پروژه

***حتما قبل از نوشتن گزارش‌های مربوط به بخش ۱ و ۲، قسمت انتهایی این مستند که مربوط به نکاتی درباره‌ی گزارش شماست را مطالعه کنید.**

معرفی پروژه

هدف این پروژه استفاده از شبکه‌های عصبی، توسعه‌های غیرخطی و چندلایه از پرسپترون خطی است، تا تصاویر را به چهار دسته شناسایی کند: کشتی، خودرو، سگ یا قورباغه. به عبارت دیگر، هدف نهایی شما ایجاد یک طبقه‌بند است که بتواند تشخیص دهد هر تصویر چه چیز را نشان می‌دهد.

در قسمت اول، یک شبکه عصبی کم‌عمق به سبک دهه ۱۹۸۰ ایجاد خواهید کرد. در قسمت دوم، این شبکه را با استفاده از تکنیک‌های مدرن تری مانند تغییر تابع فعال‌سازی، تغییر ساختار شبکه یا تغییر جزئیات دیگر افزایش خواهید داد.

شما برای اجرای این مدل‌ها از کتابخانه‌های PyTorch و NumPy استفاده خواهید کرد. کتابخانه PyTorch بیشتر کارهای سنگین را بر عهده خواهد گرفت، اما هنوز هم به شما وابسته است که دستورالعمل‌های سطح بالا مناسب را برای آموزش مدل پیاده‌سازی کنید.

پیشنهاد می‌شود منابع و مستندهای مربوط به PyTorch را سرچ و مطالعه کنید. مواردی از مستندات مفید در زیر آمده است:

https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html

https://machinelearningmastery.com/pytorch-tutorial-develop-deep-learning-models

کدها

کدهای لازم برای شروع این پروژه در پوشه‌ی همراه با این مستند؛ ضمیمه شده است. شما می‌توانید با تغییرات در فایل‌های neuralnet_part1.py و neuralnet_part2.py به بخش ۱ و ۲ این پروژه پاسخ دهید. فایل neuralnet_leaderboard.py مربوط به بخش ۳ام است که اختیاری و بدون نمره در نظر گرفته شده و برای تست پارامترها یا معماری‌هایی بیشتر از آنچه در بخش ۱ و ۲ گفته شده در نظر گرفته شده است.

کدها را قبل شروع پروژه، حتما بررسی کنید. همینطور می‌توانید با اجرای دستور `python3 mp5.py -h` در ترمینال خود، اطلاعات بیشتری را دریافت کنید.

پ.ن.: شما باید از کتابخانه‌های torch و numpy استفاده کنید. به جز این دو، فقط از ماژول‌های استاندارد زبان برنامه‌نویسی پایتون استفاده کنید. از torchvision استفاده نکنید.

مجموعه داده

مجموعه داده از ۳۰۰۰ تصویر رنگی با ابعاد 3 *31 * 31 (3 به خاطر اینکه تصاویر RGB هستند) تشکیل شده است (یک زیرمجموعه اصلاح‌شده از مجموعه داده CIFAR-10 که توسط الکس کریژهفسکی ارائه شده است). این مجموعه برای شما به ۲۲۵۰ نمونه آموزشی (که اغلب یک نمونه متوازن از اتومبیل‌ها، قایق‌ها، قورباغه‌ها و سگ‌ها هستند) و ۷۵۰ نمونه توسعه(development) تقسیم شده است.

تابع load_dataset() در فایل reader.py می‌تواند فایل مجموعه داده‌ها را unpack کرده و تصاویر و برچسب‌ها را برای مجموعه‌های آموزش و توسعه را بخواند. هر یک از این موارد یک Tensor هستند.

آموزش و dataloaders

دو ورودی اصلی به تابع اصلی آموزش شما (fit) تعداد دوره‌ها (epochs) و اندازه دسته (batch size) هستند. شما باید فرآیند آموزش خود را برای همان تعداد دوره‌ها روی مجموعه آموزش اجرا کنید. در هر دوره، کد شما باید تمام داده‌های آموزشی را دسته به دسته پردازش کند، هر دسته دارای اندازه مشخص شده است. اگر سعی کنید این کار را به صورت دستی انجام دهید، در مواردی که اندازه‌ی مجموعه‌ی داده‌ها مضربی از اندازه‌ی دسته نباشد، دچار مشکل می‌شوید، چون در این حالت همه دسته‌ها اندازه‌ی یکسان خواهند داشت، غیر از دسته‌ی آخر که اندازه‌ی آن کمتر از بقیه خواهد بود.

