تمرین عملی اول درس مبانی هوش محاسباتی

نگار موقتیان، ۹۸۳۱۰۶۲

قدم دوم: محاسبهٔ خروجی

دقت (accuracy) مدل که معادل است با تعداد عکس‌هایی که به درستی تشخیص داده شده تقسیم بر تعداد کل عکس‌ها، را گزارش کنید.

دقت این مدل پیش از شروع فرآیند یادگیری در شکل زیر آمده‌است و برابر است با ۱۱ درصد (با توجه به اینکه وزن‌ها در ابتدا به صورت رندوم مقداردهی اولیه می‌شوند این مقدار به ازای هر بار اجرای برنامه می‌تواند متفاوت باشد).

▾ Compute the accuracy of the model before being trained

```
accuracy = 0

for i in range(100):
  x = train_set[i][0]
  label = np.argmax(train_set[i][1])

  prediction, _ = feed_forward(x)

  accuracy += 1 if (prediction == label) else 0

print(f"Accuracy: {accuracy}%")
```

```
Accuracy: 11%
```

قدم سوم: پیاده‌سازی BackPropagation

دقت مدل و زمان اجرای فرآیند یادگیری را برای همان 100 داده گزارش کنید.

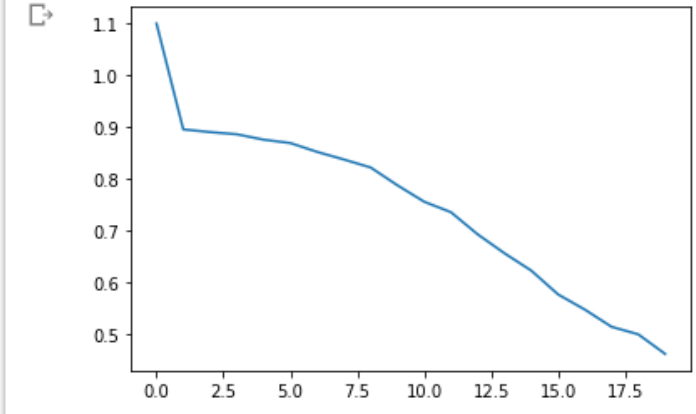
در شکل زیر دقت مدل برای ۱۰۰ داده و با استفاده از تابع "compute_grads" (که از vectorization استفاده نمی‌کند) آمده‌است.

```
epoch #1  |  cost: 1.10021  |  accuracy: 8.00%
epoch #2  |  cost: 0.89515  |  accuracy: 16.00%
epoch #3  |  cost: 0.88995  |  accuracy: 19.00%
epoch #4  |  cost: 0.88570  |  accuracy: 19.00%
epoch #5  |  cost: 0.87528  |  accuracy: 22.00%
epoch #6  |  cost: 0.86876  |  accuracy: 25.00%
epoch #7  |  cost: 0.85164  |  accuracy: 28.00%
epoch #8  |  cost: 0.83687  |  accuracy: 38.00%
epoch #9  |  cost: 0.82138  |  accuracy: 32.00%
epoch #10  |  cost: 0.78694  |  accuracy: 48.00%
epoch #11  |  cost: 0.75509  |  accuracy: 55.00%
epoch #12  |  cost: 0.73479  |  accuracy: 50.00%
epoch #13  |  cost: 0.69185  |  accuracy: 56.00%
epoch #14  |  cost: 0.65528  |  accuracy: 58.00%
epoch #15  |  cost: 0.62176  |  accuracy: 62.00%
epoch #16  |  cost: 0.57570  |  accuracy: 62.00%
epoch #17  |  cost: 0.54624  |  accuracy: 64.00%
epoch #18  |  cost: 0.51292  |  accuracy: 66.00%
epoch #19  |  cost: 0.49829  |  accuracy: 66.00%
epoch #20  |  cost: 0.46039  |  accuracy: 68.00%

Elapsed time: 269.813s.
```

همچنین نمودار cost بر حسب بیست epoch طی شده نیز مانند زیر می‌باشد.

```
plt.plot(epoch_costs)
plt.show()
```



اگر زمان اجرا برایتان معقول بود میتوانید به ازای تعداد epoch بیشتر هم، کدتان را تست کنید و نتایج را گزارش کنید.

