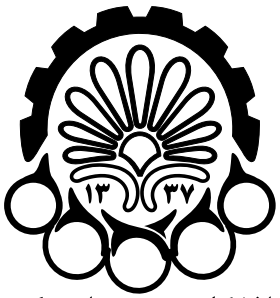
به نام خدا



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

مبانی هوش محاسباتی

تمرین پیاده‌سازی شبکه‌های عصبی

استاد درس:

دکتر عبادزاده

زمستان ۱۴۰۰

مقدمه

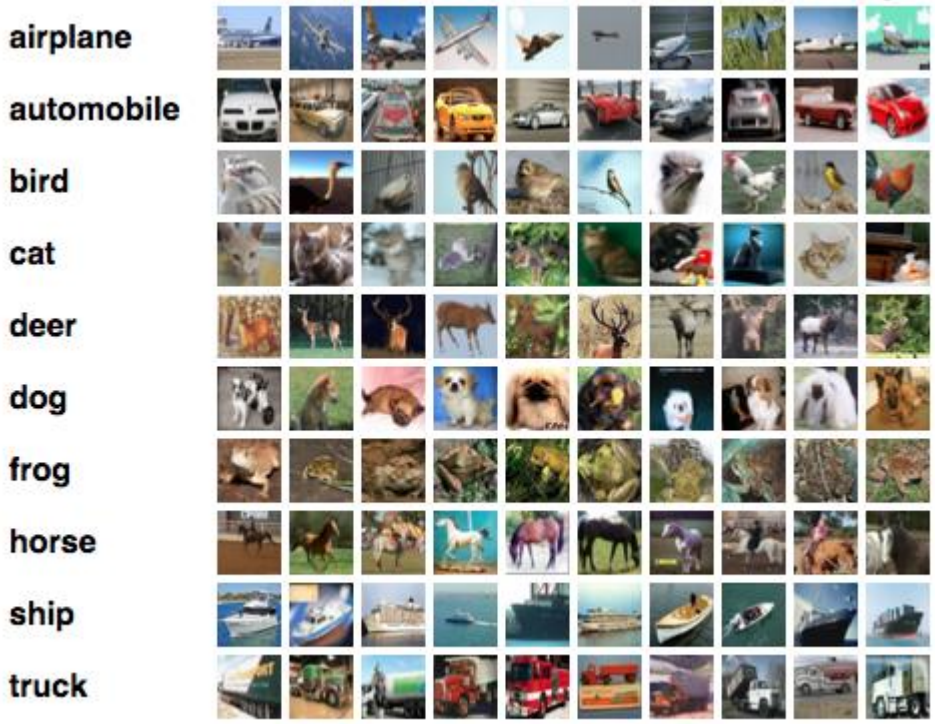
یکی از کاربردهای شبکه‌های عصبی، دسته‌بندی (classification) است. در این پروژه قصد داریم به سراغ دسته‌بندی تصاویر برویم. می‌خواهیم با استفاده از شبکه‌های عصبی مختلف (به طور خاص fully connected networks) مدلی بسازیم که عکس‌هایی را به عنوان ورودی گرفته و دسته‌بندی هر عکس رو تشخیص بدهد.

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

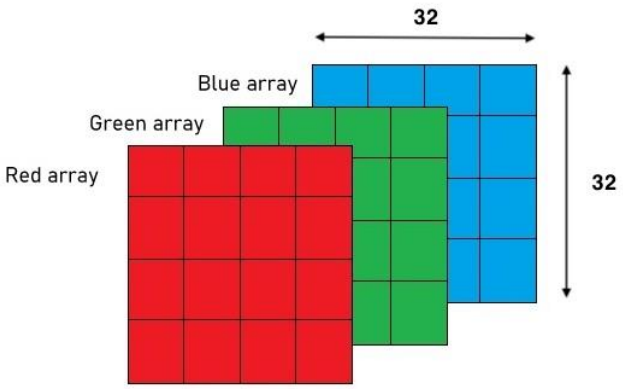
برای انجام این پروژه شما **باید** با استفاده از google colab و jupyter notebook مراحل را پیاده‌سازی و نتایج را گزارش کنید. به این منظور، در این لینک می‌توانید با اصول اولیه و کار با jupyter notebook آشنا شوید. همچنین در این لینک می‌توانید آشنایی ابتدایی از google colab بدست آورید.

