به نام خدا

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

گزارش درس یادگیری ماشین

تمرین دوم: پیاده‌سازی شبکه‌های عصبی چندلایه در پایتون

نام دانشجو:

محمد عرب زاده – ۹۷۲۳۰۵۵

استاد: دکتر سیدین

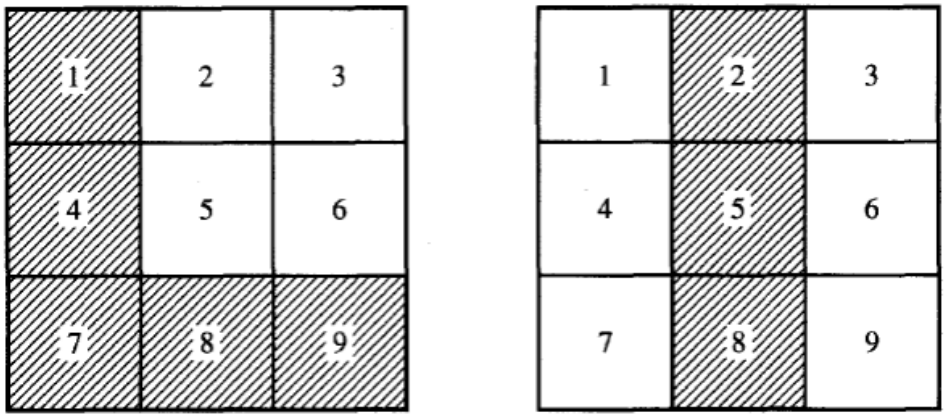
دی ۱۴۰۰

فهرست محتوا

بخش ۱. طراحی پرسپترون برای تشخیص حروف انگلیسی

بخش ۱-۱. شرح مسئله

در این مسئله قصد داریم به کمک یک پرسپترون، حروف L و I انگلیسی را تشخیص دهیم. این حروف همانند شکل ۱ به صورت ماتریس‌های ۳ در ۳ به پرسپترون داده می‌شوند، به طوری که پیکسل‌های تیره مقدار ۱، و پیکسل‌های سفید مقدار ۰ را اختیار کنند.

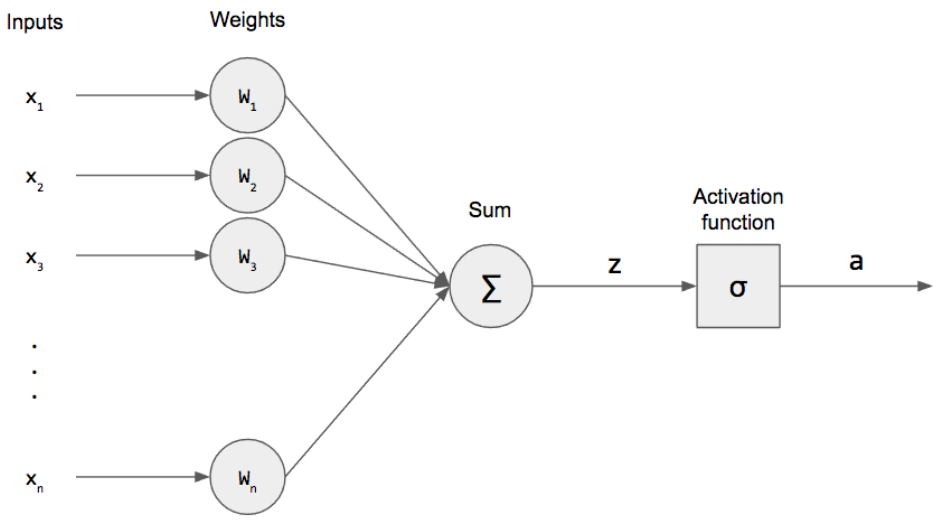


شکل ۱. داده‌های مسئله برای تشخیص به کمک یک پرسپترون

بخش ۱-۲. ساختار پرسپترون

ساختار کلی یک پرسپترون در شکل ۲ نمایش داده شده است. عمل‌کرد پرسپترون به این شکل است که به کمک وزن‌های w_i، یک ترکیب خطی از ورودی‌ها ایجاد می‌کند و سپس این ترکیب خطی را از یک تابع غیرخطی به نام فعال‌ساز[1] عبور می‌دهد.

