**دانشگاه صنعتی امیرکبیر**

**(پلی تکنیک تهران)**

**دانشکده ریاضی و علوم کامپیوتر**

**گزارش هشتم**

# مقایسه روش های مختلف فراابتکاری برای بهینه سازی ضرایب یک شبکه عصبی برای حل یک مساله طبقه بندی benchmark

**نگارش**

**سروش آریانا**

**استاد**

**دکتر مهدی قطعی**

**خرداد ۱۴۰۰**

## مقدمه

شبکه‌های عصبی یک مجموعه الگوریتم‌هایی هستند، که با تقلید از روش کار مغز انسان، سعی در یافتن روابط بین داده‌ها می‌کنند. فانکشن Loss در این شبکه‌ها یکی از مهمترین قسمت‌های یادگیری این دسته از الگوریتم‌ها هستند. در واقع Loss به معنی میزان خطای پیشبینی‌شده از داده‌ها توسط شبکه عصبی است و فانکشن Loss راهکاری محاسبه این میزان خطاست.

بهینه‌سازها (Optimisers) الگوریتم‌ها یا روش‌هایی هستند که برای تغییر خواص شبکه عصبی از جمله وزن‌های شبکه و نرخ یادگیری استفاده می‌شوند تا هرچه بیشتر Loss کم شود. برای شبکه‌های عصبی بهینه‌سازهای مختلف وجود دارد. از جمله این بهینه‌سازها میتوان به SGD، Momentum، RMSProp، Adam، Adadelta، Adamax، AdamW و ... اشاره کرد.

در این گزارش به بررسی دو نمونه از این بهینه‌سازها بر روی تسک تشخیص ارقام دست‌نویس فارسی (Mnist) می‌پردازیم.
کد های مربوطه در کولب قابل دریافت و مشاهده است که در کنار فایل pdf گزارش با نام AI_Assignment_8.ipynb ارسال شده است.

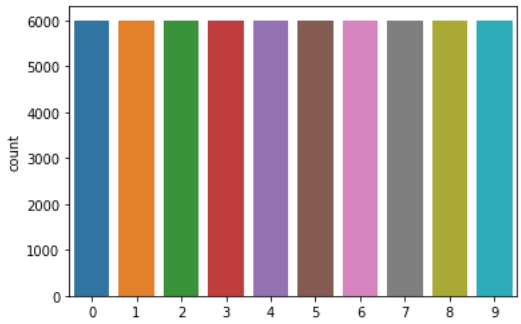
## ۱- دیتاست

دیتای ارقام دست‌نویس یا Mnist یکی از دیتاهای محبوب برای بررسی الگوریتم‌های ماشین لرنینگ، مخصوصا الگوریتم‌های طبقه‌بندی، است. در این پروژه از دیتاست هدی که یک مجموعه ارقام دست‌نویس فارسی است استفاده کردیم. این دیتاست شامل ۱۰۲۳۵۳ نمونه دست نوشته سیاه و سفید است.

این مجموعه اولین مجموعه‌ی بزرگ ارقام دست‌نویس فارسی است که از حدود ۱۲۰۰۰ فرم ثبت‌نام آزمون سراسری کارشناسی ارشد ۱۳۸۴ و آزمون کاردانی پیوسته‌ی دانشگاه جامع علمی کاربردی سال ۱۳۸۳ استخراج شده است.

| رقم ۰ | رقم ۱ | رقم ۲ | رقم ۳ | رقم ٤ | رقم ٥ | رقم ٦ | رقم ٧ | رقم ٨ | رقم ٩ |
|---|---|---|---|---|---|---|---|---|---|
| 10070 | 10330 | 9923 | 10334 | 10333 | 10110 | 10254 | 10363 | 10264 | 10371 |

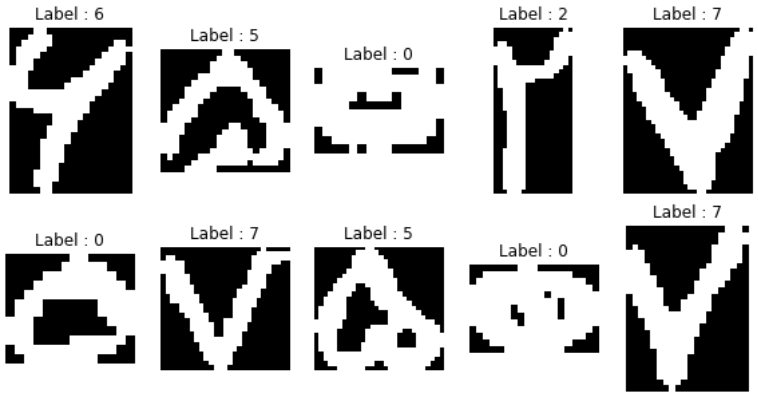
**شکل ۱ توزیع کلاس‌های مختلف داده**




**شکل۲ نمودار توزیع کلاس‌های مختلف در دیتای ترین**

همان‌طور که در شکل ۱ و شکل۲ مشخص است توزیع کلاس‌های مختلف در این دیتاست تا حد بسیار زیادی یکسان است و هر کلاس‌ها در دیتای ترین حدودا ۶۰۰۰ نمونه و در دیتای تست حدودا ۲۰۰۰ نمونه دارد.

