دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

گزارش ۸ درس هوش مصنوعی

مقایسه روشهای مختلف فرا ابتکاری برای بهینه سازی ضرایب یک شبکه عصبی برای حل یک مساله طبقه بندی benchmark

نویسنده:
امیررضا رادجو

استاد
دکتر مهدی قطعی

خرداد ۱۴۰۰

دیتاست مورد استفاده:

دیتای مورد استفاده ما در این گزارش مربوط به داده‌های موجود در کتابخانه sklearn که مربوط به سرطان سینه است می باشد. این داده دارای ۳۱ ویژگی مختلف برای 569 داده مختلف میباشد و در اینجا ما قصد داریم که دیتا ها را بر اساس خوش خیم و یا بدخیم بودن نوع سرطان دسته بندی کنیم.
دیتایی که در اختیار داریم شامل ویژگی‌های مفید و مختلفی مانند میانگین شعاع تقارن تعقر و... می‌باشد. که در ادامه با کمک شبکه‌های عصبی و بهینه ساز های مختلف آن‌ها را طبقه بندی میکنیم.

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	radius error	texture error	perimeter error	area error	smoothness error	compactness error
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	1.0950	0.9053	8.589	153.40	0.006399	0.04904
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	0.5435	0.7339	3.398	74.08	0.005225	0.01308
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	0.7456	0.7869	4.585	94.03	0.006150	0.04006
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	0.4956	1.1560	3.445	27.23	0.009110	0.07458
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	0.7572	0.7813	5.438	94.44	0.011490	0.02461

بهینه سازها:

در ابتدا نیاز است بدانیم بهینه سازها چه هستند و نقششان در شبکه‌های عصبی چیست؟
در ابتدا یک شبکه عصبی را در نظر بگیرید که دارای وزن های تصادفی اولیه می باشد. میدانیم برای سنجش هر شبکه عصبی یک معیار مشخص داریم(برای مثال کمترین مقدار مجموع مربعات را در نظر بگیرید).
حال مقدار این تابع خطا با پارامتر های اولیه تصادفی را به کمک تابع خطا که همان مجموع کمترین مربعات میباشد به دست می آوریم. در این مرحله نقش بهینه ساز ها نمایان میشود. کار بهینه ساز ها این است که با کمک گرفتن از مقدار خطای فعلی و تابع محاسبه خطا (و در برخی موارد یک سری هایپر پارامتر) کمک در بهبودی میزان خطا و کمینه کردن آن نمایند.
در اینجا قصد داریم تعدادی از این بهینه سازها را توضیح دهیم و تفاوت آن‌ها را بگوییم. سپس در بخش بعدی به شکل عملی تعدادی از آن‌ها را با هم مقایسه کرده و دلیل برتری هر یک در هر مورد را ذکر کنیم.

Batch gradient Descent

فرمول کلی این بهینه ساز به شکل زیر می‌باشد که در ادامه با یک مثال آن را توضیح خواهیم داد:

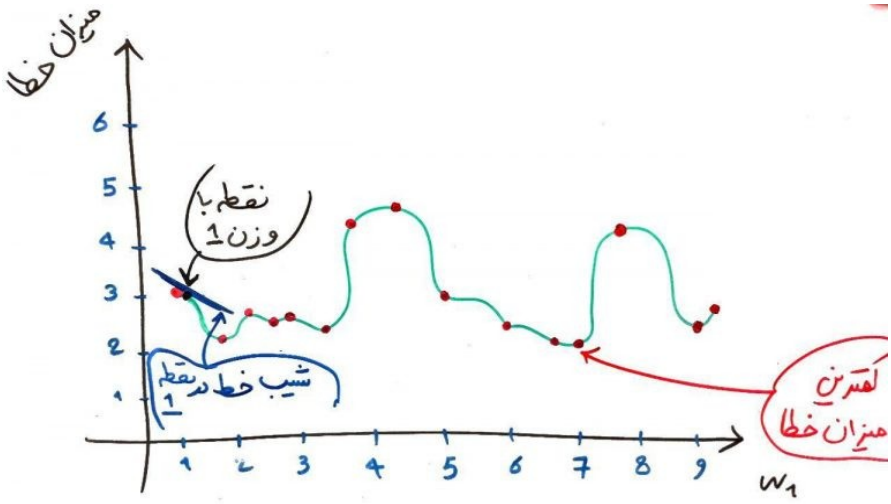
$$\text{Repeat until convergence } \{$$

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$$\}$$

به شکل زیر دقت کنید:

همان طور که مشاهده می‌کنید، کمترین میزان خطا در وزن ۷ اتفاق افتاده است. در روشِ کاهش گرادیان برای پیدا کردن این وزن از قوانین مشتق استفاده می‌شود. همان‌طور که می‌دانید مشتق، نشان‌دهنده‌ی شیبِ خطِ مماس بر یک نقطه از یک تابع است. برای اینکه کمترین میزانِ خطا را به دست آوریم فرض می‌کنیم یک نقطه‌ی دلخواه (یک وزن دلخواه) را در این تابع در نظر گرفته‌ایم. مثلاً نقطه 1



در این نقطه مشتق که همان شیب خطِ مماس بر یک نقطه است یک عدد منفی بوده، چون خط به سمت پایین است.**الگوریتمِ پس انتشار می‌داند که اگر شیبِ خط در یک نقطه (با توجه به وزن‌ها) منفی بود بایستی مقدار آن وزن را افزایش دهد تا شیب خط به صفر برسد.** شیب صفر یعنی کمترین میزان خطای ممکن در آن محدوده (برای درکِ بهتر، در همان تصویر بالا، شیب در محدوده‌ی وزنِ ۱.۷۵ را نگاه کنید، یعنی جایی که خطِ سبز در کمترین میزانِ خود قرار دارد). همان‌طور که در شکلِ بالا مشخص است، کمترین میزانِ خطای ممکن در آن محدوده برای وزن ۱.۷۵ ثبت شده است که شیبِ خط در آن‌جا صفر است (موازی محور افقی است)، حال اگر کمی مقدار وزن را از ۱.۷۵ بیشتر کنیم شیب خط مثبت می‌شود، یعنی شیب به سمت بالا می‌رود. با مثبت بودنِ شیب خط، یعنی همان مشتق در آن نقطه، **الگوریتمِ پس انتشار می‌فهمد که باید وزن را کم کند تا شیب به صفر برسد.**

همان طور که در یک مثال بالا دیدید، الگوریتمِ پس انتشار می‌تواند با استفاده از این این تکنیک یک نقطه‌ی کمینه برای خطا پیدا کند که البته کمترین مقدار در کل فضا نبود ولی به هر حال معقول به نظر می‌رسید. **به این نقطه‌ی معقول یک کمینه‌ی محلی (local minimum)ـ برای خطا می‌گویند.**در شکل