برای ساده‌تر کردن این موضوع، از یک dataloader در PyTorch استفاده خواهید کرد. Dataloader یک راه برای مدیریت بارگذاری و تبدیل داده‌ها قبل از ورود به مدل برای آموزش یا پیش‌بینی است. این به شما اجازه می‌دهد که کدی بنویسید که به نظر می‌رسد مثل این است که شما فقط در حال حلقه زدن از مجموعه داده هستید، و تقسیم به دسته‌ها به صورت خودکار انجام می‌شود. جزئیات در مورد استفاده از dataloader را می‌توانید در این آموزش توسط Shervine Amidi و Afshine Amidi پیدا کنید. ما یک تابع کمکی (get_dataset_from_arrays(X,Y) در utils.py فراهم کرده‌ایم که تنسورهای ویژگی (X) و برچسب‌ها (Y) را به یک کلاس ساده دیتاست PyTorch تبدیل می‌کند که به راحتی می‌توان آن را برای dataloaders خود بارگذاری کرد. Dataloader یک دیکشنری را برمی‌گرداند، نه یک تاپل. شما باید برچسب‌ها و ویژگی‌ها را از آن دیکشنری بیرون بیاورید.

ماتریس درهم‌ریختگی(confusion matrix)

برنامه سطح بالای mp5.py سه نوع بازخورد در مورد مدل شما ارائه می‌دهد.

۱. دقت روی مجموعه توسعه (dev set).

۲. ماتریس ابهام مربوط به مجموعه توسعه.

۳. تعداد کل پارامترهای شبکه شما.

ماتریس ابهام ابزار بسیار مفیدی برای ارزیابی مسائل دسته‌بندی چندکلاسه است، زیرا به شناسایی منابع ممکن از عدم تعادل در مجموعه داده(imbalance in your dataset) شما کمک می‌کند و ممکن است راهنمایی ارزشمندی در مورد احتمال بایاس موجود در روند آموزش شما ارائه دهد.

به طور خاص، در یک مسئله دسته‌بندی با k کلاس، یک ماتریس ابهام k سطر و k ستون دارد. هر سطر مربوط به برچسب داده‌های واقعی است و هر ستون به برچسب پیش‌بینی شده توسط دسته‌بند شما اشاره دارد. هر ورودی در این ماتریس شامل تعداد تاپل متناظر (برچسب حقیقی، برچسب پیش‌بینی شده) است. به عبارت دیگر، تمام عناصر در قطر این ماتریس مربوط به دسته‌بندی صحیح هستند و تمام عناصر دیگر به عنوان اشتباهات محسوب می‌شوند. جزئیات بیشتر را می‌توانید در این صفحه ویکی‌پدیا بیابید.

بخش ۱: شبکه عصبی کم‌عمق کلاسیک

یک شبکه عصبی پایه و کلاسیک از یک دنباله از لایه‌های پنهان که یک لایه ورودی قبل آن‌ها و یک لایه خروجی بعد آن‌ها دارند، تشکیل شده است. ورودی از لایه ورودی به یک یا چند لایه پنهان وارد می‌شود و داده از طریق لایه یا لایه‌های پنهان عبور می‌کند و به لایه خروجی می‌رود. هر شبکه عصبی یک تابع ایجاد می‌کند که با انتقال داده‌ها از لایه‌ی ورودی تا لایه‌ی خروجی ایجاد می‌شود. در اسلایدها و تمارین قبلی با جزییات این تابع آشنا شدید. برای یادآوری اینطور در نظر بگیرید که مدل ما تابع F_W را یاد می‌گیرد و پیشنهاد می‌دهد.