شرح مساله

در این مسئله، می‌خواهیم تصاویر دیتاست CIFAR-10 را دسته‌بندی کنیم. این دیتاست، نسخه کوچک‌تری از دیتاست CIFAR-100 است که محدود به تصاویری از ۱۰ دسته مختلف است. دسته‌بندی تصاویر این دیتاست را در شکل زیر مشاهده می‌کنید.



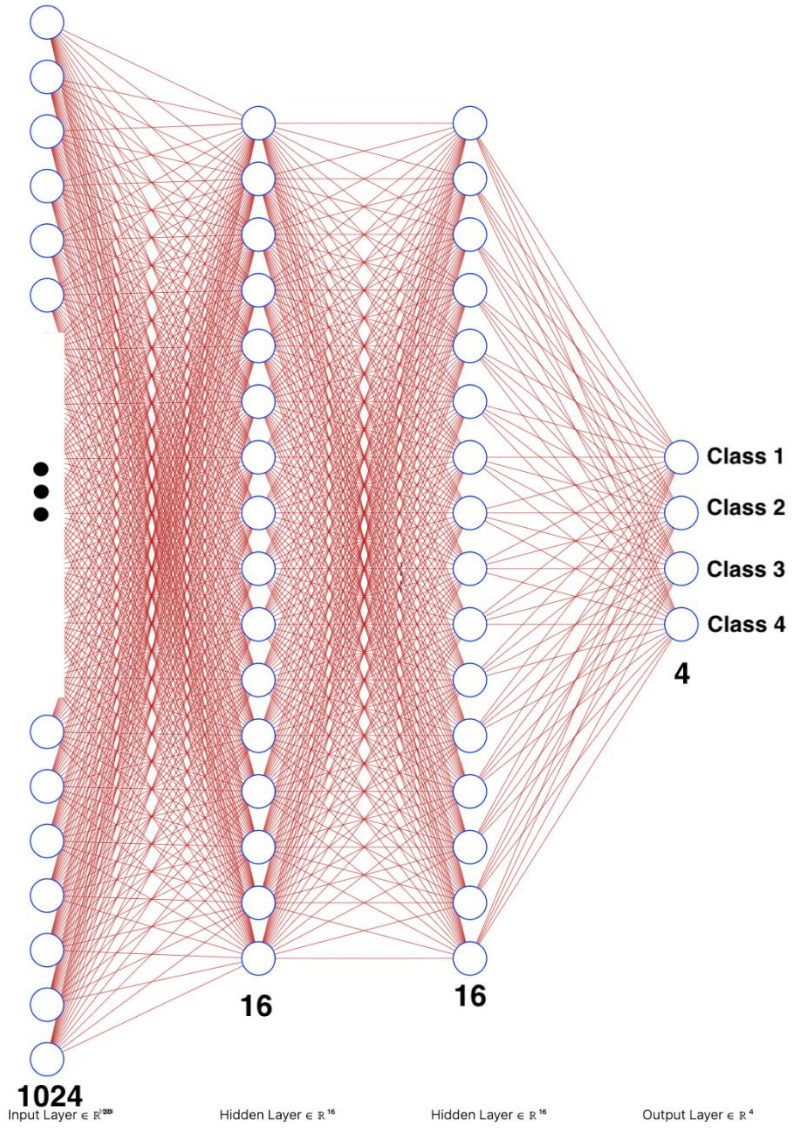
ابعاد تصاویر در این دیتاست برابر ۳۲ (طول) در ۳۲ (عرض) در ۳ (کانال‌های رنگی RGB) است.