بالا **وزن ۷** یک **کمینه‌ی سراسری، یعنی بهترین نقطه موجود در کل شکل (global minimum)** است. البته رسیدن به این نقطه‌ی سراسری برای الگوریتمِ پس انتشارِ خطا کار دشوار و زمان‌بری است.
برای همین معمولاً الگوریتم در شبکه‌های عصبی اینگونه آموزش می‌بیند که **به تعداد تکرار مشخص یا تا رسیدن به یک خطای کمِ مشخص** الگوریتم را ادامه بدهد و بعد از آن توقف کند. یعنی شبکه عصبی آنقدر تکرار را انجام می‌دهد تا به یک خطای معقول مشخصِ کم برسد . مثلاً در مثالِ بالا می‌گوییم اگر خطا زیر ۲/۵ شد دیگر کافی است. اگر این طور نشد یعنی خطا به اندازه‌ی دلخواهِ ما کم نشده است و حالا می‌توانیم برای تکرار محدودیت بگذاریم. مثلاً می‌گوییم تا ۱۰ هزار مرتبه تکرار را انجام بده (یعنی ۱۰ هزار مرتبه وزن‌ها و انحراف را آپدیت کن) و بعد از آن دیگر یادگیری را ادامه نده.
حال که یادگیری انجام شد، شبکه دارای وزن‌ها و انحرافِ مشخص است. از این به بعد شبکه می‌تواند یک سری ویژگی (مثلاً ویژگی‌های یک پراید یا اتوبوس) را بگیرد و تشخیص دهد که این یک پراید است یا خیر. که البته همان طور که واضح است، این پیش‌بینی دارای خطایی نیز هست.
مثال بالا یک حالت بسیار بسیار ساده فقط با یک وزن بود. در شبکه‌های عصبی که وزن‌های بسیار زیاد، تا ۱۰۰۰ یا بیشتر — با توجه به تعداد ویژگی‌ها یا همان ابعاد، برای به‌هنگام‌سازی وجود دارد سرعتِ روش کاهش گرادیان به خوبی نمایان می‌شود چرا که روشِ پس انتشارِ خطا همراه با کاهشِ گرادیان می‌تواند بسیار سریع نقطه‌ی کمینه‌ی معقولی برای خطا را پیدا کند.

Stocastic Gradien Descent

حال که با روش قبلی تا حد خوبی آشنا شدید یادگیری این روش و روش‌های دیگر کاهش گرادیان بسیار آسان خواهد بود زیرا با منطق اولیه آن آشنا شدیم.

این روش را روش گرادیان کاهشی تصادفی نیز مینامیم. تفاوت اصلی این روش با روش گرادیان کاهشی عادی این است که نقطه ها را به تصادف انتخاب کرده و این عمل باعث می‌شود که این بهینه ساز مزیت‌ها و معایبی را کسب کند. به طور مثال این روش باعث می‌شود که الگوریتم ما با احتمال بسیار کمتری در یک نقطه بهینه محلی گیر کند زیرا نقطه های انتخابی به صورت رندوم میباشند. ولی در کنار آن بدی‌هایی نیز دارد. یکی از مهمترین مشکلات آن تعداد زیاد عملیات مورد نیاز تا رسیدن به نقطه بهینه میباشد. بنابرین شاید استفاده از آن در داده‌هایی که noisy هستند تصمیم مناسبی نباشد. با توجه به داده‌های مورد نظر میتوان در استفاده کردن یا نکردن از این بهینه ساز تصمیم گیری کرد. فرمول این بهینه

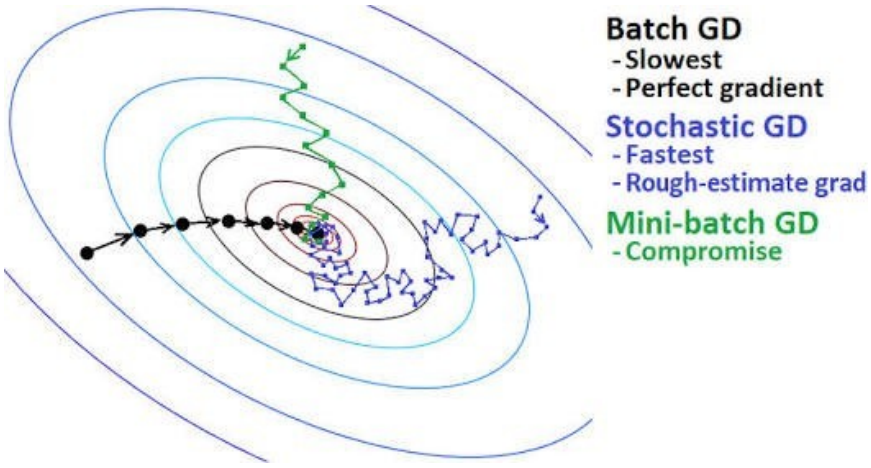
ساز مانند فرمول قبل میباشد باا این تفاوت که به جای استفاده از کل داده تنها از یک پارامتر استفاده میکنیم.