[1] *Activation Function*



شکل ۲. ساختار یک شبکه‌ی پرسپترون[1]

در مسئله‌ی ما، هر کدام از حروف L و I یک ماتریس دو بعدی هستند؛ این در صورتی است که پرسپترون تنها ورودی‌های تک بعدی دریافت می‌کند. بنابراین لازم است به شیوه‌ای این دو ماتریس دو بعدی را به یک بردار تک بعدی تبدیل کنیم. برای این کار، از تابع *flatten* در کتاب‌خانه‌ی *numpy* استفاده نموده‌ایم؛ طرز کار این تابع به این شکل است که المان‌های ردیف‌های ماتریس را پشت سر هم قرار می‌دهد تا یک بردار تک بعدی ایجاد کند.

به جز ۹ ورودی که از پیکسل‌های هر داده به دست می‌آید، ورودی دیگری نیز به عنوان بایاس، با مقدار ثابت ۱ و وزن w_0 در نظر گرفته می‌شود. در کد پیاده‌سازی شده، به کمک تابع *add_bias*، این مقدار به ورودی اضافه می‌شود.

برای تابع فعال‌ساز، از تابع غیرخطی پله استفاده شده است. تابع *tlu* در کد به همین منظور نوشته شده است.

بخش ۱-۳. نحوه‌ی آموزش شبکه‌ی پرسپترون

با توجه به این که از تابع فعال‌ساز ناپیوسته‌ی پله استفاده نموده‌ایم، نمی‌توان از قانون یادگیری دلتا استفاده نمود و باید به کمک قانون یادگیری پرسپترون شبکه را آموزش دهیم. این قانون در کلاس *PerceptronRule* پیاده‌سازی شده است. رابطه‌ی این قانون یادگیری نیز در معادله (۱) نشان داده شده است.

$$w_i[k+1] = w_i[k] + \eta \cdot (y_j - \hat{y}_j) \cdot \mathrm{x}_j \qquad (۱)$$

در معادله (۱)، (x_j, y_j) ورودی و خروجی مطلوب نمونه‌ی jام داده‌های آموزشی است و $\hat{y}_j$ مقدار پیش‌بینی شده با وزن‌های کنونی. $w_i[k]$ وزن کنونی iام را نشان می‌دهد و $w_i[k+1]$ وزن جدید است. توجه کنید

که وزن‌ها در ابتدا به صورت تصادفی انتخاب می‌شوند؛ به این منظور در کد پیاده‌سازی شده تابع *weight_init* نوشته شده است.

برای این داده‌ها، خروجی مطلوب حرف L را ۰ و حرف I را ۱ در نظر گرفته‌ایم.

بخش ۱-۴. آموزش و ارزیابی

برای آموزش این شبکه، ۵ بار آپدیت وزن‌ها به کمک دو داده‌ی موجود انجام شده است. طبق شکل ۳ می‌بینیم که خطا به صفر رسیده است. در شکل ۴ نیز می‌بینیم که هر دو داده به درستی پیش‌بینی شده است.

```
############ Training Process Started ############
---- Epoch 1
Error = 1
---- Epoch 2
Error = 0
---- Epoch 3
Error = 0
---- Epoch 4
Error = 0
---- Epoch 5
Error = 0
############## Training Process Ended #############
```

شکل ۳. خطای مدل حین یادگیری

```
idx   X                       Target  Pred
=======================================
0     [1 0 0 1 0 0 1 1 1]   0       0
1     [0 1 0 0 1 0 0 1 0]   1       1
=======================================

Accuracy = 1.0
```

شکل ۴. پیش‌بینی داده‌های آموزشی

در مرحله‌ی بعد، ابتدا ۲۵ درصد و سپس ۵۰ درصد پیکسل‌ها را به صورت تصادفی عوض می‌کنیم؛ این الگوریتم در تابع *add_noise* نوشته شده است. حال این داده‌های نویزی را به کمک همان شبکه پیش‌بینی می‌کنیم.

