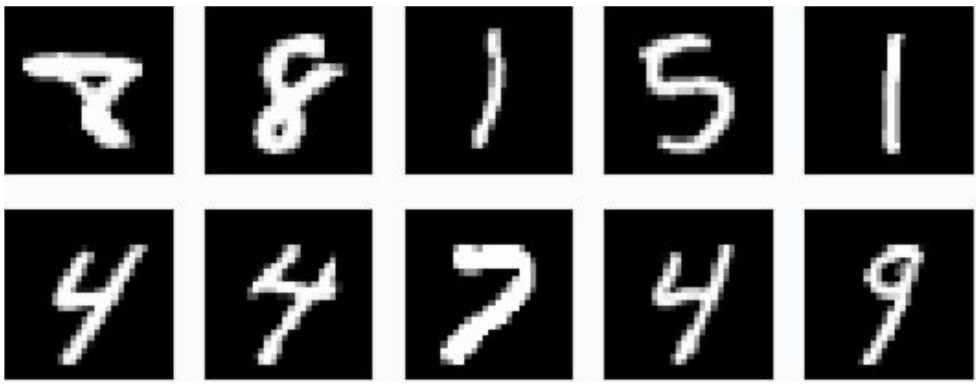
مقدمه

یکی از کاربردهای شبکه‌های عصبی، تشخیص الگو (Pattern Recognition) می‌باشد. در این پروژه قصد داریم سراغ مسئله تشخیص ارقام دست‌نویس برویم. این مسئله، یکی از مسائل کلاسیک و معروف پردازش تصویر هستش که افراد مختلف با متدهای گوناگونی مثل SVM، K-Nearest Neighbor و معماری‌های مختلف شبکه عصبی سراغش رفتن[1]. ما در اینجا می‌خواهیم به کمک شبکه‌های عصبی Feedforward Fully Connected که توی درس باهاش آشنا شدیم، این مسئله رو حل کنیم.

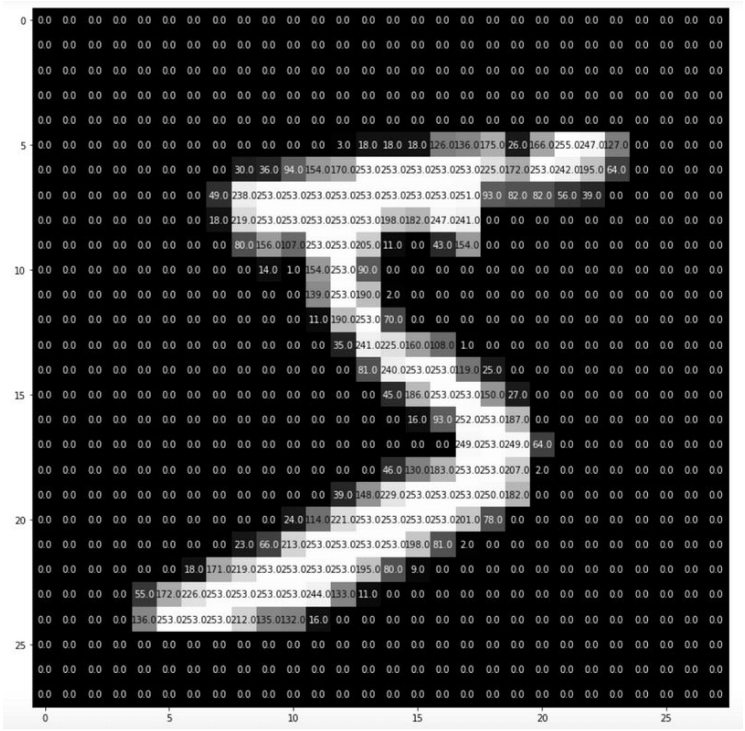
[1] توی این لینک می‌تونید لیست متدها و میزان خطای هر کدوم رو مشاهده کنید.

شرح مسئله

در این مسئله، ما تصاویری سیاه-سفید به عنوان ورودی دریافت می‌کنیم که در هر تصویر، یک رقم نوشته شده. مدل ما باید تشخیص بده که اون رقم، چه رقمی هستش.