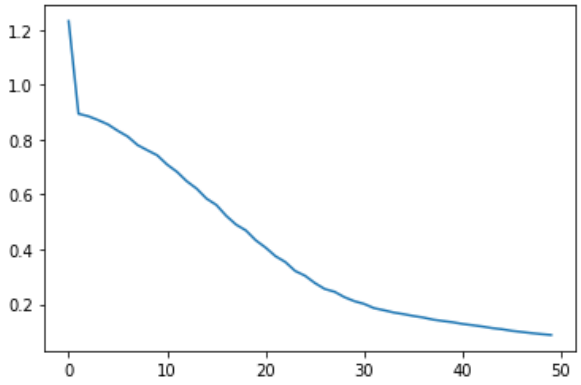
با استفاده از همان تابع قبل و تعداد ۱۰۰ داده این بار دقت مدل را برای پنجاه epoch بررسی می‌کنیم.

```
epoch #31  |  cost: 0.20153  |  accuracy: 89.00%
epoch #32  |  cost: 0.18617  |  accuracy: 90.00%
epoch #33  |  cost: 0.17850  |  accuracy: 90.00%
epoch #34  |  cost: 0.17012  |  accuracy: 91.00%
epoch #35  |  cost: 0.16426  |  accuracy: 93.00%
epoch #36  |  cost: 0.15751  |  accuracy: 92.00%
epoch #37  |  cost: 0.15194  |  accuracy: 95.00%
epoch #38  |  cost: 0.14417  |  accuracy: 95.00%
epoch #39  |  cost: 0.13919  |  accuracy: 95.00%
epoch #40  |  cost: 0.13462  |  accuracy: 95.00%
epoch #41  |  cost: 0.12852  |  accuracy: 95.00%
epoch #42  |  cost: 0.12393  |  accuracy: 95.00%
epoch #43  |  cost: 0.11913  |  accuracy: 95.00%
epoch #44  |  cost: 0.11330  |  accuracy: 95.00%
epoch #45  |  cost: 0.10917  |  accuracy: 95.00%
epoch #46  |  cost: 0.10333  |  accuracy: 95.00%
epoch #47  |  cost: 0.09921  |  accuracy: 95.00%
epoch #48  |  cost: 0.09504  |  accuracy: 95.00%
epoch #49  |  cost: 0.09138  |  accuracy: 95.00%
epoch #50  |  cost: 0.08816  |  accuracy: 95.00%

Elapsed time: 669.683s.
```

همچنین نمودار cost بر حسب پنجاه epoch طی شده نیز مانند زیر می‌باشد.

```
plt.plot(epoch_costs)
plt.show()
```



قدم چهارم: Vectorization

تعداد epoch را افزایش دهید به عدد 200 و دقت مدل نهایی، زمان اجرای فرآیند یادگیری و همچنین پلات cost در طی زمان را گزارش کنید.