طبق شکل ۵ می‌بینیم که در صورتی که ۲۵ درصد نویز اضافه شده باشد، شبکه به درستی عمل می‌کند؛ اما طبق شکل ۶، زمانی که ۵۰ درصد نویز اضافه شود، صحت شبکه به صفر می‌رسد.

```
idx   X                       Target  Pred
=======================================
0     [0 0 0 0 0 0 1 1 1]   0       0
1     [0 1 1 0 1 0 0 1 1]   1       1
=======================================

Accuracy = 1.0
```

شکل ۵. پیش‌بینی داده‌ها با ۲۵ درصد نویز

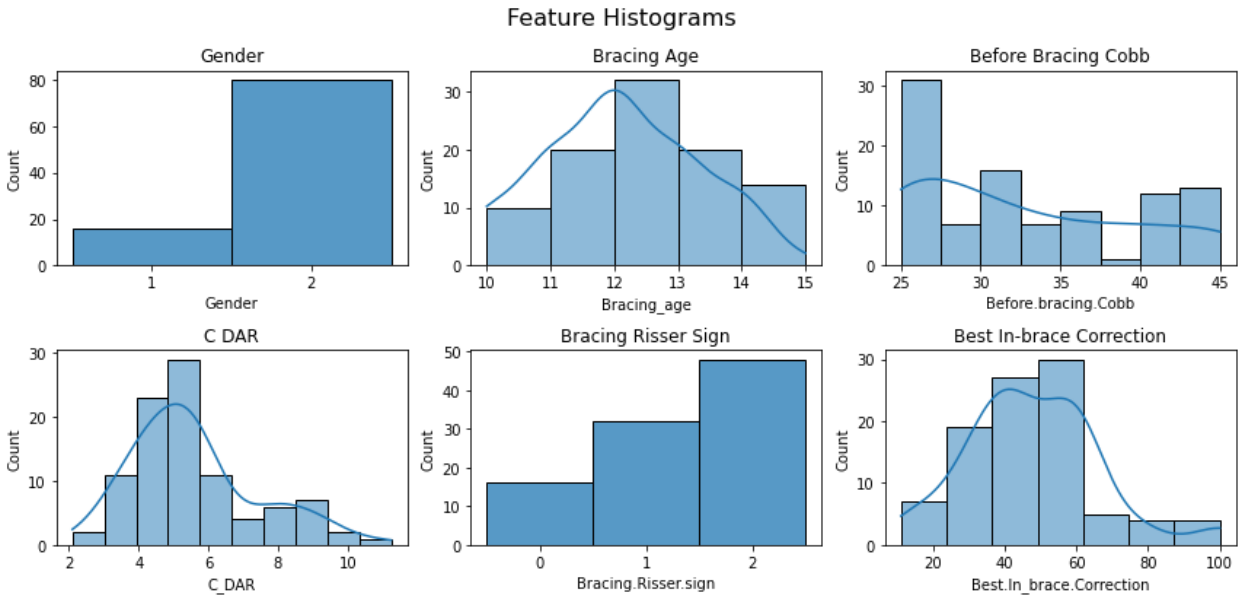
```
idx   X                       Target  Pred
=======================================
0     [1 1 1 0 0 0 0 1 1]   0       1
1     [0 0 0 0 0 0 0 1 0]   1       0
=======================================

Accuracy = 0.0
```

شکل ۶. پیش‌بینی داده‌ها با ۵۰ درصد نویز

بخش ۲. پیش‌بینی تاثیر درمان بیماری به کمک شبکه‌ی عصبی پرسپترون چندلایه[۲]

بخش ۲-۱. بررسی داده‌ها

در این دیتاست ۶ متغیر ویژگی[۳] و ۲ متغیر هدف[۴] وجود دارد. دو متغیر هدف، *Outcome* و *Two Years Follow-up Cobb* نام دارند که اولی برچسب[۵]های صفر و یک دارد و دومی دارای مقادیری بین ۱ تا ۱۰۰ است. بنابراین برای متغیر *Outcome* باید طبقه‌بندی[۶] انجام دهیم و برای *Two Years Follow-up Cobb* باید رگرسیون[۷] انجام دهیم.



شکل ۷. هیستوگرام هر یک از ویژگی‌های موجود در دیتاست

در نمودارهای شکل ۷، هیستوگرام[۸] هر یک از ویژگی‌ها را رسم نموده‌ایم. مشاهده می‌شود که پراکندگی مقادیر این ویژگی‌ها یک‌نواخت نیست؛ این موضوع ممکن است باعث اُورفیت[۹] شدن مدل شبکه‌ی عصبی شود.