یکی از مهم‌ترین قدم‌ها در آموزش مدل‌های هوش مصنوعی، آماده‌سازی و پیش‌پردازش داده‌ها، به طوری که در قالب مناسب ورودی دادن به مدل درآیند، است. در این خصوص در ادامه توضیح کامل‌تری داده می‌شود. در اینجا، ابتدا نیاز داریم که در یک گام پیش‌پردازش، تصاویر رنگی رو به خاکستری (grayscale)، یعنی تنها متشکل از یک کانال، تبدیل کنیم. پس از این گام تصاویری به ابعاد ۳۲ در ۳۲ خواهیم داشت. در نتیجه لایه‌ی ورودی شبکه دارای ۳۲×۳۲=۱۰۲۴ نورون هستش که هر نورون میزان روشنایی یک پیکسل رو به صورت یک عدد int از ۰ تا ۲۵۵ نشون می‌ده.

در اینجا برای کاهش حجم محاسباتی، تنها از تصاویر ۴ کلاس اول این دیتاست استفاده می‌کنیم. در نتیجه لایه‌ی خروجی ما شامل ۴ نورون خواهد بود. نورونی که بیش‌ترین مقدار activation رو داره، به عنوان دسته‌ی تشخیص داده شده توسط مدل ما، انتخاب می‌شه.

برای این شبکه‌ی عصبی، دو لایه‌ی پنهان (hidden layer) در نظر می‌گیریم که هر کدوم دارای ۱۶ نورون است.
پس ساختار شبکه‌ی عصبی ما به شکل زیر خواهد بود:



شبه‌کد

شبه‌کد فرآیند یادگیری شبکه عصبی ما طبق روش Stochastic Gradient Descent، به شکل 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، مصورسازی داده‌ها و پیش‌پردازش

در ابتدا نیاز است دیتاست CIFAR10 را از این لینک دانلود کنید. با استفاده از کتابخانه gdown، که به طور پیش‌فرض در colab نصب است، می‌توانید این فایل را به سادگی دریافت کنید. با وارد کردن سلول زیر در ژوپیتر، عمل دریافت و اکسترکت دیتاست برای شما انجام می‌شود.

```
!gdown --id 1Y1vgzPvMeVcXSxDfOlCVia7wsU7p8M6g -O CIFAR10.tar.gz
!tar xzf CIFAR10.tar.gz
```

پس از این کار، تصاویر دیتاست در پوشه CIFAR10 قابل دسترسی است. در این پوشه تصاویر به دو دسته train و test تقسیم شده‌اند. در بخش train، از هر دسته ۵ هزار تصویر است و مجموعا دیتاست از ۵۰ هزار تصویر برای آموزش تشکیل شده است. برای آشنایی بیشتر با دیتاست، با استفاده از کتابخانه matplotlib ده تصویر از کلاس‌های مختلف دیتاست همراه با برچسب (label) آنها نمایش دهید.

سپس مراحل زیر را به منظور آماده‌سازی و پیش‌پردازش روی داده‌ها انجام دهید:

۱- در هر دو مجموعه داده‌های آموزش و تست، ۴ دسته‌ی اول دیتاست شامل داده کلاس‌های airplane، automobile، bird و cat را در قالب یک آرایه numpy بخوانید (مثلا با تابع imread) و در قالب ماتریسی با ابعاد (n_samples, width, height, channels) ذخیره کنید. به عنوان مثال، برای داده‌های تست باید ماتریسی به ابعاد (4000, 32, 32, 3) داشته باشید. همچنین برچسب‌های این داده‌ها را نیز در ماتریس جداگانه‌ای به شکل one-hot با ابعاد (n_samples, n_classes) ذخیره کنید. برای داده‌های تست، ماتریس برچسب ابعادی برابر (4000, 4) خواهد داشت. در انتهای این مرحله، شما باید ۴ ماتریس شامل داده‌های train و test و برچسب‌های train و test داشته باشید.

