دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر

پروژه اول درس هوش محاسباتی

نگارش
بردیا اردکانیان

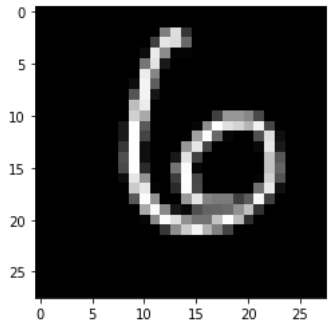
استاد
دکتر عبادزاده

آذر ۱۴۰۱

پیاده‌سازی پروژه

۱) دریافت دیتاست

این پروژه در پلتفرم گوگل کولب پیاده سازی شده است. بنابراین دیتاست مربوطه را در گوگل درایو بارگزاری کرده‌ام و با کمک کدی که در اختیارمان گذاشته شد دیتاست را دریافت و به لیست‌های داده ازمایشی و داده تستی تقسیم کردم. برای تست کردن صحت این موضوع یکی از داده‌های دیتاست را به دلخواه رسم کرده‌ام.



۲) محاسبه خروجی (Feed Forward)

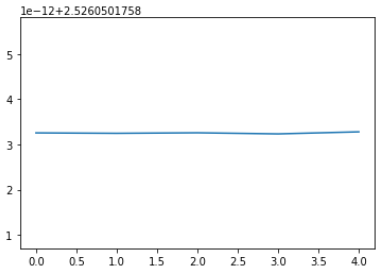
تابع feed_forward به صورت زیر پیاده سازی شده است.

```
def feed_forward():
  global a0, a1, a2, a3
  global z1, z2, z3
  global y
  global W1, b1, W2, b2, W3, b3

  # Layer 1
  z1 = W1 @ a0 + b1
  a1 = sigmoid(z1)
  # Layer 2
  z2 = W2 @ a1 + b2
  a2 = sigmoid(z2)
  # Layer 3
  z3 = W3 @ a2 + b3
  a3 = sigmoid(z3)
```

- نرون‌ها (متغیر a) از ماتریس‌های ۰ تشکیل‌شده‌اند.
- وزن‌ها از مقادیر تصادفی بین ۰ و ۱ تشکیل‌شده‌اند.
- بایاس‌ها از ماتری‌های ۰ تشکیل‌شده‌اند.

در صورتی که بعد از پیشروی به سمت جلو، backpropagation نداشته باشیم دقت مدل بسیار پایین می‌آید. در آزماشی که صورت گرفته بعد از پنج epoch دقت 9.04 درصدی در داده آموزشی و 8.92 درصدی در داده تست بدست آورد. همچنین میانگین هزینه به شکل ذیل می‌باشد.



۲) پیاده سازی Backpropagation

این تابع به شکل ذیل پیاده سازی شده است.

```
def backpropagation():
  global a0, a1, a2, a3
  global sig1, sig2, sig3
  global y
  global W1, b1, W2, b2, W3, b3
  global grad_W1, grad_b1, grad_W2, grad_b2, grad_W3, grad_b3

  # Layer 3
  grad_W3 += (2 * d_sigmoid(sig3) * (a3 - y)) @ (np.transpose(a2))
  grad_b3 += 2 * d_sigmoid(sig3) * (a3 - y)
  grad_a2 = np.transpose(W3) @ (2 * d_sigmoid(sig3) * (a3 - y))
  # Layer 2
  grad_W2 += (d_sigmoid(sig2) * grad_a2) @ (np.transpose(a1))
  grad_b2 += d_sigmoid(sig2) * grad_a2
  grad_a1 = np.transpose(W2) @ (d_sigmoid(sig2) * grad_a2)
  # Layer 1
  grad_W1 += (d_sigmoid(sig1) * grad_a1) @ (np.transpose(a0))
  grad_b1 += d_sigmoid(sig1) * grad_a1
```

همچنین بعد از انجام عملیات فوق، وزن‌لایه‌ها و بایاس‌ها به‌روز می‌شوند.