دیتاست‌ای که قراره ازش استفاده کنیم، دیتاست MNIST هستش. تصاویر این مجموعه، ابعاد ۲۸ در ۲۸ دارن. در نتیجه لایه ورودی شبکه عصبی ما دارای ۲۸ × ۲۸ = ۷۸۴ نورون هستش که هر نورون میزان روشنایی اون پیکسل رو به صورت یک عدد int از 0 تا 255 نشون میده:

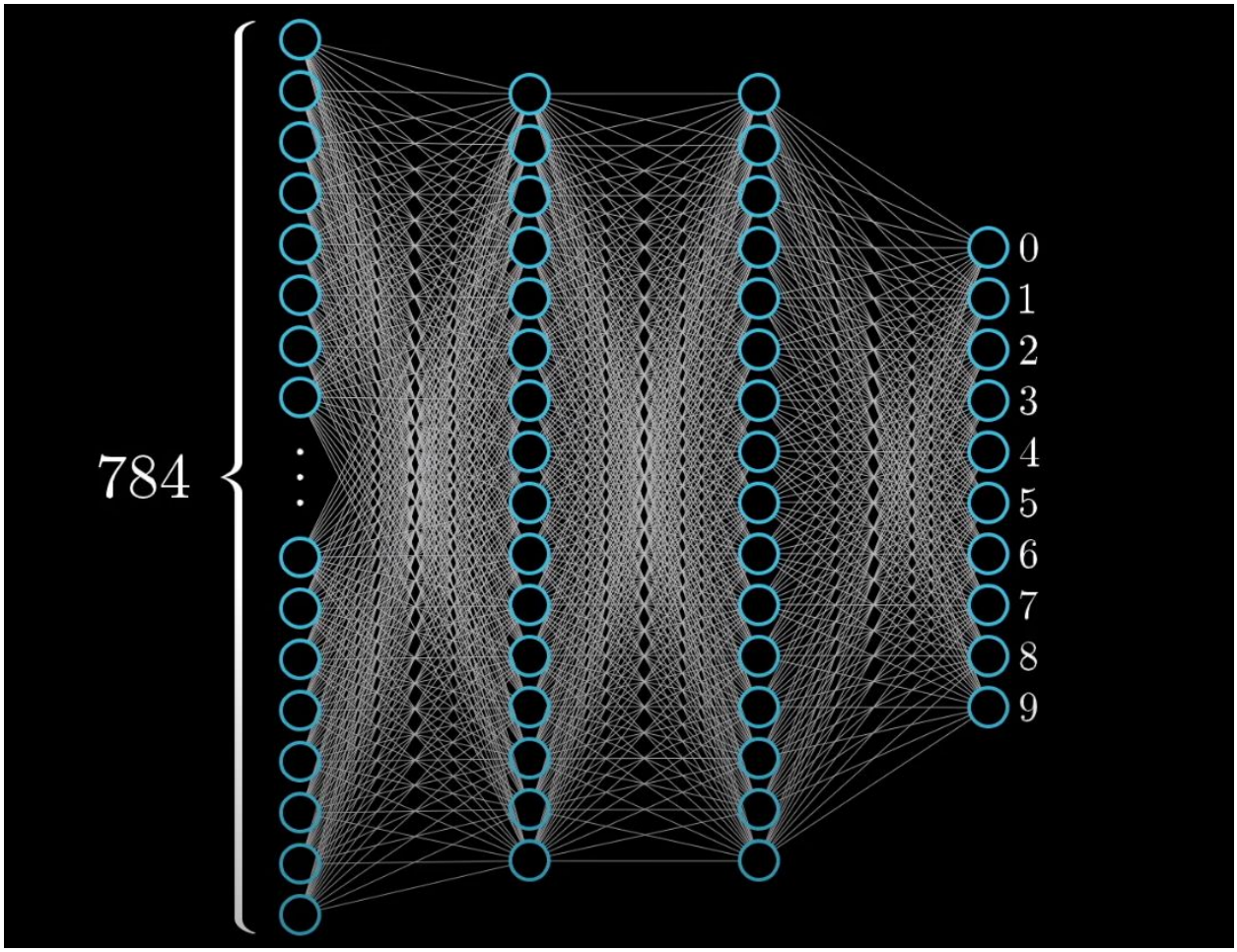


البته این مقادیر رو باید تقسیم بر ۲۵۶ کنیم تا میزان Activation نورون‌های ورودی در بازه‌ی 0 تا 1 قرار بگیره.