۲- برای کاهش پیچیدگی محاسباتی، تصاویر را خاکستری کنید. اگر ماتریس‌های داده در مرحله قبل را به درستی ساخته باشید، تابع زیر تمام تصاویر دیتاست شما را به تصاویر خاکستری تبدیل می‌کند.

```
def rgb2gray(rgb):

    r, g, b = rgb[:, :, :, 0], rgb[:, :, :,1], rgb[:, :, :, 2]
    gray = 0.2989 * r + 0.5870 * g + 0.1140 * b

    return gray
```

۳- نرمال‌سازی: در شبکه‌های عصبی، به خصوص زمان استفاده از توابع فعال‌سازی مانند sigmoid و tanh)، بزرگ بودن مقادیر ورودی می‌تواند اندازه گرادیان و در نتیجه سرعت یادگیری را کاهش دهد. به همین منظور، با تقسیم داده‌ها به ۲۵۵، آنها را به بازه صفر تا یک ببرید.

۴- Flat کردن داده‌ها: در لایه ورودی شبکه ۱۰۲۴ نورون داریم پس نیاز است تا داده‌ی ورودی شبکه به شکل برداری با اندازه‌ی ۱۰۲۴ باشد. به این منظور می‌توانید بر روی ماتریس داده‌های خود دستور reshape(-1, 1024). را بزنید تا ابعاد ماتریس داده‌ها به (n_samples, 1024) تبدیل شود.

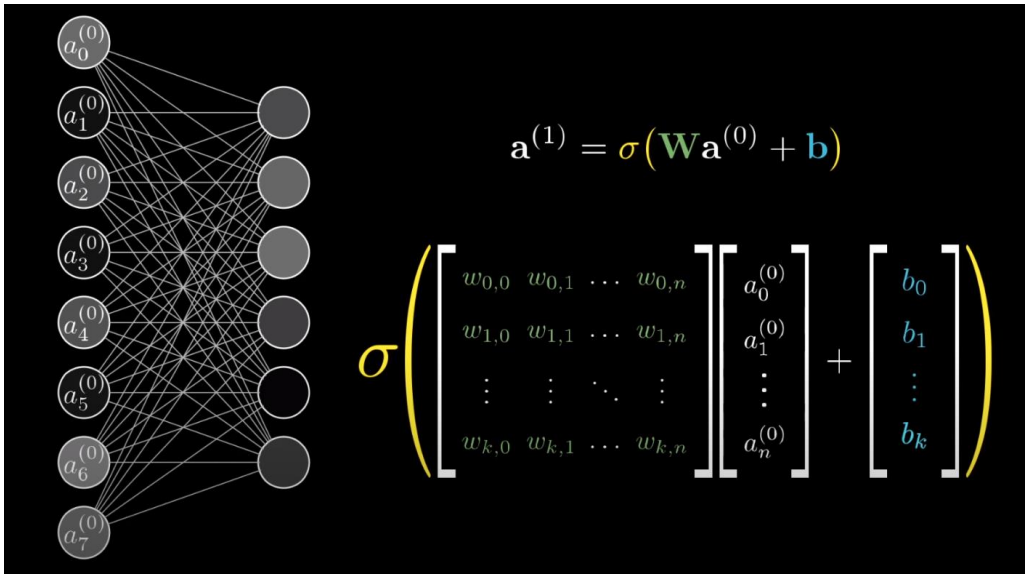
۵- شافل کردن داده‌ها: ماتریس‌های داده خود را پیش از شروع آموزش، بهم بریزید. توجه کنید همان ترتیب بهم ریختگی که در ماتریس داده اعمال می‌کنید، باید در ماتریس برچسب‌ها نیز اعمال شود. از آنجا که در ابتدای هر ایپاک از آموزش نیز باید این کار انجام شود، پیشنهاد می‌شود برای این کار یکی تابع جداگانه بنویسید.