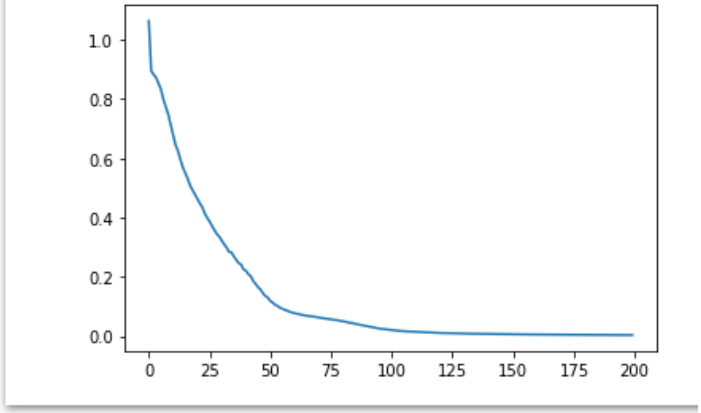
در شکل زیر دقت مدل برای ۱۰۰ داده و با استفاده از تابع "compute_grads_vectorized" (که از vectorization استفاده می‌کند) آمده‌است (می‌توان گفت در این حالت مدل overfit شده‌است).

```
epoch #182  |  cost: 0.00460  |  accuracy: 100.00%
epoch #183  |  cost: 0.00456  |  accuracy: 100.00%
epoch #184  |  cost: 0.00451  |  accuracy: 100.00%
epoch #185  |  cost: 0.00447  |  accuracy: 100.00%
epoch #186  |  cost: 0.00443  |  accuracy: 100.00%
epoch #187  |  cost: 0.00438  |  accuracy: 100.00%
epoch #188  |  cost: 0.00435  |  accuracy: 100.00%
epoch #189  |  cost: 0.00431  |  accuracy: 100.00%
epoch #190  |  cost: 0.00427  |  accuracy: 100.00%
epoch #191  |  cost: 0.00423  |  accuracy: 100.00%
epoch #192  |  cost: 0.00419  |  accuracy: 100.00%
epoch #193  |  cost: 0.00416  |  accuracy: 100.00%
epoch #194  |  cost: 0.00412  |  accuracy: 100.00%
epoch #195  |  cost: 0.00409  |  accuracy: 100.00%
epoch #196  |  cost: 0.00405  |  accuracy: 100.00%
epoch #197  |  cost: 0.00401  |  accuracy: 100.00%
epoch #198  |  cost: 0.00399  |  accuracy: 100.00%
epoch #199  |  cost: 0.00395  |  accuracy: 100.00%
epoch #200  |  cost: 0.00392  |  accuracy: 100.00%

Elapsed time: 9.572s.
```

همچنین نمودار cost بر حسب دویست epoch طی شده نیز مانند زیر می‌باشد.

```
plt.plot(epoch_costs)
plt.show()
```



قدم پنجم: تست کردن مدل

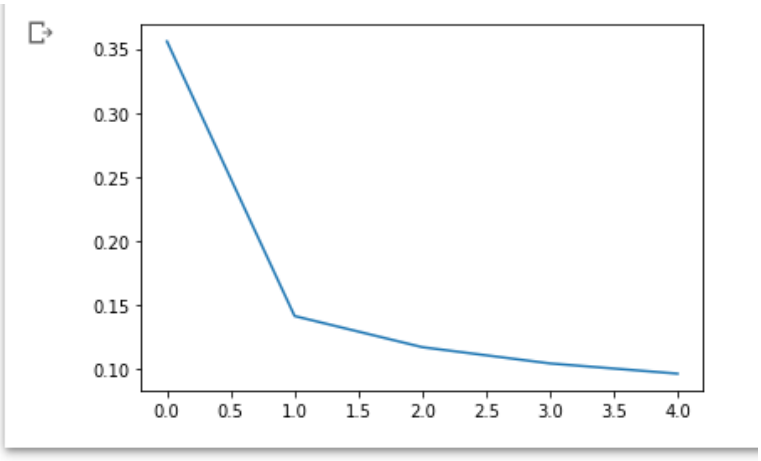
در پایان این مرحله، دقت مدل را برای مجموعه‌ی train و همچنین برای مجموعه test گزارش کنید. همچنین همانند قبل میانگین cost را نیز پلات کنید.

در شکل زیر دقت مدل برای تمامی داده‌ها و با استفاده از تابع "compute_grads_vectorized" (که از vectorization استفاده می‌کند) آمده‌است.

```
epoch #1  |  cost: 0.35561  |  accuracy: 77.88%
epoch #2  |  cost: 0.14153  |  accuracy: 91.50%
epoch #3  |  cost: 0.11727  |  accuracy: 92.89%
epoch #4  |  cost: 0.10460  |  accuracy: 93.66%
epoch #5  |  cost: 0.09662  |  accuracy: 94.15%

Elapsed time: 130.203s.
```

همچنین نمودار cost بر حسب پنج epoch طی شده نیز مانند زیر می‌باشد.