اگر ماتریس وزن‌های لایه‌ی پنهان و لایه‌ی خروجی را به ترتیب W_1 و W_2 و بایاس‌های این دولایه را نیز به ترتیب b_1 و b_2در نظر بگیریم، ابعاد هر یک به این صورت است:

$$W_1 \in R^{h \times d}, W_2 \in R^{h \times 4}, b_1 \in R^{h}, b_2 \in R^{4}$$

که d ابعاد ورودی(تعداد فیچرها)، h ابعاد لایه‌ی پنهان(تعداد نرون‌های لایه‌ی پنهان) و ۴ هم ابعاد خروجی(تعداد کلاس‌ها) هستند. در نهایت مدل ما باید تابع F_W را به صورت زیر یاد بگیرد:

$$F_W(x) = W_2\sigma(W_1 x + b_1) + b_2$$

که σ تابع فعال‌سازی است که شما می‌توانید از یکی از توابع فعال‌سازی sigmoid یا ReLU استفاده کنید.

برای این مجموعه داده، شما 2883 فیچر ورودی دارید. چون هر عکس 31*31 است و 3 چنل رنگی هم دارد پس:

$$31*31*3 = 2883$$

شما باید قادر به نوشتن و اجرای کد بخش ۱ با حداکثر ۲۰۰ نرون در یک لایه‌ی پنهان باشید.

آموزش و توسعه (train and development)

یادگیری(training):در این پروژه، می‌توانید از تابع خطا cross entropy استفاده کنید. شما نیازی به پیاده‌سازی این توابع ندارید و می‌توانید از توابع داخلی PyTorch استفاده کنید. به لینک‌های زیر رجوع کنید:

https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

https://pytorch.org/docs/stable/generated/torch.nn.functional.cross_entropy.html

نکته: در لینک اول، تابع cross entropy را به صورت یک کلاس ایمپورت می‌کنید و باید ابتدا یک آبجکت از آن بسازید و سپس از آن استفاده کنید:

```
import torch.nn as nn
criterion = nn.CrossEntropyLoss()
loss = criterion(predicted_labels, true_labels)
```

اما در لینک دوم تابع cross entropy را به صورت یک تابع ایمپورت می‌کنید و مستقیم از آن استفاده می‌کنید:

```
import torch.nn.functional as F
loss = F.cross_entropy(predicted_labels, true_labels)
```

توجه داشته باشید که به دلیل اینکه CrossEntropyLoss در PyTorch شامل یک تابع سیگموئید است، نیازی به اضافه کردن یک تابع فعال‌سازی به لایه آخر شبکه ندارید.

توسعه(development): پس از آموزش مدل شبکه عصبی، باید مجموعه‌ی ارزیابی یا همان توسعه(dev یا validation، هر دو به یک معنا هستند) را به مدل بدهیم تا مدل تصمیم بگیرد که هر تصویر متعلق به کدام کلاس است. این کار با ماکسیمم گرفتن(argmax) از ۴ نرون لایه‌ی خروجی، انجام می‌شود.

استانداردسازی داده(Data Standardization): سرعت همگرایی و دقت‌ها می‌توانند با ساده‌سازی داده‌های ورودی توسط کاهش میانگین نمونه و تقسیم بر انحراف معیار نمونه به طور قابل توجهی افزایش یابند. به صورت دقیق‌تر، شما می‌توانید ماتریس داده X

را با تنظیم $X := (X-\mu) / \sigma$

تغییر دهید. توجه داشته باشید که شما یک مقدار ویژگی را در سراسر تصاویر استاندارد می‌کنید، نه استاندارد کردن یک مقدار ویژگی نسبت به ویژگی‌های دیگر در همان تصویر. این استانداردسازی باید در تابع fit() انجام شود، نه در تابع forward(). این کار را در ابتدا انجام دهید!

با طراحی مدل و نکات فوق، انتظار دارید **دقت حدود ۰.۶۲** برای مجموعه توسعه داشته باشید.

همچنین برای خوانایی بیشتر خروجی پیشنهاد می‌شود از tqdm استفاده کنید.

بررسی کد اولیه

فایل‌های neuralnet_part1.py، neuralnet_part2.py و neuralnet_leaderboard.py یک کلاس NeuralNet ارائه می‌دهند که یک torch.nn.module را پیاده‌سازی می‌کند. این کلاس شامل توابع __init__()، forward() و step() است. تابع اصلی fit() از این توابع برای آموزش شبکه و سپس دسته‌بندی تصاویر از مجموعه تست/توسعه استفاده خواهد کرد.