Mini-batch Gradient Descent

این نوع بهینه ساز درواقع ترکیبی از دو بهینه ساز قبلی میباشد.

با توجه به اینکه بهینه ساز اول بسیار کند بود و بهینه ساز دوم نیز در عین سریع بودن مشکلاتی داشت(از قبیل پیدا نکردن مقدار دقیق و تقریبی بودن بیش از حد) این روش ترکیبی به کار گرفته شد. این روش به این شکل میباد که در هر بار استفاده از آن نه از کل داده‌ها استفاده می‌شود و نه از یک داده. بلکه تعدادی از داده‌ها استفاده می‌شود تا حالتی بین دو حالت قبل رخ دهد. یعنی هم سرعت آن از بهینه ساز حالت اول بیشتر خواهد بود و هم جواب دقیق‌تری را نسبت به بهینه ساز دوم به ما خواهد داد.

در شکل پایین میتوانید یک مقایسه اجمالی روی این سه بهینه ساز داشته باشید.



:Momentum

یکی دیگر از روش‌های بهینه سازی در شبکه‌های عصبی استفاده از این روش میباشد.
این روش درواقع همان SGD است با این تفاوت که یک ویژگی به آن اضافه کرده ایم. همانطور که قبل تر اشاره کردیم یکی از نقاط ضعف SGD این است که امکان دارد در داده‌های نویزی به مشکل بر بخورد. یکی دیگر از مشکلات آن این است که ممکن است بخاطر تغییر مسیر های ناگهانی خود به دلیل رندوم بودن مقدار داده انتخابی در هر مرحله از ظرف مقدار بهینه به بیرون پرت شود(آنقدر اختلاف زیاد باشد که دره بهینه را رد کرده و از آن خارج شود).

برای کنترل این حالت یک هایپر پارامتر تعریف کرده و از آن در الگوریتم خود استفاده میکنیم.
و این مقدار در جواب قبلی ضرب می‌شود و به کمک آن جواب بعدی که بدست می‌آید فاصله زیاد ناگهانی با مقدار قبلی نخواهد داشت. مقدار این هایپر پارامتر را معمولاً برابر ۰,۹ ذخیره میکنند.
این موضوع سبب می‌شود تا شتاب نمودار به سمت نقطه مینیمم در هر مرحله بیشتر شود(با در نظر گرفتن مقدارهای قبلی) که باعث می‌شود در بسیاری از موارد زودتر به نقطه مینیمم برسیم.

Repeat Until Convergence {

$$\nu_j \leftarrow \eta * \nu_j - \alpha * \nabla_w \sum_{1}^{m} L_m(w)$$

$$\omega_j \leftarrow \nu_j + \omega_j$$

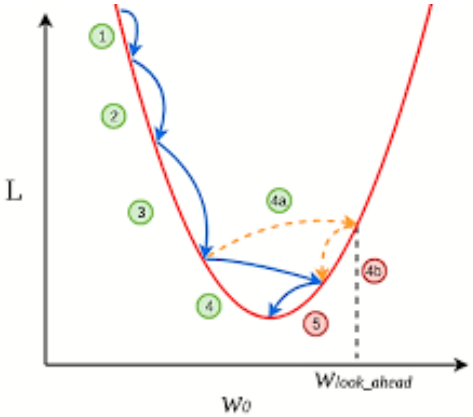
}

:Nestrov Accelerated Gradient

این روش نیز درواقع راه momentum را ادامه داده است. تفاوت این روش با روش قبلی در آن است که به جای آنکه گرادیان برای حالت عادی محاسبه شود(تتا) برای یک مقدار جلوتر رفته آن محاسبه میشود(تتا به علاوه بتا(یک ضریب دلخواه) در m).
منطق این قضیه به این شکل است که در نظر میگیرد بردار تقریباً در جهت درستی قرار دارد پس مقداری از آن هم جلو تر می‌رود و در محاسبه از آن استفاده میکند.