با توجه به اینکه مدل ما در نهایت قراره یکی از ۱۰ رقم انگلیسی رو تشخیص بده، لایه خروجی شبکه عصبی ما دارای ۱۰ نورون خواهد بود. اون نورونی که بیشترین Activation رو داره، به عنوان رقم تشخیص داده‌شده توسط مدل ما، انتخاب میشه.

برای این شبکه عصبی، دو لایه پنهان (Hidden Layer) در نظر می‌گیریم که هر کدوم دارای ۱۶ نورون هست.

پس ساختار شبکه عصبی ما به صورت زیر می‌باشد:



شبه‌کد

شبه‌کد فرآیند یادگیری شبکه عصبی ما طبق روش Stochastic Gradient Descent، به شکل زیر هستش:

```
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 کار کنیم، می‌تونیم در هر پیمایش، داده‌ها رو به بخش‌هایی تحت عنوان mini-batch تقسیم کنیم، گرادیان مربوط به هر سمپلِ اون mini-batch بدست بیاریم، و در نهایت، میانگین اونها رو بدست بیاریم، و بعد تغییرات رو اعمال کنیم. این کار باعث میشه که محاسبات در هر پیمایش کمتر بشه و زمان یادگیری مدل ما، کاهش پیدا کنه.

تعداد سمپل‌هایی که هر مرحله باهاشون کار می‌کنیم رو بهش میگن batch size. همچنین، به هر دور که تمامی mini-batch ها (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 اش کنید. این دیتاست شامل ۶۰,۰۰۰ نمونه در مجموعه Train و ۱۰,۰۰۰ نمونه در مجموعه‌ی Test هستش.

توضیحات مربوط به فرمت فایل‌ها و نحوه‌ی خوندن‌شون، به طور کامل در سایت مربوطه گفته شده؛ اما برای اینکه کارتون ساده‌تر بشه، کد پایتون مربوط به خوندن فایل‌ها رو می‌تونید از اینجا دریافت کنید. (۴ فایل مربوط به دیتاست رو کنار کد بذارید.)

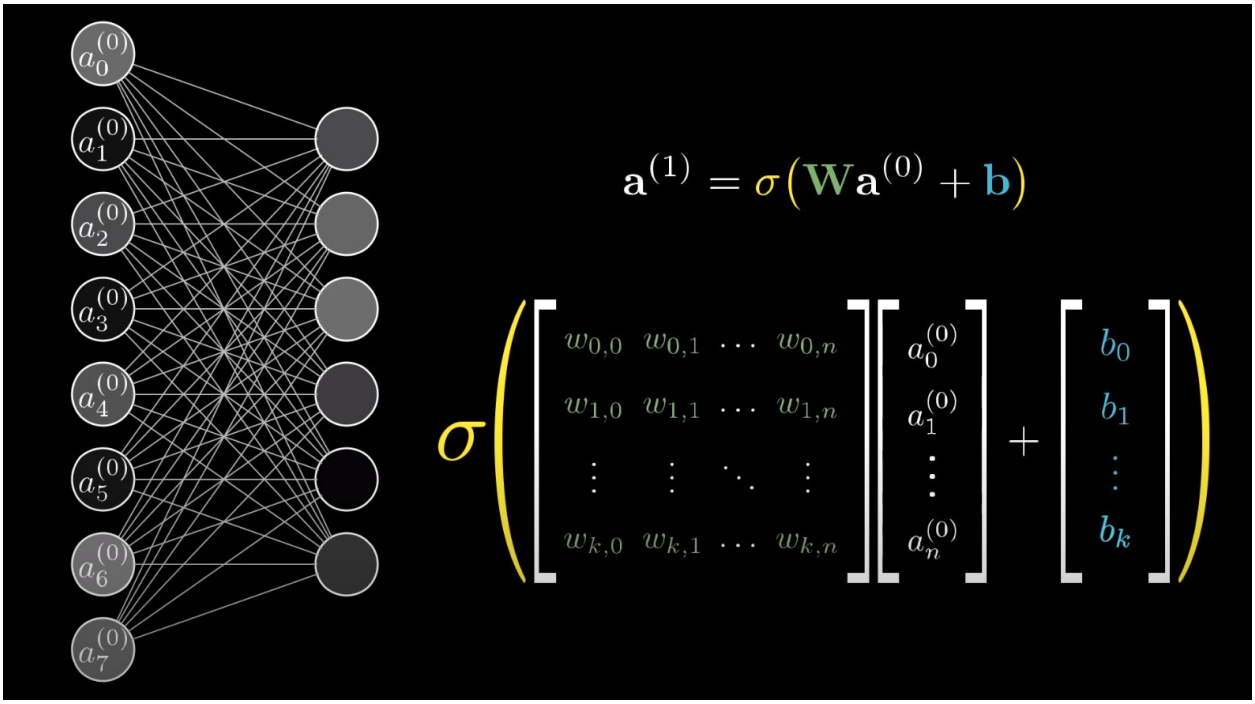
در پایان این قدم، چندتا از عکس‌ها رو پلات کنید و لیبل مربوطه بهش رو هم پرینت/پلات کنید تا مطمئن شید که خوندن فایل‌ها به درستی انجام شده.

قدم دوم: محاسبه خروجی (Feedforward)

همونطور که می‌دونید، برای محاسبه‌ی خروجی از روی ورودی در شبکه‌های عصبی، در هر لایه عملیات زیر انجام میشه:

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