دقت این مدل پس از فرآیند یادگیری بر روی داده‌های تست در شکل زیر آمده‌است و برابر است با ۹۳,۹۱ درصد.

```
accuracy = 0
l = np.shape(test_set)[0]

for i in range(l):
  x = test_set[i][0]
  label = np.argmax(test_set[i][1])

  prediction, _ = feed_forward(x)

  accuracy += 1 if (prediction == label) else 0

print(f"Accuracy on the test set: {((accuracy / l) * 100):.02f}%")
```

```
Accuracy on the test set: 93.91%
```

بخش تحقیقی امتیازی

سوال اول

مجموعهٔ اعتبارسنجی به طور کلی برای بررسی عملکرد شبکهٔ عصبی به ازای هایپرپارامترها و روش‌های مختلف و در راستای بهبود آن استفاده می‌شود. برای مثال در صورتی که بخواهیم مقدار مناسبی را برای learning rate شبکه انتخاب کنیم می‌توانیم چند شبکه را به ازای learning rate های مختلف آموزش دهیم و سپس عملکرد هر یک از این شبکه‌ها را بر روی داده‌های اعتبارسنجی بررسی کنیم. سپس مقداری از learning rate را که به ازای آن عملکرد شبکه بهتر است را برگزینیم. این کار به طور کلی بر روی هایپرپارامترهای مختلف و حتی تعداد لایه‌ها و نورون‌های شبکه قابل انجام است. همچنین عملکرد روش‌های مختلف مانند روش‌های مختلف optimization را می‌توان بر روی آن سنجید.

به طور معمول در صورتی که تعداد کل داده‌ها کم باشند (تا حدود ده هزار داده) داده‌های train، validation و test به نسبت ۶۰، ۲۰ و ۲۰ درصد انتخاب شده و در صورتی که داده‌ها زیاد باشند (در حدود یک میلیون داده) داده‌های train، validation و test به نسبت ۹۸، ۱ و ۱ درصد انتخاب می‌شوند. به علاوه به طور کلی داده‌های اعتبارسنجی باید از توزیع یکسانی با داده‌های تست (داده‌هایی که در دنیای واقعی مدل قرار است در رابطه با آن‌ها به ما پاسخ دهد) باشند تا بتوانند عملکرد مدل را در دنیای واقعی بسنجند، اما لزوماً می‌توانند با تمام داده‌های آموزشی توزیع یکسانی نداشته باشند (هر چند که بهتر است این توزیع‌ها نزدیک باشند).

تفاوت این مجموعه با مجموعهٔ تست این است که مجموعهٔ تست نباید در روند یادگیری شبکه و برای تصمیم‌گیری برای آن استفاده شود و تنها پس از اینکه مدل به طور کامل آموزش داده شد باید بتواند عملکرد کلی شبکه را گزارش کند. اما از مجموعهٔ اعتبارسنجی همانطور که بیان شد در طی روند ساخت شبکهٔ نهایی استفاده می‌شود.

سوال دوم

تفاوت این روش‌ها در زمانی است که پارامترهای مدل (با استفاده از روش gradient descent) را آپدیت می‌کنیم. در روش «گرادیان کاهشی دسته‌ای» آپدیت کردن پارامترها پس از پیمایش تمام داده‌ها، در روش «گرادیان کاهشی تصادفی» به ازای هر یک از داده‌ها و در روش «گرادیان کاهشی دسته‌ای کوچک» پس از پیمایش تعداد مشخصی از داده‌ها (به اندازهٔ batch size) انجام می‌شود.