قدم دوم: محاسبه خروجی (Feedforward)

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، ۲۰۰ داده (داده‌های ۲۰۰ 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 ۲۰۰ داده را محاسبه کنید. محاسبه خروجی را باید به طریقی که بالاتر گفتیم (یعنی به صورت ضرب و جمع ماتریسی/برداری و اعمال تابع سیگموئید) 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 دستی 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 ۲۵ 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:

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

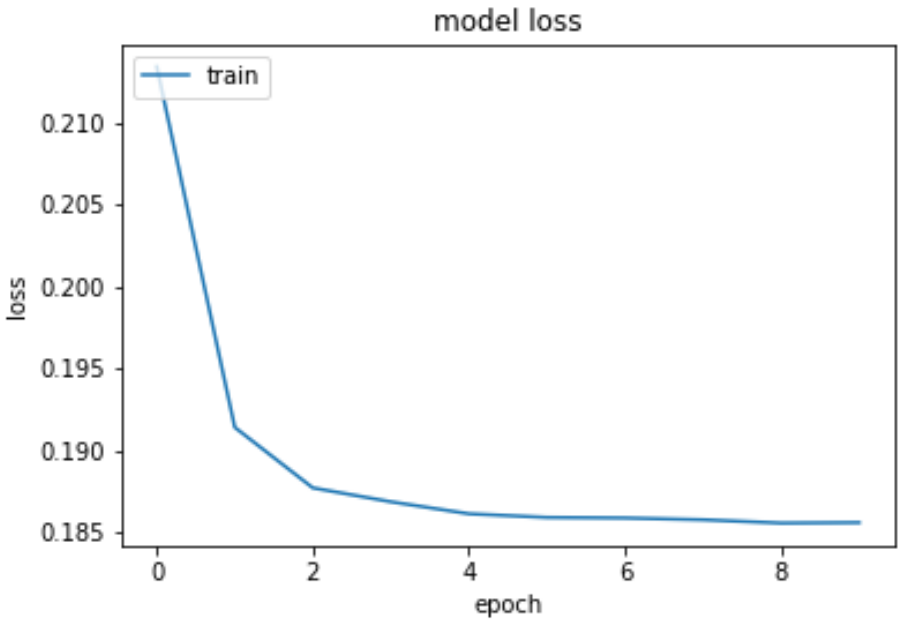
بدست آوردن این مشتق‌ها به کمک backpropagation انجام می‌شود.

در این قدم از پروژه، شبه‌کدی که بالاتر گفته شد را به طور کامل پیاده‌سازی کنید. مجموعه train را، همان ۲۰۰ داده که در مرحله‌ی قبل گفته شد، در نظر بگیرید. hyperparameterها را هم به صورت زیر قرار دهید: مقدار batch_size برابر با ۱۶، ضریب یادگیری برابر با ۰.۳ و تعداد epochها برابر با ۱۰.

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 ۲۰۰ داده گزارش کنید. با توجه به اینکه تعداد epochها و داده‌های آموزشی کم هستند، انتظار می‌رود ر پایان فرآیند یادگیری، دقت مدل در این حالت، به طور میانگین حدود ۳۰ درصد باشد. اگر زمان اجرا برایتان معقول بود می‌توانید به ازای تعداد epoch بیشتر هم، کدتان را تست کنید و نتایج را گزارش کنید. همانطور که می‌دانید مقداردهی اولیه شبکه که به طور تصادفی انجام می‌شود می‌تواند تا حدی روی فرآیند آموزش شبکه تاثیر بگذارد و دقت مدل شما کمی با ۳۰ درصد فاصله داشته باشد.

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 costها دیده نشود، به احتمال زیاد مشکلی در پیاده‌سازی الگوریتمتان وجود دارد.