در نتیجه، توی پیاده‌سازی شبکه عصبی، برای وزن‌های بین هر دو لایه، یک ماتریس k در n در نظر می‌گیریم که k، تعداد نورون‌های لایه‌ی بعدی و n، تعداد نورون‌های لایه‌ی فعلی. در نتیجه، هر سطر ماتریس W، وزن‌های مربوط به یک نورون خاص در لایه‌ی بعدی هستش. همچنین، برای بایاس‌های بین هر دو لایه هم، یک بردار جداگانه در نظر می‌گیریم که ابعادش برابر با تعداد نورون‌های لایه بعدی هستش.



در این قدم از پروژه، ۱۰۰ عکس اول مجموعه Train رو 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 عکس رو محاسبه کنید. محاسبه خروجی رو باید به طریقی که بالاتر گفتیم (یعنی به صورت ضرب و جمع ماتریسی/برداری و اعمال تابع سیگموید) انجام بدید.

سپس دقت (Accuracy) مدل؛ یعنی، تعداد عکس‌هایی که به درستی تشخیص داده شده تقسیم بر تعداد کل عکس‌ها، را گزارش کنید. با توجه به اینکه هنوز فرآیند یادگیری طی نشده و مقدارهی‌ها رندوم بوده، انتظار میره که دقت حدود ۱۰ درصد باشه.

نکته: اگر پروژه رو با پایتون انجام می‌دید، حتما برای کار با ماتریس‌ها، از **NumPy** استفاده کنید.

قدم سوم: پیاده‌سازی Backpropagation

همونطور که می‌دونید، فرآیند یادگیری شبکه‌ی عصبی به معنی مینیمم کردن تابع Cost هستش:

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

که این‌کار به کمک روش Gradient Descent انجام میشه که در اون با بدست آوردن مشتقات جزئی تابع Cost نسبت به تمامی پارامترها (یعنی همان گرادیان)، تغییرات مورد نظر بر روی پارامترها رو انجام می‌دیم:

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

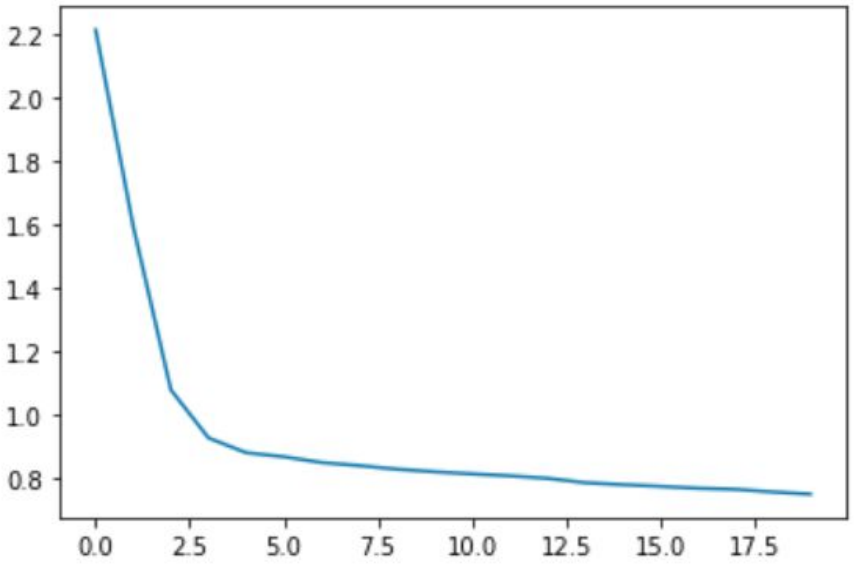
بدست آوردن این مشتق‌ها، به کمک Backpropagation انجام میشه.
در مورد Backpropagation و پیاده‌سازیش به طور مفصل توی اسلاید توضیح دادیم.

در این قدم از پروژه، شبه‌کدی که بالاتر گفته شد رو به طور کامل پیاده‌سازی کنید. مجموعه Train رو، همون ۱۰۰ عکس اول که تو مرحله‌ی قبل گفته شد، در نظر بگیرید. Hyperparameter ها رو هم بدین شکل ست کنید: مقدار batch_size برابر با ۱۰، ضریب یادگیری برابر با ۱ و تعداد epoch ها برابر با ۲۰.