$$v_t = \gamma\, v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$

چیزی که حس می‌شود این است که این الگوریتم در رسیدن به محدوده مینیمم بسیار سریعتر عمل کند اما در رسیدن به نقطه‌ای بسیار نزدیک به مینیمم سرعت کمتری نسبت به حالت قبلی داشته باشد زیرا احتمالاً به دلیل الگوریتمی که دارد بار اول از آن رد خواهد شد. در ادامه با آزمایش این مورد درکد به بررسی آن می‌پردازیم.



همانطور که مشاهده میکنید سرعت رسیدن به نقطه مینیمم در هر مرحله افزایش می‌یابد تا جایی که از آن رد میشود.

سنجش مدل:

بعد از هر مسئله‌ی دسته‌بندی، بر اساس نتیجه‌ی مدل ماتریسی تشکیل داده می‌شود، به نام ماتریس درهم‌ریختگی (*Confusion Matrix*) . فرض کنید مسئله‌ی دسته‌بندی با دو کلاس (برچسب)، ۰ و ۱ داریم. بعد از اینکه مدل را روی داده آموزش دهیم و خروجی مدل را روی داده‌ی آزمایشی بگیریم می‌توان نتیجه را به شکل ماتریس زیر نمایش داد:

Confusion Matrix

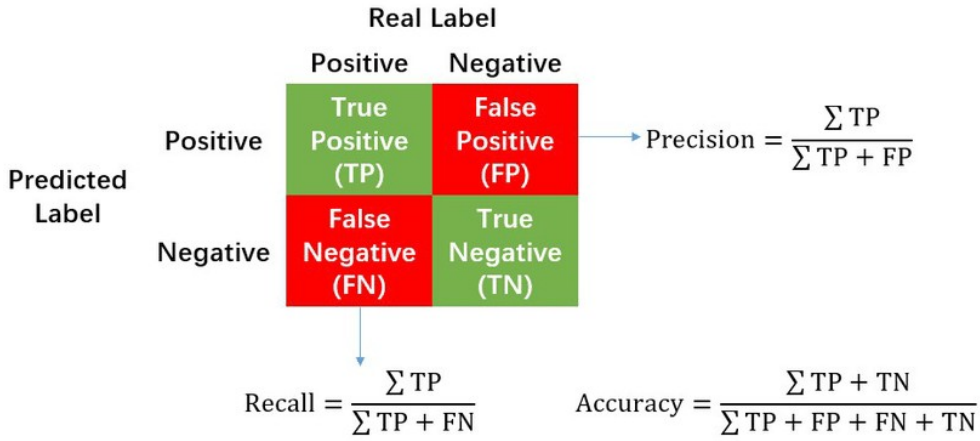
	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

در این ماتریس *True Positives* به پیش‌بینی‌هایی می‌گوییم که برای کلاس مثبت به درستی انجام شده‌اند. به عنوان مثال، مساله‌ی سرطان سینه را در نظر بگیرید. *True Positive* در این مثال به تعداد نمونه‌های از داده می‌گوییم که کلاسی که مدل برای آن‌های پیش‌بینی می‌کند، بدخیم است (*Positive*) و در واقعیت نیز آن‌ها در گروه بدخیم سرطان بوده‌اند (*True*). به همین ترتیب نیز *True Negatives* به نمونه‌هایی که گفته می‌شود در گروه از خوش خیم سانحه پیش‌بینی شده‌اند (*Negative*) و در حقیقت نیز بدخیم بودند(*True*). اما گاهی کلاس پیش‌بینی شده با کلاس حقیقت متفاوت است، در این مواقع اگر در حقیقت نمونه‌ای به کلاس منفی (خوش خیم) تعلق داشته باشد و در کلاس مثبت (بدخیم) پیش‌بینی شود، از عبارت *False Positive* استفاده می‌کنیم در حالت عکس از عبارت *False Negative* استفاده می‌کنیم. حال که با درایه‌های این ماتریس را آشنا شدیم، به معرفی معیارهای دقت (*Accuracy*)، صحت (*Precision*)، یادآوری (*Recall*) و امتیاز (*F1-Score*) *F-1* می‌پردازیم.