__init__()

در __init__() باید ساختار شبکه را بسازید. دو راه برای انجام این کار وجود دارد:

۱. استفاده از آبجکت‌های Linear و Sequential. به یاد داشته باشید که Linear از یک مقداردهی اولیه یکنواخت Kaiming He برای ماتریس‌های وزن استفاده می‌کند و اصطلاحاً اعضای تعیین شده را به همه صفرها می‌نشاند.(گزینه‌ی پشنهادی که ساده‌تر و بهینه‌تر است.)

۲. به طور جایگزین، می‌توانید به صورت صریح ماتریس‌های وزن W1، W2،... و اصطلاحات تعیین b1، b2،... را با تعریف آن‌ها به عنوان Tensorها، تعریف کنید. این رویکرد دستی بیشتری است و به شما امکان می‌دهد تا مقداردهی اولیه خود را انتخاب کنید. با این حال، برای این تکلیف، مقداردهی اولیه یکنواخت Kaiming He کافی بوده و یک انتخاب خوب است.

علاوه بر این، در این تابع باید یک آبجکت بهینه‌ساز(optimizer) را نیز مقداردهی اولیه کنید تا در تابع step() برای بهینه‌سازی شبکه استفاده شود. انتخاب optimizer به عهده‌ی خودتان است. می‌توانید گزینه‌های مختلف را امتحان کنید. توضیحات مربوط به هر موردی که استفاده کردید را به طور خلاصه در گزارش خود بنویسید.

forward()

تابع forward() باید یک forward pass از شبکه را انجام دهد. این کار می‌تواند با فقط فراخوانی آبجکت Sequential که در __init__() تعریف شده است یا (اگر ترجیح می‌دهید که به صورت صریح ماتریس‌های وزن را تعریف کنید) با ضرب در ماتریس‌های وزن با داده‌های خود انجام شود.

step()

تابع step() باید به‌روزرسانی گرادیان از یک دسته از داده‌های آموزش (نه کل مجموعه داده آموزشی) را انجام دهد. این را می‌توانید با فراخوانی loss_fn(yhat,y).backward() و سپس به‌روزرسانی مستقیم وزن‌ها خودتان انجام دهید، یا می‌توانید از یک آبجکت بهینه‌سازی استفاده کنید که ممکن است در __init__() مقداردهی اولیه شده باشد تا به شما در به‌روزرسانی شبکه کمک کند. حتماً از تابع zero_grad() روی بهینه‌ساز خود فراخوانی کنید تا بافر گرادیان از فراخوانی قبلی پاک شود.

زمانی که از این تابع مقدار loss_value را بر می‌گردانید، اطمینان حاصل کنید که آن را به یک عدد ساده تبدیل کنید. این امکان می‌دهد که جمع‌آوری زباله(garbage collection) به درستی انجام شود تا برنامه شما مقدار حافظه زیادی را به خود اختصاص ندهد. دو گزینه برای این کار:

۱. return loss_value.item(): این روش وقتی کار می‌کند که مقدار `loss_value` فقط یک عدد باشد.

۲. یا از loss_value.detach().cpu().numpy() استفاده کنید: این روش باعث جدا شدن مقدار loss از محاسباتی که به آن منجر شد، می‌شود(این محاسبات در آبجکت تنسور نگه‌داری می‌شوند و موقع محاسبه‌ی گرادیان به کار می‌روند)، آن را به CPU منتقل می‌کند (برای مثال، اگر محلی از GPU استفاده می‌کنید)، و سپس آن را به یک آرایه NumPy تبدیل می‌کند.

fit()

تابع fit() باید یک شیء NeuralNet بسازد و به طور تکراری تابع step() شبکه عصبی را فراخوانی کند تا شبکه را آموزش دهد. سپس fit() باید شبکه عصبی را روی مجموعه توسعه اجرا کرده و ۳ مورد را برگرداند: یک لیست از خطاها برای هر دوره آموزش، یک آرایه NumPy با برچسب‌های کلاس تقریبی (۰، ۱، ۲ یا ۳) برای مجموعه توسعه و شبکه آموزش دیده NeuralNet.