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 اون عنصر رو بدست آورید.

در پایان این مرحله، دقت مدل رو برای همان ۱۰۰ عکس، گزارش کنید. با توجه به اینکه تعداد Epoch ها کم هستش، انتظار میره در پایان فرآیند یادگیری، دقت مدل تا حدود ۲۵-۵۰ درصد باشه. اگر زمان اجرا معقول بود (حدودا ۲-۳ دقیقه) می‌تونید به ازای تعداد epoch بیشتر هم، کدتون رو تست کنید.

همچنین میانگین Cost نمونه‌ها را در هر epoch محاسبه کنید و در آخر پلات کنید. انتظار میره که این میانگین‌ها، در هر epoch کاهش پیدا کنه و در نتیجه نمودار نهایی شبیه نمودار زیر بشه:



اگر این سیر نزولی در Cost ها دیده نشه، حتما مشکلی توی پیاده‌سازی الگوریتم وجود داره.

در آخر، زمان اجرای فرآیند یادگیری رو هم گزارش کنید.

قدم چهارم: Vectorization

دلیل اینکه تا اینجا فقط با ۱۰۰ عکس اول دیتاست کار کردیم اینه که زمان اجرای پیاده‌سازی فعلی‌مون خیلی زیاد هستش. برای اینکه این مشکل رو برطرف کنیم، از مفهومی تحت عنوان **Vectorization** استفاده می‌کنیم. این مفهوم به این معنیه که به جای اینکه بیایم و توی کار با دیتامون، for بزنیم روی درایه‌ها، سعی کنیم عملیاتی که می‌خوایم انجام بدیم رو به شکل عملیات ماتریسی (ضرب و جمع ماتریسی و برداری، ضرب داخلی، ترانهاده کردن و اعمال توابع روی تک‌تک عناصر ماتریس‌ها) پیاده‌سازی کنیم.

این کار باعث میشه که زمان اجرای کد خیلی کمتر بشه. دلیلش اینه که عملیات‌های ماتریسی خیلی خوب می‌تونن موازی‌سازی بشن و به صورت چندهسته‌ای اجرا شن روی CPU، و همچنین پردازنده‌ها Instruction هایی مخصوص کار کردن با داده‌های بزرگ و برداری دارن که خیلی efficient تر اجرا می‌شن.

مرحله Feedforward الگوریتم رو، از همون اول به صورت Vectorized پیاده‌سازی کردیم. حالا توی این مرحله، باید Backpropagation رو هم Vectorized کنید. در پایان این مرحله انتظار میره که محاسبه‌ی مشتقات جزئی هر لایه (یعنی مشتقات نسبت به W و b و a ها) بدون for زدن انجام بشه.

برای مثال، کد پایین، برای محاسبه 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])
```