---

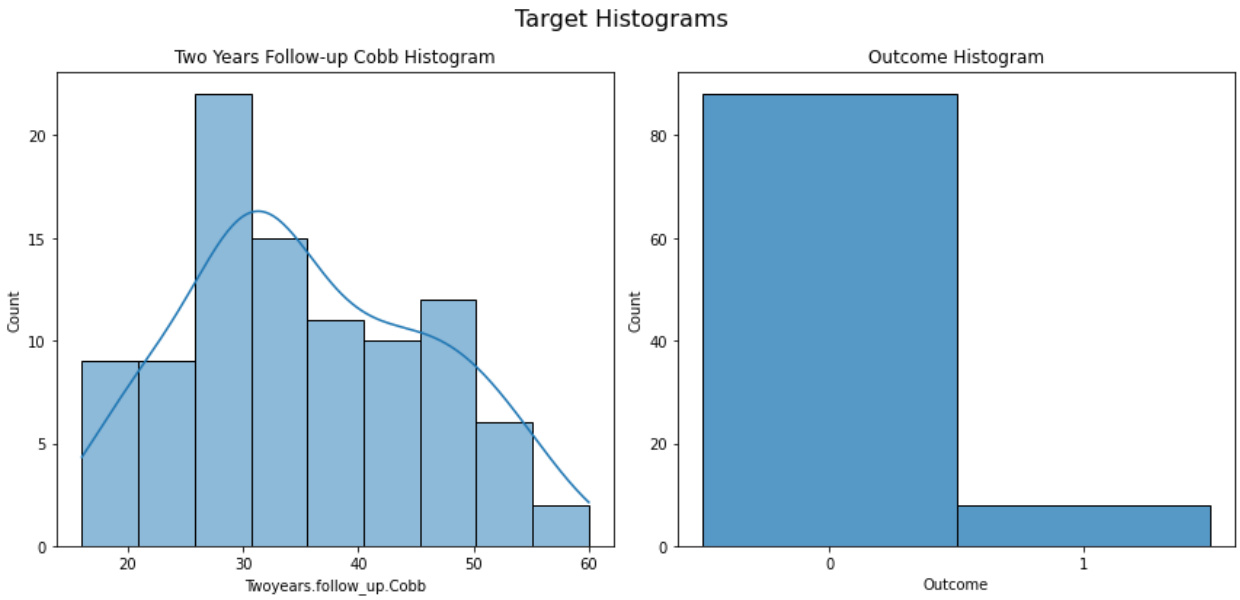
[2] *Multi-Layer Perceptron Neural Network*
[3] *Feature Variable*
[4] *Target Variable*
[5] *Label*
[6] *Classification*
[7] *Regression*
[8] *Histogram*
[9] *Overfit*

برای مثال متغیر *Gender* را در نظر بگیرید؛ تقریبا ۸۰ درصد داده‌ها مربوط به *Gender=2* بوده‌اند. طبق شکل ۸ نیز می‌بینیم که تنها یک داده با *Gender=1* در میان داده‌هایی که *Outcome* آن‌ها برابر یا ۱ است وجود دارد. این موضوع ممکن است باعث ایجاد مدلی شود که *Gender=1* را هیچ‌گاه درون کلاس *Outcome=1* قرار ندهید. این در حالی است که ممکن است کم بودن تعداد *Gender=1* به دلیل کوچک بودن سایز دیتاست باشد؛ در این صورت مدل بر روی داده‌های آموزش به خوبی عمل خواهد کرد، اما در تست‌های واقعی عمل‌کرد پایینی خواهد داشت.