این دیتاست از سایت farsiocr قابل دریافت برای استفاده تحقیقاتی است.




**شکل۳ داده‌های خام دیتاست**

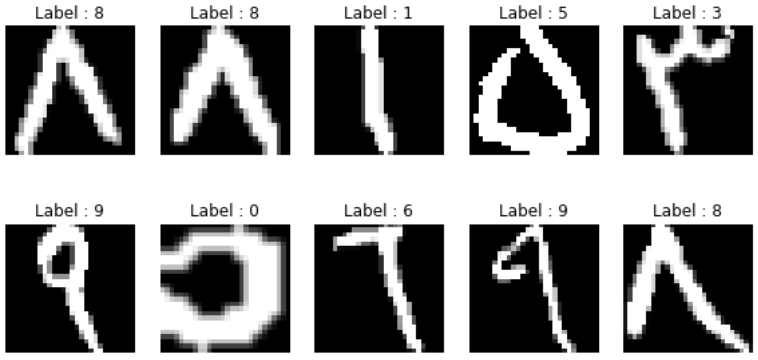
## ۲-۱- پیش‌پردازش

همانطور که در شکل۳ پیداست، در اولین نگاه به دیتاست پس از خواندن داده‌ها اولین مشکلی که به نظر می‌رسد یکسان نبودن ابعاد تصاویر است. برای حل این مشکل، ابتدا تصاویر را برای رسیدن به طول و عرض یکسان لایه‌گذاری (padding) کردیم و سپس آن‌ها را به سایز دلخواه تغییر اندازه دادیم.

```
def reshape_images(dataset, size):
  result = []
  for d in dataset:
    height, width = d.shape
    ww = max(height, width)
    pad = np.zeros((ww, ww))
    xx = (ww - width) // 2
    yy = (ww - height) // 2
    pad[yy:yy+height, xx:xx+width] = d
    result.append(cv2.resize(pad, (size, size)))
  return np.array(result)
```

**کد ۱ تابع مربوط به پیش‌پردازش تصاویر**

در نهایت، همانطور که در شکل۴ مشخص است، تمامی تصاویر به یک اندازه تبدیل شده‌اند و سایز تصاویر ما ۲۸ در ۲۸ است.




**شکل۴ تصاویر دیتاست پس از انجام عملیات پیش‌پردازش**

## ۲- مدل شبکه عصبی

برای این مدل تنها از سه لایه ساده استفاده کردیم. در لایه اول چون مدل ما بصورت چند لایه Dense است و از مدل‌های پیچیده تر (مانند CNN) استفاده نمی‌کنیم، ابتدا ورودی تصویر مدل که با سایز ۲۸ در ۲۸ است را به یک وکتور ۷۸۴تایی تبدیل می‌کنیم. در لایه بعدی این ورودی ۷۸۴تایی را به یک لایه ۱۲۸تایی تبدیل می‌کنیم و در لایه آخر به تعداد کلاس-های موجود در لایه نورون قرار می‌دهیم که تعداد کلاس‌های ما در این تسک ۱۰ است.