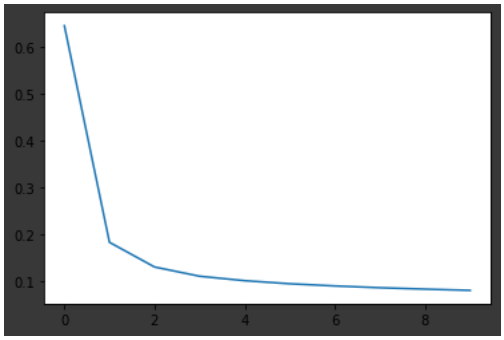
```
def update():
  global W3, b3, W2, b2, W1, b1

  # Weight and Bias 3
  W3 = W3 - alpha * (grad_W3 / batch_size)
  b3 = b3 - alpha * (grad_b3 / batch_size)
  # Weight and Bias 2
  W2 = W2 - alpha * (grad_W2 / batch_size)
  b2 = b2 - alpha * (grad_b2 / batch_size)
  # Weight and Bias 1
  W1 = W1 - alpha * (grad_W1 / batch_size)
  b1 = b1 - alpha * (grad_b1 / batch_size)
```

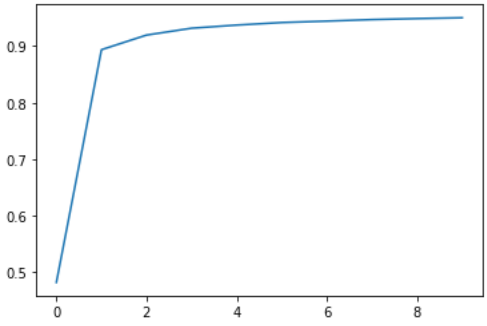
با توجه به عملیات فوق و با اعمال یادگیری بر ۱۰۰ داده انتخاب شده دقت مدل در داده آزمایشی 43.50 درصد و در داده تست 42.82 درصد می‌باشد. دقت کنید که این داده مربوط به یک epoch می‌باشد. با افزایش تعداد epoch به ده دقت مدل در داده آزمایشی 95.41 درصد و در داده تست 94.20 درصد می‌باشد. همچنین پلات شده میانگین هزینه و دقت در هر epoch به شرح ذیل می‌باشد.

```
26099, 60000
Train Accuracy = 43.50%
```

```
4282, 10000
Test Accuracy = 42.82%
```



پلات شده هزینه



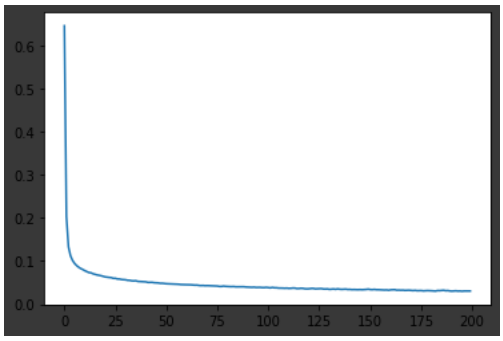
پلات شده دقت

۴) Vectorization

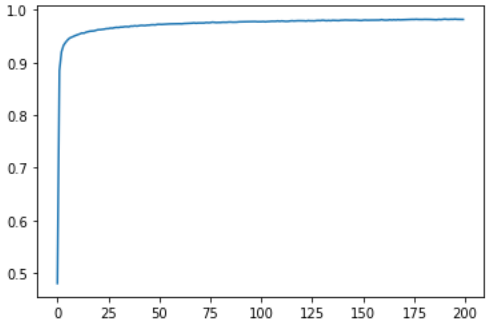
با بهبود بخشیدن به توابع نوشته شده (توابع فوق را ببینید با همین تکنیک زده شده ورژن ذخیره شده‌ای از قبل این مرحله نداشته‌ام) سرعت مدل چند برابر شد و زمینه برای آزمایش در ۲۰۰ epoch فراهم شد. دقت مدل در داده آزمایشی 98.29 درصد و در داده تست 94.01 درصد می‌باشد. همچنین پلات شده میانگین هزینه و دقت در هر epoch به شرح ذیل می‌باشد.

```
58973, 60000
Train Accuracy = 98.29%
```

```
9401, 10000
Test Accuracy = 94.01%
```