	Gender	Bracing_age	Before.bracing.Cobb	C_DAR	Bracing.Risser.sign	Best.In_brace.Correction	Twoyears.follow_up.Cobb	Outcome
33	2	13	45	4.30	1	33.333333	55	1
35	1	11	45	11.25	0	15.555556	55	1
37	2	12	45	7.50	1	11.111111	55	1
38	2	12	40	10.00	1	17.500000	55	1
60	2	12	40	8.00	2	20.000000	52	1
76	2	11	30	6.00	1	33.333333	53	1
89	2	10	45	5.60	0	11.111111	60	1
91	2	10	30	10.00	0	33.333333	56	1

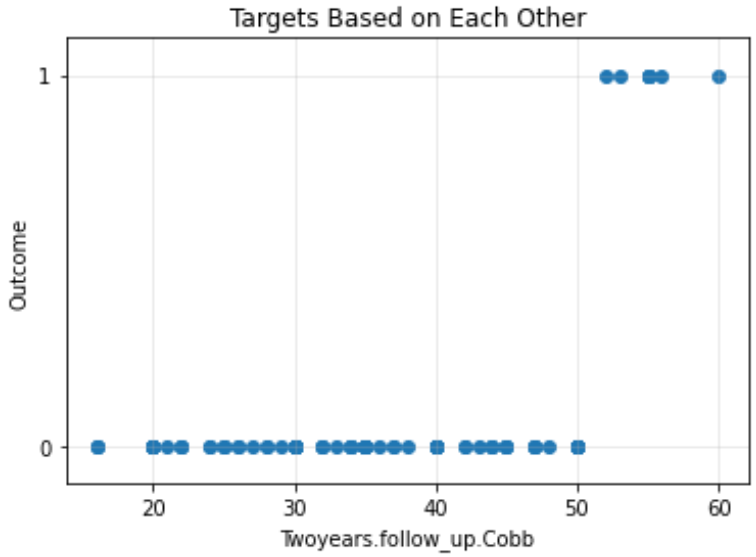
شکل ۸. جدول داده‌هایی که دارای *Outcome=1* هستند

دو مشکل اصلی این دیتاست، کوچک بودن سایز آن و غیربالانس بودن توزیع متغیرهای هدف است. دیتاست تنها شامل ۹۶ داده است که برای یک شبکه‌ی عصبی، عددی بسیار کوچک است. همچنین طبق شکل ۹، مشاهده می‌شود که تنها ۸ داده در کلاس *Outcome=1* قرار دارند و توزیع مقادیر متغیر *Two Years Follow-up Cobb* نیز غیریکنواخت است.

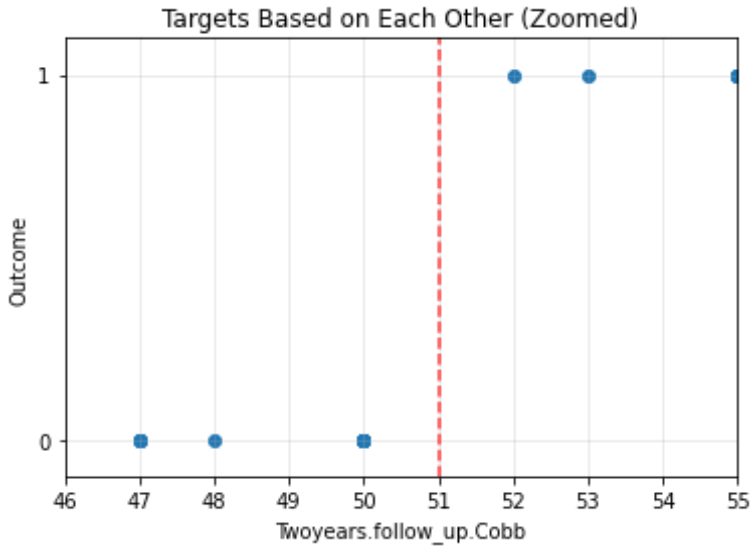


شکل ۹. هیستوگرام متغیرهای هدف

برای بررسی وجود رابطه‌ای میان دو متغیر هدف، این دو متغیر را در شکل ۱۰ بر حسب یک‌دیگر رسم نموده‌ایم. در این شکل می‌بینیم که متغیر *Outcome* به صورت خطی به کمک متغیر *Two Years Follow-up Cobb* جداپذیر است! شکل ۱۱ که حاصل بزرگ‌نمایی نمودار شکل ۱۰ است، نشان می‌دهد که همه‌ی داده‌هایی که *Two Years Follow-up Cobb* آن‌ها کوچک‌تر از ۵۱ است متعلق به *Outcome=0* و بقیه متعلق به *Outcome=1* هستند.