در شکل ۵ خلاصه ساختار مدل به تصویر کشیده شده است.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
time: 4.88 ms (started: 2021-06-09 18:54:51 +00:00)
```

**شکل ۵ خلاصه ساختار مدل**

## ۳- توضیح کد

برای خواندن دیتا از فرمت cdb به آرایه پایتون، از کلاس پایتون معرفی شده در سایت رسمی هدی استفاده کردیم. فانکشن read_hoda_cdb به عنوان ورودی آدرس فایل cdb حاوی دیتای ارقام را دریافت کرده و به عنوان خروجی، داده را به عنوان آرایه پایتون تحویل می‌دهد.

برای به تصویر کشیدن این آرایه بصورت تصویر از کتابخانه matlibplot استفاده کردیم. به اینصورت که ابتدا، به تعداد دلخواه، تعدادی از تصاویر را از داده انتخاب کرده آرایه مربوط به آن تصویر و برچسب آن تصویر را دریافت کرده و با فانکشن imshow در کتابخانه آن‌ها را به تصویر کشیدیم.

برای این گزارش ما برای تعریف مدل شبکه عصبی از کتابخانه tensorflow استفاده کردیم. برای راحتی کار از APIی Sequential کراس استفاده کردیم. این API به ما در عین قدرت بالا برای تعریف مدل، سادگی برای اضافه کردن جزئیات بیشتر به مدل می‌دهد و مدل را طبق تعریف بخش ۴ تعریف کردیم.

در تعریف مدل لایه tf.keras.layers.Flatten برای تک بعدی کردن آرایه دو بعدی تصویر و از لایه tf.keras.layers.Dense برای معرفی لایه کاملا متصل استفاده کردیم.

برای هر قسمت از کد بهینه‌سازها از فانکشن compile تنسورفلو برای آماده‌سازی مدل برای آموزش استفاده کردیم. از SparseCategoricalCrossentropy به عنوان فانکشن Loss در مدل‌ها استفاده کردیم. تفاوتی که این فانشکن با CategoricalCrossentropy معمولی دارد فقط در خروجی مدل هست که در اولی شماره لیبل خروجی داده می‌شود در حالی که در دومی خروجی بصورت یک وکتور one-hot است.

به عنوان متریک مدل از SparseCategoricalAccuracy استفاده کردیم که پیاده‌سازی دقت برای خروجی sparse است و بعد از کامپایل کردن مدل فانکشن fit را روی آن صدا زدیم که مدل را ترین می‌کند. خروجی این فانکشن تمامی مقادیر مربوط به Loss و متریک‌های معرفی شده در تمامی ایپاک‌ها است.

## ۴- بهینه سازها

### Momentum

در الگوریتم Momentum با استفاده از مفاهیم فیزیک سعی در کم کردن سرعت همگرایی الگوریتم می‌شود. در این الگوریتم پیدا کردن سرعت نزول شیب دیگر فقط وابسته به حالت فعلی مدل نیست و علاوه بر آن از اطلاعات بدست آمده از حالت قبلی هم استفاده می‌شود.

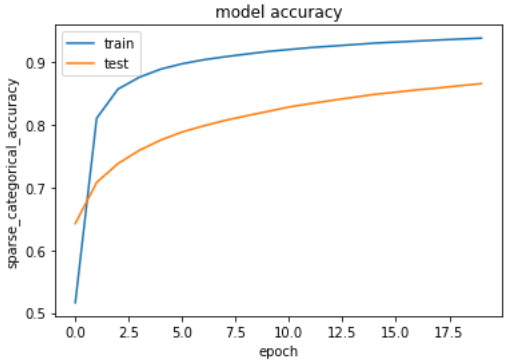
### Adam

Adam یکی از بهترین الگوریتم‌های موجود برای جایگزینی با الگوریتم SGD کلاسیک است. در الگوریتم SGD نرخ یادگیری در طول اجرای مدل ثابت است. اما در Adam با ترکیب تکنیک‌های استفاده شده در RMSProp و Momentum سعی در تغییر مقدار نرخ یادگیری همزمان با پیشرفت یادگیری می‌شود.

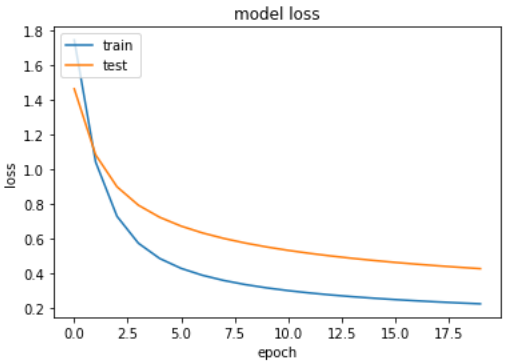