قدم چهارم: Vectorization

تا اینجا تنها با ۲۰۰ 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 تک‌تک 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(4):
    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(4):
        grad_a2[k, 0] += w3[j, k] * sigmoid_deriv(z3[j, 0]) * (2 * a3[j,0]
- 2 * y[j, 0])
```

که به صورت 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 epochها را افزایش دهید به عدد ۲۰ و دقت مدل نهایی، زمان اجرای فرآیند یادگیری و همچنین پلات cost در طی زمان را گزارش کنید.

با توجه به اینکه سرعت اجرای کد شما به دلیل ماتریسی شدن عملیات‌ها افزایش یافته است، برای این بخش کدتان را به نحوه‌ای طراحی کنید که ۱۰ بار از اول کدتان اجرا شود و نتیجه‌ی نهایی را به صورت میانگین کل نتایج خروجی دهد.

قدم پنجم: تست کردن مدل

حال که الگوریتم را تا حد خوبی بهینه کرده‌ایم، می‌توانیم بر روی کل داده‌های ۴ کلاس (جمعا ۸۰۰۰ داده) train را انجام دهیم. مقدار batch_size برابر با ۱۶، ضریب یادگیری برابر با ۰.۳ و تعداد epochها برابر با ۴۰ قرار دهید.

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. برای این قدم نیز مشابه حالت قبل کدتان باید نتایج ۱۰ اجرا را به صورت میانگین خروجی دهد.

اگر پیاده‌سازی درست انجام شده باشد، انتظار می‌رود که دقت مدل برای train و test حدود ۵۰ تا ۵۵ درصد باشد.

امتیازی‌ها