رو می‌تونید به صورت زیر بنویسید:

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

(علامت @ برای ضرب ماتریسی هستش).

یا محاسبه گرادیان برای نورون‌های لایه یکی‌مونده به آخر به شکل زیر هست:

```
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 میشه به صورت زیر Vectorized اش کرد:

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

سایر عبارات رو هم مشابه همین Vectorized کنید.

در پایان این مرحله، انتظار میره که کدتون توی زمان خیلی کمتری نسبت به مرحله‌ی قبل اجرا بشه. در نتیجه تعداد epoch ها رو افزایش بدید به عدد ۲۰۰ و دقت مدل نهایی و همچنین پلات Cost در طی زمان رو گزارش کنید.

قدم پنجم: تست کردن مدل

حالا که الگوریتم رو تا حد خوبی بهینه کردیم، می‌تونیم بریم و روی کل ۶۰,۰۰۰ عکس مجموعه Train، فرآیند یادگیری رو انجام بدیم. مقدار batch_size رو برابر با ۵۰، ضریب یادگیری رو برابر با ۱ و همچنین تعداد epoch ها رو ۵ در نظر بگیرید.

در پایان این قدم، دقت مدل رو برای مجموعه‌ی Train و همچنین برای مجموعه‌ی Test گزارش کنید. همچنین، همانند قبل میانگین Cost رو نیز پلات کنید.

اگر پیاده‌سازی‌ها درست انجام شده باشه، انتظار میره که دقت مدل برای Train و Test، حدود ۹۰ درصد باشه.

نکته: بسته به زبانی که باهاش پروژه رو پیاده‌سازی کردید، و قدرت سیستم‌تون، زمان اجرای فرآیند یادگیری متفاوت هستش. برای مثال با زبان پایتون و استفاده از NumPy، روی پردازنده Intel 7700HQ، حدود ۱ دقیقه زمان اجرای فرآیند یادگیری شد.

امتیازی‌ها

۱- فرآیند یادگیری مدل رو بر روی عکس‌های مجموعه Train رو بدون هیچ تغییری انجام بدید. سپس تمام عکس‌های مجموعه‌ی Test رو، ۴ پیکسل به سمت راست شیفت بدید؛ یعنی، ۴ ستون از سمت راست عکس حذف کرده و ۴ ستون تمام سیاه به سمت چپ اضافه کنید. دقت مدل را برای این مجموعه Test جدید گزارش کنید.
با وجود اینکه اعداد داخل عکس به همان شکل قبل، قابل مشاهده هستن و صرفا شیفت پیدا کردن، به نظرتون چرا دقت مدل در این حد کم میشه؟
(به این تغییرات اندک و عمدی در ورودی مدل‌ها که از نظر ما انسان‌ها، تغییر نامحسوسی عه، اما می‌تونه مدل رو کاملا به اشتباه بندازه، **Adversarial Attack** میگن. اگر دوست داشتید می‌تونید اینجا در موردش بیشتر بخونید.)

۲- توابع Activation گوناگونی وجود دارن. از Sigmoid و Tanh گرفته تا ReLU. شبکه عصبی خود رو با یه تابع Activation دیگه تست کنید و خروجی نهایی و سرعت یادگیری مدل رو با حالتی که از Sigmoid استفاده کردید، مقایسه کنید.

۳- روش Stochastic Gradient Descent به نوبه‌ی خودش یک‌سری مشکلات داره که باعث میشه سرعت همگرایی در فرآیند یادگیری کم باشه یا بعضا مدل ما در مینیمم محلی گیر بیافته. برای همین، خیلی اوقات از SGD خام استفاده نمیشه بلکه از نسخه‌های پیشرفته‌ترش استفاده میشه. توی این نوشتار، متدهای پیشرفته‌تر SGD بررسی شده اند. یکی از اون‌ها رو پیاده‌سازی کنید و تغییرات ایجاد شده در مدل و فرآیند یادگیری‌اش رو بررسی کنید.