پلات شده هزینه



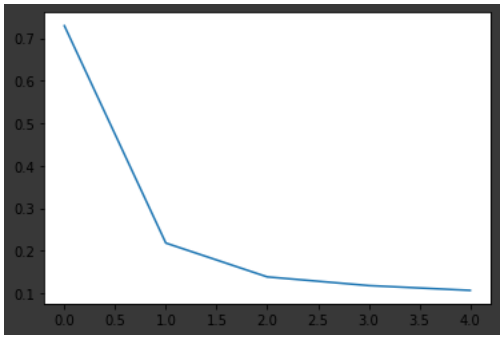
پلات شده دقت

۵) تست کردن مدل

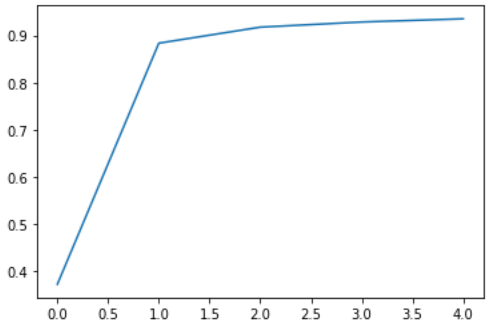
با تغییر دادن هایپرپارامترها به موارد ذکر شده دقت مدل در داده آزمایشی 94.08 درصد و در داده تست 93.13 درصد می‌باشد. همچنین پلات شده میانگین هزینه و دقت در هر epoch به شرح ذیل می‌باشد.

```
56445, 60000
Train Accuracy = 94.08%
```

```
9313, 10000
Test Accuracy = 93.13%
```



پلات شده هزینه



پلات شده دقت

بخش امتیازی تحقیقی

سوال اول)

Cross validation تکنیکی برای ارزیابی مدل‌های ML با آموزش چندین مدل ML بر روی زیر مجموعه‌های داده‌های ورودی موجود و ارزیابی آن‌ها بر روی زیر مجموعه داده‌های مکمل است. از Cross valiation برای تشخیص بیش برازش و مشخص کردن هایپرپارامترها استفاده می‌شود. مجموعه داده اعتبار سنجی (validation set) عملا بخشی از مجموعه داده آموزشی (training set) بوده که برای اعتبار سنجی مدل حین یادگیری استفاده می‌شود، در صورتی که مجموعه داده تست در زمان آموزش مدل در اختیار ما نیست.

حال برای cross validation که معمولا در روش k-fold استفاده می‌شود، n بار داده ها را به k بخش تقسیم کرده که از k-۱ بخش برای آموزش و از بخش مانده باری validation استفاده می‌کنیم. این تکنیک زمانی استفاده می‌شود که دیتاست موجود به میزانی نباشند که بتوانیم دو مجموعه اعتبار سنجی و آموزش جدا داشته باشیم.