در بخش اصلی دیدیم چطور می‌توان یک شبکه عصبی fully connected پیاده سازی کرد اما با وجود ۱۰ کلاس مختلف برای طبقه‌بندی کردن عملکرد این شبکه قابل قبول نبود.

شبکه های عصبی پیچشی (convolutional neural network) دقیقا مانند سیستم پردازش تصویر در موجودات زنده عمل میکنند. این نوع شبکه ها برخلاف تاریخچه قدیمی به دلیل ضعف در توانایی سخت‌افزار برای پیاده‌سازی آن‌ها، تا چند سال گذشته مورد استفاده قرار نمی‌گرفتند.

اما در جریان رویداد ImageNet Large Scale Recognition Challenge این نوع شبکه توجه همگان را جلب کرد و توانست با خطای ۱۶ درصدی ۱۰۰۰ کلاس مختلف را طبقه بندی کند در حالی که بهترین روش به خطای ۲۶ درصد رسیده بود.

اکنون میخواهیم از یک شبکه‌ی عصبی پیچشی بر روی مجموعه‌داده CIFAR-10 استفاده کنیم و نتایج را با شبکه‌ایی که در بخش اصلی پروژه پیاده سازی کردید مقایسه کنیم. برای این کار ابتدا باید با فریمورک tensorflow آشنا شویم. بدین منظور یک ویدیو آموزشی در سامانه کورسز برای شما قرار گرفته که در آن روی همین دیتاست یک شبکه عصبی پیچشی پیاده‌سازی می‌شود لذا قبل از شروع این بخش ویدیو مورد نظر را مشاهده کنید.

مراحلی که باید انجام دهید:

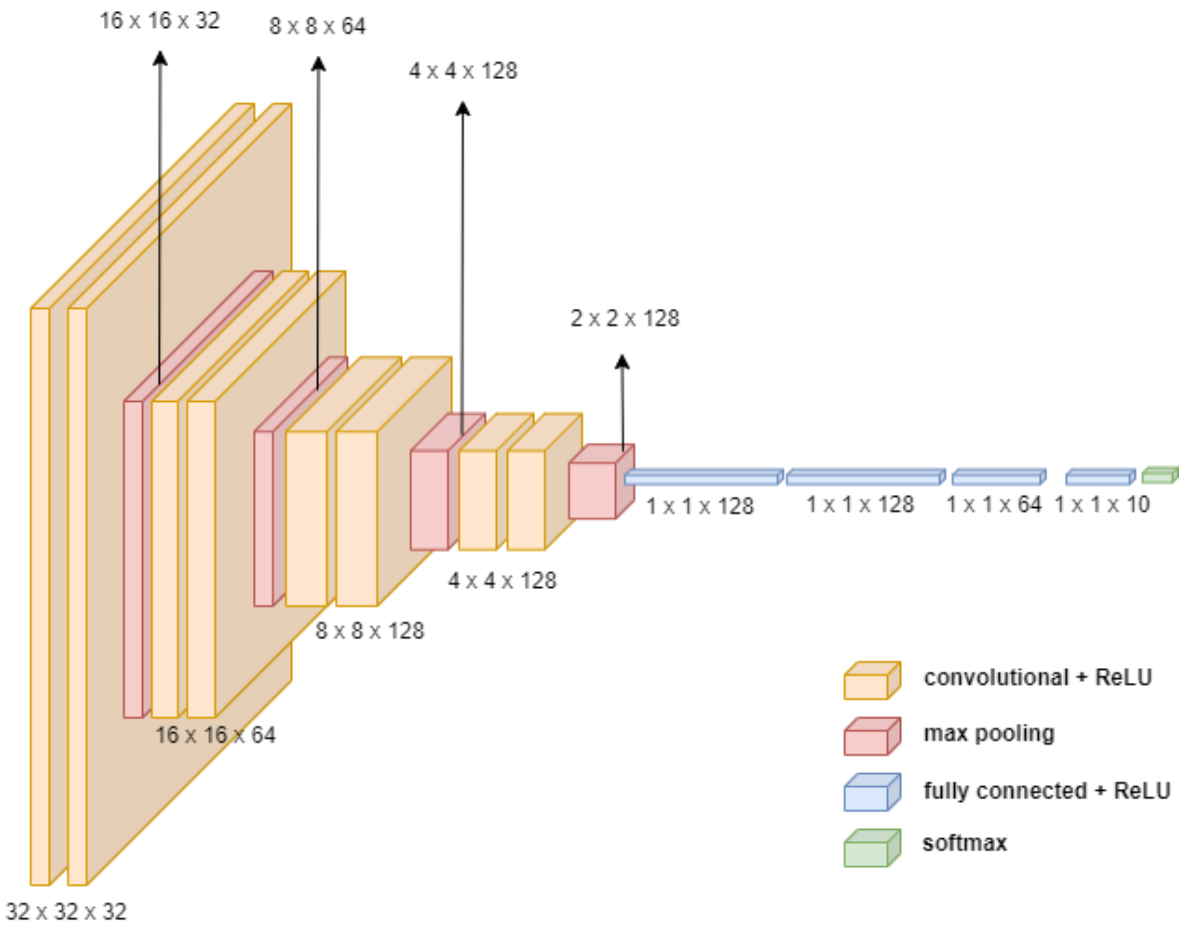
بخش اول:

- در این مرحله می‌خواهیم بر روی کل داده‌های دیتاست اصلی آموزش و تست انجام دهیم. به منظور راحتی کار با دیتاست CIFAR10، در این مرحله می‌توانید با دستور زیر این دیتاست را به راحتی از کتابخانه keras دریافت کنید.