دقت شهودی‌ترین اندازه‌ای است که به عملکرد یک مدل می‌توان نسبت داد، و آن نسبت پیش‌بینی‌های درست به کل مشاهدات است.

صحت نسبت پیش‌بینی‌های درست از کلاس مثبت به کل پیش‌بینی‌های مثبت است.

recall نیز نسبت پیش‌بینی‌های درست از کلاس مثبت، به تمام نمونه‌های با کلاس مثبت در داده است.



امتیاز F1 نیز جمع وزن‌داری از دو معیار *صحت* و *یادآوری* است. بنابراین این معیار هر دو مفهوم*False Positive* و *False Negative* را مد نظر قرار می‌دهد. به‌طور شهودی، تعبیر این معیار به سادگی معیاری مثل دقت نیست، اما بیشتر از آن استفاده می‌شود به خصوص زمانی که توزیع کلاس‌های متفاوت یکی نباشد. دقت در حالتی که ما برای*False Positive* و *False Negative* هزینه‌ی یکسانی پرداخت کنیم، معیار خوبی است. منظور از هزینه داشتن، همان عواقب تصمیم‌گیری بر اساس مدل است. فرض کنید مسئله‌ی ما شناسایی ژن موثر در بیماری و در نهایت از بین بردن این ژن است. واضح است که در این مساله *False Positive* هزینه‌ی زیادی برای ما دارد چراکه اگر ژن واقعا در بیماری موثر نباشد، ما در واقع داریم یک ژن سالم را از بین می‌بریم، بنابراین هزینه‌ای که *False Positve* دارد در این مسئله با هزینه‌ی *False Negative* متفاوت است. زمانی که هزینه‌ی این دو متفاوت باشد، بهتر است در سنجش عملکرد مدل هر دو را دخیل کنیم، و در این مسئله‌ها معیار *امتیاز F1** ، *بیشتر استفاده می‌شود*.

$$\text{F1 Score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}$$

روند کد:

ابتدا داده‌ها را دسته بندی کرده و مقداری داده تست را از مجموعه داده هایمان جدا کردیم.

```
data = load_breast_cancer()
```

```
dataframe = pd.DataFrame(data.data,columns=data.feature_names)
dataframe['target'] = data.target
dataframe['target'] = dataframe['target'].apply(lambda x:data.target_names[x])
```

```

```

```

```

```
X = dataframe.drop(columns='target')
```

```
y = dataframe['target'].map({'benign':0,'malignant':1})
```

```
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size = 0.25, stratify = y)
```

در روند پیاده‌سازی شبکه‌های این کد از کتابخانه keras کمک گرفتیم. مدلی که برای این شبکه‌ها پیاده‌سازی شده از نوع sequential بوده است و تمام لایه‌های شبکه از نوع dense می باشند. لایه ابتدایی در این پروژه دارای نود هایی به تعداد ویژگی‌های ورودی می‌باشد که با کمک تابع shape این مقدار را استخراج کردیم. در میانه این شبکه نیز تعدادی لایه برای آنالیز ویژگی‌ها قرار دادیم و در نهایت یک خروجی با اکتیویشن سیگموید برای ارضاء کردن خاصیت دسته بندی استفاده کردیم.

```
]: deep_model = keras.models.Sequential([
    keras.layers.InputLayer(input_shape=X_train_std.shape[1]),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(1, activation="sigmoid")
])
deep_model.save_weights(filepath="./init_weights.h5")
```