سوال دوم)

Batch gradient descent، خطا را برای هر مثال در مجموعه داده آموزشی محاسبه می‌کند، اما تنها پس از ارزیابی همه نمونه‌های آموزشی، مدل به‌روزرسانی می‌شود. کل این فرآیند مانند یک چرخه است و به آن دوره آموزشی می گویند. برخی از مزایای این تکنیک کارایی محاسباتی آن است: یک گرادیان خطای پایدار و یک همگرایی پایدار ایجاد می‌کند. برخی از معایب این است که گرادیان خطای پایدار گاهی اوقات می‌تواند منجر به حالت همگرایی شود که بهترین حالتی نیست که مدل می تواند به دست آورد. همچنین نیاز دارد که کل مجموعه داده آموزشی در حافظه باشد و در دسترس الگوریتم باشد. در این روش ما تمام مجموعه داده آموزشی را در هر epoch استفاده می‌کنیم تا اول آموزش دهیم و بعد وزن های فعلی را آپدیت کنیم که باعث می‌شود در مجموعه داده بزرگ مثل ۱۰ میلیون داده، زمان آموزش را در هر epoch به نحو قابل ملاحظه ای زیاد کند. این روش، حالت optimal بوده اما معمولا در سناریو واقعی، منابع مورد نیاز برای این کار را نداریم.

در مقابل، نزول گرادیان تصادفی (SGD) این کار را برای هر مثال آموزشی در مجموعه داده انجام می‌دهد، به این معنی که پارامترهای هر مثال آموزشی را یک به یک به روز می‌کند. بسته به مشکل، این می تواند SGD را سریعتر از نزول گرادیان دسته‌ای کند. یکی از مزیت‌ها این است که به‌روزرسانی‌های مکرر به ما امکان می‌دهد تا میزان پیشرفت بسیار دقیقی داشته باشیم. با این حال، به‌روزرسانی‌های مکرر از نظر محاسباتی گران‌تر از رویکرد نزولی گرادیان دسته‌ای هستند. علاوه بر این، فرکانس این به‌روزرسانی‌ها می‌تواند منجر به گرادیان‌های پر نویزی شود که ممکن است باعث شود به جای کاهش آهسته میزان خطا، به اطراف بپرد. در این روش به دلیل برداشتن تنها یک داده در هر epoch، نمودار آموزش با نوسانات نسبتا زیادی مواجه می‌شود. از طرفی هیچ گاه به نقطه مینیمم محلی نمیرسد، اما در این حین روند کاهشی خواهد بود.

Mini batch gradient descent روشی است که به آن توجه می‌شود زیرا ترکیبی از مفاهیم SGD و گرادیان نزولی دسته‌ای است. به سادگی مجموعه داده آموزشی را به دسته های کوچک تقسیم می‌کند و برای هر یک از آن دسته ها به روز رسانی انجام می‌دهد. این تعادل بین استحکام نزول گرادیان تصادفی و کارایی نزول گرادیان دسته ای ایجاد می‌کند. اندازه‌های مینی بچ معمولی بین ۵۰ تا ۲۵۶ است، اما مانند هر تکنیک یادگیری ماشین دیگری، قانون واضحی وجود ندارد زیرا برای کاربردهای مختلف متفاوت است. این الگوریتمی است که هنگام آموزش شبکه عصبی استفاده می شود و رایج ترین نوع نزول گرادیان در یادگیری عمیق است. مشکل روش قبلی آن است که به دلیل برداشتن تنها یک داده آموزشی در هر مرحله، نمی‌توان فرایند را vectorize کرده و در نتیجه سرعت کاهش میابد. برای حل این مشکل از Mini-Batch Gradient Descent استفاده می‌کنیم

سوال سوم)

نرمال سازی دسته‌ای روشی است که برای آموزش شبکه های عصبی مصنوعی سریعتر و پایدارتر از طریق عادی سازی ورودی لایه ها با مرکزیت مجدد و مقیاس گذاری مجدد استفاده می‌شود. افزایش سرعت و متعادل نگه داشتن شبکه عصبی به گونه‌ای است که قبل انتقال داده گان به تابع فعال سازی، به کمک واریانس و میانه آنها را نرمال می‌کند. این روش کمک می‌کند تا اگر در وزن‌ها نوسانات زیاد را تعدیل کند تا از overfit زود هنگام جلوگیری شود. به عبارتی خروجی این الگوریتم داده های با توزیع نرمال خواهند بود.