در روش‌های دسته‌ای کوچک و تصادفی تمام مجموعهٔ آموزشی باید پس از هر epoch به صورت تصادفی shuffle شود، اما انجام این کار برای روش دسته‌ای لزومی ندارد زیرا تمام داده‌ها به هر حال پیمایش می‌شوند و ترتیب آن‌ها اهمیتی ندارد.
به طور کلی سرعت یادگیری در روش دسته‌ای از روش دسته‌ای کوچک کمتر بوده و سرعت یادگیری در روش دسته‌ای کوچک از روش تصادفی کمتر می‌باشد. به همین دلیل معمولاً بر روی داده‌های با تعداد بالا از روش دسته‌ای استفاده نمی‌شود. از طرفی روش دسته‌ای در صورتی که زمان کافی داشته باشد (به تعداد کافی epoch اجرا شود) به سمت یک جواب بهینه همگرا خواهد شد اما در روش دسته‌ای کوچک و تصادفی لزوماً در هر بار آپدیت کردن پارامترها به سمت پاسخ بهینه نخواهیم رفت (هر چند که روند کلی به سمت مینیمم هزینه‌ها است).
این ویژگی شاید به طور کلی خوب باشد اما مشکل آن این است که در روش دسته‌ای احتمال گیر کردن در یک مینیمم محلی بیش‌تر است، زیرا در دو روش دیگر آپدیت کردن پارامترها لزوماً به سمت مینیمم هزینه برای کل داده‌ها نیست و هر بار به جهت مختلفی حرکت خواهیم کرد که می‌تواند ما را از مینیمم محلی خارج کند.
در نهایت به دلیل معایب و مزایای هر یک از روش‌ها امروزه به طور کلی بیش‌تر از روش دسته‌ای کوچک که حالت میانه‌ای میان این روش‌هاست استفاده می‌شود تا سرعت و دقت یادگیری بالاتری داشته باشیم.

سوال سوم

یکی از ایده‌هایی که برای افزایش سرعت یادگیری شکبهٔ عصبی استفاده می‌شود نرمال‌سازی داده‌های ورودیست. این نرمال‌سازی باعث می‌شود که هر یک از فیچرهای ورودی به طور کلی بر روی بازهٔ یکسانی و حول مبدأ توزیع شوند. برای این کار ابتدا میانگین فیچرهای داده‌ها را از هر یک از آن‌ها کم کرده و سپس آن‌ها را تقسیم بر واریانس می‌کنیم. این کار علی‌الخصوص زمانی که در ابتدا بازهٔ ورودی‌ها بسیار متفاوت‌اند (برای مثال یکی از ۰ تا ۱ تغییر کرده و دیگری از ۱ تا ۱۰۰۰ متغیر است) سرعت یادگیری را افزایش می‌دهد. زیرا منجر به این می‌شود که در نهایت cost function شکل قرینه‌تر و گردتری داشته باشد و بتوانیم بدون oscillate کردن بسیار با learning rate بزرگ‌تری این تابع را بهینه‌سازی کنیم.
حال می‌توان این ایده را از روی لایهٔ اول شبکهٔ عصبی به تمام لایه‌ها تعمیم داد. هر یک از لایه‌های دیگر شبکه نیز در حقیقت مانند لایهٔ ورودی ورودی‌هایی دارند که همان خروجی activation function پرسپترون‌های لایهٔ قبل است. بنابراین این ورودی می‌تواند با ایدهٔ مشابهی نرمال‌سازی شود تا فرآیند یادگیری با سرعت بیش‌تری صورت گرفته و بتوانیم از learning rate بزرگ‌تری استفاده کنیم.
برای این کار می‌توان خروجی پرسپترون لایهٔ قبل را را پیش و یا پس از اعمال تابع فعالیت نرمال‌سازی کرد، اما در عمل این کار به طور معمول پیش از اعمال تابع فعالیت انجام می‌شود. فرض کنید برای یک لایهٔ خاص از شبکه

با m نورون، خروجی نورون i ام را (پیش از اعمال تابع فعالیت) با z_i نمایش دهیم. در این صورت مراحل انجام این نرمال‌سازی مانند زیر است:

$$\mu = \frac{1}{m}\sum_{i=0}^{m} z_i$$
$$\sigma^2 = \frac{1}{m}\sum_{i=0}^{m} (z_i - \mu)^2$$
$$z_{norm_i} = \frac{z_i - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$
$$\tilde{z}_i = \gamma z_{norm_i} + \beta$$