```
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
```

توجه: برای گزارش نتایج مراحل زیر، خطا، دقت و ماتریس آشفتگی را برای داده‌ی تست محاسبه کنید و گزارش دهید.

۱- معماری زیر را پیاده سازی کنید و نتیجه را گزارش دهید. برای لایه‌های کانولوشنی اندازه کرنل را برابر ۳ بگیرید و padding را در حالت same قرار دهید.



۲- در مورد batch normalization تحقیق کنید و توضیح دهید، سپس این لایه ها را در جای مناسب به شبکه اضافه کنید. نتایج را گزارش کنید و نتیجه جدید را با نتایج قبلی مقایسه کنید.

۳- در مورد drop out تحقیق کنید و توضیح دهید، سپس این لایه ها را در جای مناسب به شبکه‌ی بدست آمده از مرحله قبل اضافه کنید. نتایج را گزارش کنید و نتیجه جدید را با نتایج قبلی مقایسه کنید.

۴- در مورد انواع optimizer های مختلف که از طریق keras در دسترس هستند تحقیق کنید و توضیح دهید. سپس آن‌ها را بر روی شبکه بدست آمده از مرحله قبل اعمال کنید و با یکدیگر مقایسه کنید و در نهایت بهترین آن‌ها را انتخاب کنید (اعمال ۳ optimizer مختلف کافیست).

۵- در مورد معیار های ارزیابی precision، recall و f1 تحقیق کنید و نتایج را گزارش دهید. سپس برای بهترین شبکه‌ی بدست آمده از مرحله قبل این معیارها را محاسبه کنید و مقادیر بدست آمده را تحلیل کنید.

توجه: در پایان این مرحله باید به دقت بالای ۸۰ درصد رسیده باشید و نتایج بدست آمده از معیارهای ارزیابی مختلف و confusion matrix قابل قبول باشد.

بخش دوم:

از جمله مشکلاتی که در بسیاری از پروژه‌های مربوط به هوش مصنوعی و علم داده وجود دارد محدود بودن دیتاست و یا نامتوازن بودن تعداد داده‌ها در هر کلاس است. این مشکلات سبب ایجاد اختلال در عملکرد شبکه می‌شوند. یکی از روش‌های حل این مشکل استفاده از data augmentation است که در این بخش با آن آشنا می‌شوید.

۱- در مورد data augmentation تحقیق کنید و توضیح دهید که نحوه عملکرد این روش چگونه است و تبدیل هایی که در آن استفاده میشود را شرح دهید. آیا از این روش برای داده های تست استفاده می‌شود؟ علت را شرح دهید.

۲- با استفاده از چند تبدیل که در بخش قبل یاد گرفتید از یکی از تصاویر موجود در دیتاست ۱۰ نمونه مصنوعی ایجاد کنید و آن ها را به همراه تصویر اصلی نمایش دهید (اعمال ۴ یا ۵ تبدیل مختلف کافی است).

۳- از کلاس های گربه و سگ در داده های آموزش ۹۰ درصد را حذف کنید (یعنی ۴۵۰۰ عکس از کلاس گربه و ۴۵۰۰ عکس از کلاس سگ). سپس داده های جدید را با بهترین مدلی که در بخش اول بدست آمده آموزش دهید. سپس برای نتایج بدست آمده confusion matrix را نمایش دهید. چه اختلالی در عملکرد شبکه رخ داده است؟ آن را تحلیل کنید.

آنچه باید ارسال کنید:

فایل jupyter notebook با فرمت ipnyb. را از google colab دانلود کنید (توجه کنید که خروجی هر سلول باید در فایل ارسالی مشخص باشد) و در قالب فایل زیپ همراه با فایل pdf پاسخ سوالات تشریحی با فرمت زیر در سامانه کورسز اپلود کنید:

CI_{student_id}_nnproject

دقت کنید که پروژه تحویل آنلاین **نخواهد** داشت پس حتما باید گزارش کاملی در فایل pdf ارسالی قبل از پاسخ‌گویی به سوالات تشریحی قرار بدید. همچنین می‌توانید گزارش و پاسخ سوالات تشریحی را در قالب markdown در فایل ipynb در کنار کدهای خود تحویل دهید توجه کنید باید کامنت های مناسب و کاملی در کد ارسالی نوشته باشید.

موفق باشید

تیم تدریس یاری درس مبانی هوش محاسباتی