برای آنکه مقایسه بین شبکه‌های مختلف به صورت عادلانه تری باشد نیاز بود که وزن های اولیه برای همگی آن‌ها یکسان در نظر گرفته شود. به همین منظور وزن های اولیه را ذخیره کردیم و در هر بار ساختن شبکه جدید و آموزش آن دوباره وزن ها را لود کردیم.

```
deep_model.load_weights("./init_weights.h5")
deep_model.compile(keras.optimizers.SGD(lr=0.001), loss = keras.losses.binary_crossentropy, metrics=["accuracy"])
history['sgd'] = deep_model.fit(X_train_std, y_train, epochs=50)
```

```
Epoch 1/50
14/14 [==============================] - 1s 2ms/step - loss: 0.6837 - accuracy: 0.6600
Epoch 2/50
14/14 [==============================] - 0s 2ms/step - loss: 0.6709 - accuracy: 0.7398
Epoch 3/50
14/14 [==============================] - 0s 2ms/step - loss: 0.6722 - accuracy: 0.7173
Epoch 4/50
14/14 [==============================] - 0s 2ms/step - loss: 0.6662 - accuracy: 0.7305
Epoch 5/50
14/14 [==============================] - 0s 2ms/step - loss: 0.6573 - accuracy: 0.7766
Epoch 6/50
14/14 [==============================] - 0s 2ms/step - loss: 0.6590 - accuracy: 0.7749
Epoch 7/50
14/14 [==============================] - 0s 2ms/step - loss: 0.6521 - accuracy: 0.7941
Epoch 8/50
14/14 [==============================] - 0s 2ms/step - loss: 0.6469 - accuracy: 0.7922
Epoch 9/50
14/14 [==============================] - 0s 2ms/step - loss: 0.6426 - accuracy: 0.8163
```

خوشبختانه در کتابخانه keras تمامی روش‌های بهینه سازی وجود دارد بنابرین تنها کافی بود که در هر بار ساختن شبکه نوع بهینه ساز مورد نظر خود را انتخاب کنیم. این ویژگی باعث شد تا کد نهایی بسیار تمیز باشد.

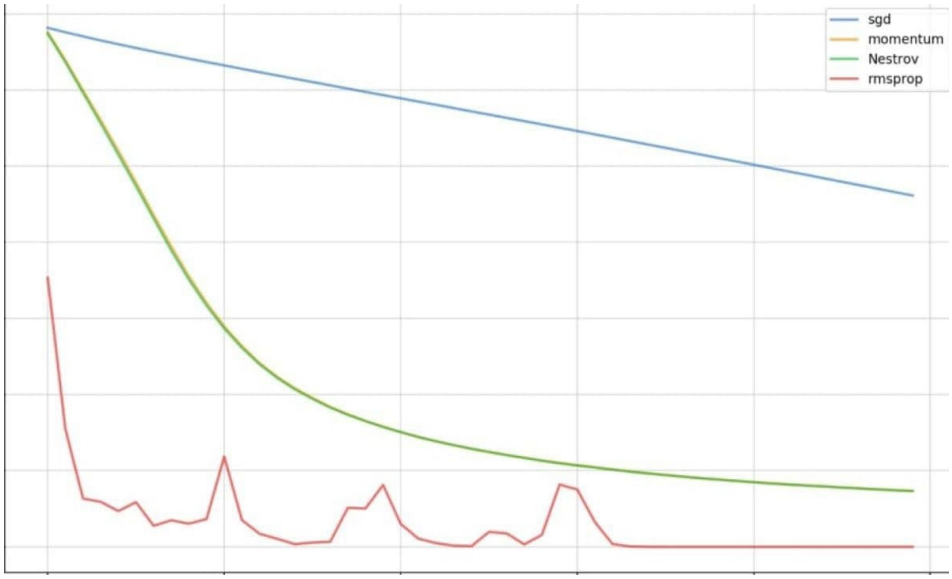
و در آخر عملکرد هر کدام از این شبکه‌ها را ارزیابی نمودیم.

```
print(classification_report(y_test, y_preds_deep))
```

	precision	recall	f1-score	support
0	0.89	0.99	0.94	90
1	0.98	0.79	0.88	53
accuracy			0.92	143
macro avg	0.93	0.89	0.91	143
weighted avg	0.92	0.92	0.91	143