## ۵- بررسی و آنالیز

معیار اصلی‌ای که برای بررسی نتایج این دو بهینه‌ساز استفاده کردیم، دقت (Accuracy) است که تعریف آن بصورت ساده نسبت تعداد پیش‌بینی‌های درست مدل نسبت به تعداد کل پیشبینی‌های مدل است. همچنین علاوه بر دقت مقدار Loss و طول اجرای الگوریتم‌ها در هر طول Step ۲۰ هم بررسی می‌کنیم.

برای تست هر الگوریتم ما یکبار مدل را از ابتدا تعریف کرده و سپس کامپایل و متد fit را روی هر بهینه‌ساز صدا زدیم. برای هر الگوریتم نرخ یادگیری را روی ۰،۰۰۱ قرار دادیم و بقیه پارامترها را بدون تغییر گذاشتیم. در انتها با یک کد ساده matplotlib تاریخچه یادگیری هر کدام از الگوریتم‌ها را با رسم کردیم.

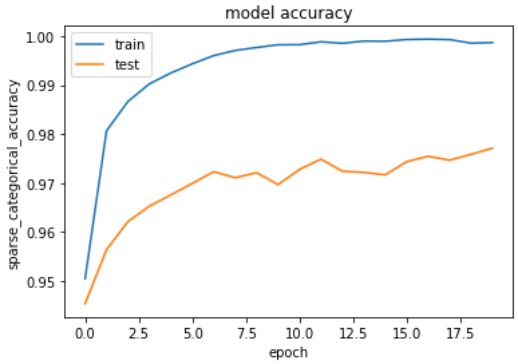



**شکل۶ مقدار دقت مدل در الگوریتم Momentum**

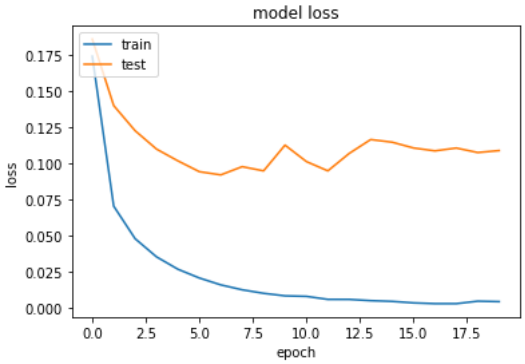



**شکل۷ مقدار Loss مدل در الگوریتم Momentum**

در اجرای الگوریتم Momentum مدل از Loss تقریبا برابر با ۷.۱ و دقت ۵۱ درصد در داده ترین شروع کـرده و بعـد از ۲۰ مرحله به Loss ۰,۲۲ و دقت تقریبا ۹۴ درصد در داده ترین می‌رسد. در داده تست مقـدار Loss از ۱,۴۶ بـه ۰,۴۲ و دقـت از ۶۴ درصد به ۸۶ درصد می‌رسد که دقت قابل قبولی برای تسک ما است.




**شکل ۸ مقدار دقت مدل در الگوریتم Adam**




**شکل ۹ مقدار Loss مدل در الگوریتم Adam**

در اجرای الگوریتم Adam مدل از Loss تقریبا برابر با ۰,۱۷ و دقت ۹۵ درصد در داده ترین شروع کرده و بعد از ۲۰ مرحلـه به Loss ۰,۰۰۴ و دقت تقریبا ۹۹,۸۷ درصد در داده ترین می‌رسد. در داده تست مقدار Loss از ۰,۱۷ بـه ۰,۱ و دقـت از ۹۴ درصد به ۹۸ درصد می‌رسد که دقت فوق العاده خوبی برای تسک Mnist است.

## ۶- نتیجه‌گیری

همانطور که مشاهده شد، هر دو بهینه‌ساز به دقت‌های بالایی روی تسک رسیدند اما در مورد Adam نتایج بسیار بهتـر بـود و این فقط از مقدار Loss و دقت در ایپاک اول قابل مشاهده است. در ایپاک اول الگوریتم Adam بسیار سریع همگرا می‌شـود و مقدار Loss به زیر ۰,۲ می‌رسد در حالی که در ایپاک اول الگوریتم Momentum ایپاک نزدیک بـه ۲ اسـت کـه حـدودا ۱۰ برابر متفاوتند.

در نهایت با استفاده از هر دو بهینه‌ساز تقریبا می‌توان به یک جواب رسید اما برای رسیدن به یک جـواب یکسـان روی هـر دو بهینه‌ساز، Momentum تعداد خیلی بیشتری ایپاک نیاز دارد.

# منابع و مراجع


[۱] “Stochastic Gradient Descent .[درون خطی] ”,Available: https://en.wikipedia.org/wiki/Stochastic_gradient_descent.

[۲] “Plotting with matplotlib .[درون خطی] ”,Available: https://pandas.pydata.org/pandas-docs/version/0.13/visualization.html.

[۳] .[درون خطی] ”مجموعه ارقام دستنویس هدی,“Available: http://farsiocr.ir/%D9%85%D8%AC%D9%85%D9%88%D8%B9%D9%87-%D8%AF%D8%A7%D8%AF%D9%87/%D9%85%D8%AC%D9%85%D9%88%D8%B9%D9%87-%D8%A7%D8%B1%D9%82%D8%A7%D9%85-%D8%AF%D8%B3%D8%AA%D9%86%D9%88%DB%8C%D8%B3-%D9%87%D8%AF%DB%8C./

[۴] “TensorFlow .[درون خطی] ”,Available: http://tensorflow.org./

[۵] “Keras: the Python Deep Learning API .[درون خطی] ”,Available: https://keras.io./