بخش ۲: شبکه مدرن

در این بخش، سعی می‌کنید عملکرد خود را با استفاده از تکنیک‌های مدرن یادگیری ماشین بهبود بخشید. این تکنیک‌ها شامل امکانات زیر است:

۱. انتخاب تابع فعال‌سازی: برخی از گزینه‌های ممکن شامل Tanh، ELU، Softplus و LeakyReLU هستند. هر یک را امتحان کنید و خواهید دید که انتخاب صحیح تابع فعال‌سازی می‌تواند منجر به همگرایی به طور قابل توجه سریعتر، بهبود عملکرد کلی، یا حتی هر دو شود.

۲. رگولاریزاسیون L۲: رگولاریزاسیون زمانی استفاده می‌شود که می‌خواهید توانایی مدل خود را در تعمیم به نمونه‌های ناشناخته افزایش دهید. یک شکل رایج از رگولاریزاسیون، رگولاریزاسیون L2 است. می‌توانید رگولاریزاسیون L2 را با اضافه کردن یک عبارت اضافه پیاده‌سازی کنید که norm وزن‌ها را جریمه کند. راجع به این روش تحقیق کنید.**(این مورد از بین ۴ مورد نام برده‌شده، اختیاری‌ست. اتجام آن نمره‌ای هم ندارد!)**

۳. عمق و عرض شبکه: شبکه‌ای که در بخش ۱ پیاده‌سازی کردید، یک شبکه دولایه است چرا که از دو ماتریس وزن استفاده می‌کند. گاهی اوقات افزودن لایه‌های پنهان بیشتر یا اضافه کردن ماتریس‌های وزن بیشتر برای به دست آوردن توانایی نمایش بیشتر و آموزش آسان‌تر ممکن است به بهبود عملکرد کمک کند.

۴. استفاده از شبکه‌های عصبی کانوولوشن (CNN): هرچند امکان دارد با شبکه‌های چندلایه سنتی نتایج خوبی حاصل شود، اما در تسک‌هایی که تصاویر را دسته‌بندی می‌کنند، اغلب بهتر است از شبکه‌های عصبی کانوولوشن استفاده کنید که به طور خاص برای وظایف پردازش سیگنال مانند تشخیص تصویر طراحی شده‌اند. می‌توانید نتایج مدل خود را با استفاده از لایه‌های کانوولوشن در شبکه بهبود بخشید. توجه کنید: ورودی به لایه CNN باید ۴بعدی باشد:

(batch_size, channel_num, height, width)

نکته: از آنجایی که ورودی شبکه بخش اول به صورت ۲بعدی است، قبل از feedکردن ورودی به لایه‌های کانوولوشن، باید آن را reshape کنید و ابعاد آن را طبق همان ابعادی که در بالا نوشته شده، تغییر دهید.

سعی کنید حداکثر ۵۰۰،۰۰۰ پارامتر کلی را استفاده کنید. این به این معناست که اگر هر مقدار اعشاری در همه وزن‌های شما شامل مقادیر بایاس باشد (به عنوان مثال توسط این تابع کمکی پایتورچ برگردانده شده)، شما تنها حداکثر از ۵۰۰،۰۰۰ مقدار اعشاری استفاده می‌کنید.

با اعمال تغییرات ذکر شده، شما می‌توانید به دقتی **حدود ۰.۷۹** در مجموعه‌ی توسعه برسید.

نکات مربوط به گزارش‌ها

۱. از مراحل و پارامترها و اجراهای مختلف خود گزارش بنویسید.

۲. برای epoch number، batch size و learning rate، اعداد مختلفی را امتحان کنید(هر کدام حدود ۴ عدد مختلف، مثلا برای batch size می‌توانید ۱۶، ۳۲، ۶۴ و ۱۲۸ را امتحان کنید) و اجرا بگیرید و نتایج را در جداولی برای هر یک از این پارامترها گزارش کنید.

۳. اگر فرضیاتی کردید که در جزییات این مستند نبود و یا فرضیات شما با بخش‌هایی از این مستند متفاوت بود با ذکر توضیح و دلیل، این فرضیات را بنویسید.

۴. دقت کنید که در گزارش خود چگونگی کار کدهایی که می‌نویسید را توضیح ندهید و فقط چرایی هر بخش را توضیح دهید؛ به عبارت دیگر توضیح خط به خط کد نمره‌ای ندارد بلکه باید مفهوم را بیان کنید. توضیحات و تعاریف روش‌ها و متدهای مختلف موجود در پروژه اگر کپی‌شده از اینترنت باشد، نمره‌ای ندارد.