سوال چهارم)

لایه های Polling برای کاهش ابعاد ویژگیی‌ها استفاده می‌شود. بنابراین، تعداد پارامترهای یادگیری و میزان محاسبات انجام شده در شبکه را کاهش می دهد. لایه ادغام ویژگی های موجود در یک منطقه از نقشه ویژگی ایجاد شده توسط یک لایه کانولوشن را خلاصه می کند. معمولا تعداد زیادی از عکس ها بخش‌هایی دارد که اطلاعات زیادی به ما نمیدهند و مفید نیستند و به علت تعداد زیاد داده‌ها معمولا فضای زیادی اشغال کرده و باعث می‌شوند مدل سنگین شود. به عنوان مثال در دیتاست MNIST گوشه‌های تصویر که تماما سیاه هستند ممکن است مفید نباشد. پس از pooling استفاده می‌کنیم تا اطلاعات بخشی از عکس را خلاصه کنیم. این خلاصه کردن به کمک مینیمم گیری، ماکسیمم گیری، متوسط گیری، میانه و واریانس صورت می‌گیرد. با این کار می‌تواند در چند لایه اندازه تصاویر را کوچک تر کرد تا سرعت پردازش افزایش یابد و از الگوبرداری از الگوهای ناخواسته جلوگیری شود.

در ادامه، شبکه CNN می‌تواند روابط بین پیکسل‌های نزدیک را درک کند در صورتی که در یک شبکه ساده MLP این کار امکان پذیر نیست. همچنین عکس یک مفهوم دو بعدی است که شبکه های MLP نمی‌توانند آنرا پردازش کرده و باعث می‌شود بخشی از اطلاعات عکس از بین برود. در ادامه CNN سریع‌تر همگرا می‌شود و برخلاف MLP الگوهای ناخواسته را الگوبرداری نمی‌کند.

بخش امتیازی پیاده سازی

در پیاده‌سازی بخش امتیازی از چندین تکنیک استفاده شده که در جدولی به توضیح آنها می‌پردازیم. تلاش کردم تا با کمک OPTIMIZERهای مختلف در کنار انواع SCHEDULERها به بهبود مدل بپردازم. چالشی که وجود دارد از پیش اماده بودن مدل می‌باشد که انعطاف را از پیاده سازی و استفاده از مدل RESNET۵۰ می‌گیرد. در جدول زیر انواع پیاده‌سازی‌ها و نتایج دقت در داده تست و آموزشی لیست شده‌اند.

nn	Optimizer	Learning Rate	Scheduler	SWA	Epochs	Train Accuracy	Test Accuracy
Linear	Adam	1e-3	x	x	200	84	60
Linear	Adam	1e-3	ExponentialLR, MultiStepLR	x	5	79	52
Linear	Adam	1e-2	x	x	5	74	57
Linear	Adam	1e-2	ExponentialLR, MultiStepLR	x	5	80	51
Linear	Adam	1e-1	x	x	5	81	54
Linear	Adam	1e-1	ExponentialLR, MultiStepLR	x	5	81	54
Linear	SGD	3e-4	x	x	5	29	19
Linear	SGD	3e-3	x	x	5	78	59
Linear	SGD	3e-2	x	x	5	77	59
Sequential	SGD	3e-4	x	x	30	72	52
Sequential – HL128	SGD	3e-3	x	x	100	90	66
Sequential – HL256	SGD	3e-3	x	x	100	87	70

به نظر می‌آید که با تغییر LR مدل بهبود آنچنانی نخواهد داشت. حدس می‌زنم که بعضی هایپرپارامترها به صورت بهینه انتخاب نشده‌اند. همچنین Optimizer نیز جای بهبود دارد. کماکان ۷۰ درصد در داده تست و ۸۷ درصد در داده آموزشی بهترین نتیجه‌ای بود که حاصل شد.

تمامی کدهای ژوپیتر در فایل زیپ در دسترس هستند.