مقایسه داده‌ها و نتیجه گیری:

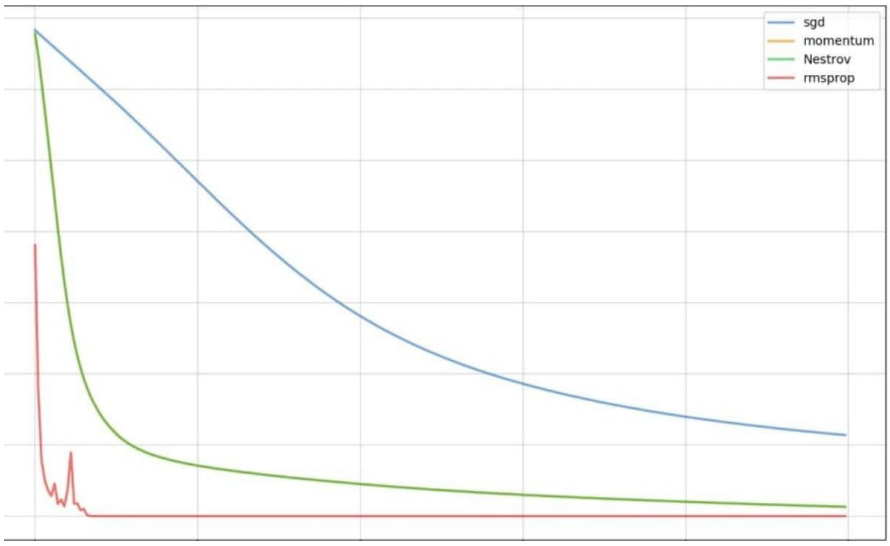
ابتدا نمودار داده‌ها برای epoch = 50 را بررسی میکنیم تا سرعت اولیه بهینه ساز ها را با یکدیگر مقایسه کنیم.



طبق نمودار به نظر میرسد که rmsprop از همان epoch اول اختلاف فاحشی با دو نمودار دیگر دارد و این قضیه بیان میکند که در شروع مقدار loss بسیار کمتری نسبت به بقیه دارد. اما روند پیشرفت آن به شکل نوسانی میباشد اما با این حال در پایان ۵۰ مرحله از بقیه نمودار ها وضعیت بسیار بهتری دارد.
نمودار nestrov به شکل نمایی به نظر میرسد و مقدار پیشرفت بسیار خوبی دارد اما sgd معمولی یک نمودار خطی دارد و در پایان ۵۰ مرحله نتیجه قابل اعتمادی را حاصل نمیکند.

بنابرین به نظر میرسد نمودار های nestrov و rmsprop قابل اعتماد تر میباشند.

حال برای epoch=250 بررسی را انجام میدهیم:



با توجه به نمودار قبلی به نظر میرسد که دو نمودار دیگر بعد از گذشت ۲۵۰ مرحله نیز نتوانسته اند مدل را از بر کنند(قرار ندادن validation data از قصد بوده است برای مقایسه سرعت رسیدن به مرحله بهینه) اما rms در همان epoch های اولیه توانست نمودار را از بر کند. این قضیه نشان میدهد سرعت rms از دو نمودار دیگر بسیار بیشتر بوده است و بیانگر قدرت فوق‌العاده آن در یادگیری مدل در این دیتا میباشد.

منابع:

https://chistio.ir
https://virgool.io/@danialfarsy
hands on machine learning tensorflow 2.0
https://www.youtube.com/watch?v=NB31z-bBUYY
https://www.kdnuggets.com/2020/12/optimization-algorithms-neural-networks.html
https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/
دوره ماشین لرنینگ کویرا