در این فرآیند برای اینکه همواره میانگین داده‌ها برابر با صفر و واریانس داده‌ها برابر با یک نشود، از دو پارامتر گاما و بتا (واریانس و میانگین نهایی) استفاده کرده‌ایم که خود قابل یادگیری می‌باشند و می‌توانند با هر روش optimization ای یاد گرفته شوند. همچنین اپسیلون عدد کوچکی است که برای جلوگیری از صفر شدن مخرج به مقدار واریانس افزوده شده‌است.
در نهایت به جای مقدار z_i به تابع فعالیت هر نورون، مقدار $\tilde{z}_i$ را به عنوان ورودی به آن خواهیم داد.

سوال چهارم

در شبکه‌های CNN بسته به تعداد فیلترها ابعاد خروجی لایه‌ها می‌توانند به تدریج بسیار بزرگ شوند و حتی به صورت نمایی رشد کنند. این قضیه باعث می‌شود هر چه پیش می‌رویم پارامترهای بیش‌تری داشته و برای محاسبهٔ خروجی هر لایه نیاز به محاسبات بیش‌تری داشته باشیم. به همین دلیل لایه‌های pooling میان لایه‌های اصلی قرار می‌گیرند تا پس از گذر از هر چند لایه ابعاد خروجی لایهٔ قبل را کوچک‌تر کنند.
فیلتر استفاده شده در عمل pooling تقریباً همیشه یک فیلتر ۲ در ۲ و با stride دو پیکسل می‌باشد (در این حالت ابعاد عکس نصف خواهد شد). دو روش موجود برای این کار مانند زیر می‌باشند.

۱. استفاده از ماکسیمم‌گیری: در این روش مقدار ماکسیمم پیکسل‌های درون فیلتر گرفته شده و این مقدار برای یک پیکسل در عکس جدید در نظر گرفته می‌شود. با این کار تصویر بدست آمده نسبت به تصویر اولیه sharp تر خواهد بود.
۲. استفاده از میانگین‌گیری: در این روش مقدار میانگین پیکسل‌های درون فیلتر گرفته شده و این مقدار برای یک پیکسل در عکس جدید در نظر گرفته می‌شود. با این کار تصویر بدست آمده نسبت به تصویر اولیه smooth تر خواهد بود.

به طور کلی در شبکه‌های CNN استفاده از روش اول مرسوم‌تر است زیرا فیچرهای تصویر را برای استفادهٔ لایهٔ بعدی مشخص‌تر نشان خواهد داد.

دلیل ارجحیت شبکه‌های CNN بر شبکه‌های عادی برای مسائلی که با عکس‌ها سر و کار دارند این است که این نوع از شبکه‌ها می‌توانند به صورت اتوماتیک به تدریج ویژگی‌های خاصی را از یک عکس استخراج کنند. برای مثال در یک شبکهٔ CNN ممکن است لایهٔ اول لبه‌ها، لایهٔ دوم منحنی‌ها، لایهٔ سوم شکل یک دایره، لایهٔ چهارم شکل چرخ یک دوچرخه و ... را تشخیص داده تا نهایتاً یک شیء کامل در عکس شناسایی شود. این ویژگی علاوه بر ساختارمندتر کردن شبکه باعث می‌شود بتوانیم از شبکه‌هایی که از پیش آموزش دیده‌اند برای مسائل جدید استفاده کنیم، زیرا بسیاری از ویژگی‌های پایه (مانند تشخیص اشکال سادهٔ هندسی) میان تصاویر مختلف مشترک می‌باشند و نیازی به یادگیری مجدد برای لایه‌های ابتدایی نیست.
به علاوه با استفاده از این شبکه‌ها و فیلترهای استفاده شده در آن‌ها در صورتی که یک عکس با ابعاد بالا را به شبکه بدهیم می‌توانیم ابعاد آن را بدون آن‌که اطلاعات اساسی تصویر را از دست دهیم کاهش دهیم.