شکل ۱۰. مقدار متغیرهای هدف بر حسب یک‌دیگر

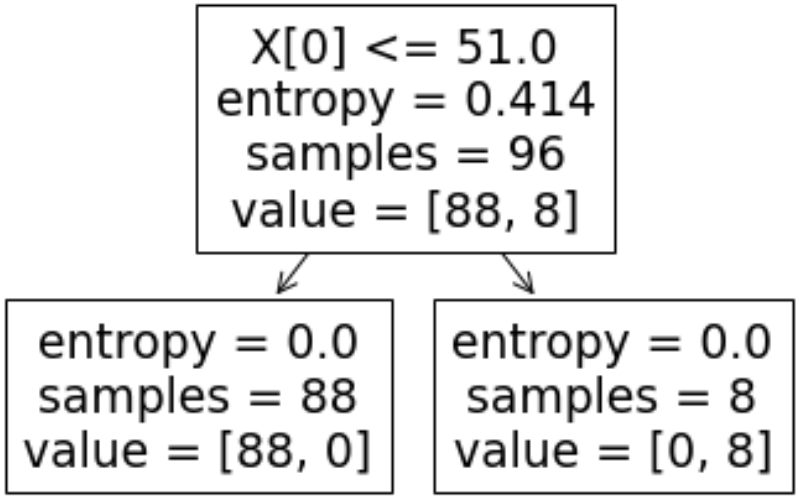


شکل ۱۱. نمودار بزرگ‌نمایی شده‌ی شکل ۱۰. خط‌چین قرمز نشان می‌دهد که این دو متغیر جدا پذیر خطی هستند.

با این که ممکن است کم بودن تعداد داده‌ها باعث شده باشد که دو متغیر هدف *Outcome* وابسته به متغیر دیگر باشد، اما تنها با داشتن این داده‌ها نمی‌توان نتیجه‌ی دیگری گرفت. بنابراین به نظر می‌رسد ساده‌ترین روش این است که مدلی برای رگرسیون *Two Years Follow-up Cobb* طراحی کنیم و سپس بر اساس نتیجه‌ی آن، مقدار *Outcome* را تعیین کنیم.

در شکل ۱۲ مشاهده می‌شود که با داشتن مقدار *Two Years Follow-up Cobb* می‌توان تنها با یک گره ریشه[۱۰]، به کمک الگوریتم درخت تصمیم[۱۱] مقدار *Outcome* را به دست آورد. مشاهده می‌شود که مقدار آنتروپی در برگ‌ها، به صفر مطلق رسیده است؛ این به آن معناست که داده‌های موجود در این گره‌ها کاملا خالص هستند و با صحت صد درصد طبقه‌بندی شده‌اند.

[10] *Root Node*
[11] *Decision Tree Classifier*



شکل ۱۲. استفاده از درخت تصمیم برای پیش‌بینی متغیر *Outcome* بر اساس متغیر *Two Years Follow-up Cobb*

در این پروژه، برای به دست آوردن مقدار *Outcome* تنها یک شرط آستانه بر روی مقدار *Two Years Follow-up Cobb* که توسط مدل شبکه‌ی عصبی پیش‌بینی می‌شود، اعمال شده است.

بخش ۲-۲. پیش‌پردازش داده‌ها

یکی از موارد پیش‌پردازش برای این دیتاست مربوط به ویژگی *Bracing Risser Sign* است. همان‌طور که در شکل ۷ دیدیم، مقادیری که این ویژگی دریافت می‌کند، مقادیر گسسته ۰، ۱ و ۲ است. برای عمل‌کرد بهتر شبکه‌ی عصبی، بهتر است چنین داده‌هایی را به صورت *One-hot* کدگذاری کنیم.

داده‌های ویژگی *Gender* نیز مقادیر گسسته‌ی ۱ و ۲ را دارند؛ اما از آنجایی که تنها دو دسته هستند، می‌توان آن‌ها را با صفر و یک نشان داد؛ بنابراین مقادیر ۱ و ۲ آن را به ۰ و ۱ نگاشت می‌کنیم.

بر روی ویژگی‌های دیگر، به همراه متغیر هدف *Two Years Follow-up Cobb* لازم است نرمال‌سازی[۱۲] انجام شود. روش‌های مختلفی برای نرمال‌سازی داده‌ها وجود دارد؛ در این پروژه برای نرمال‌سازی داده‌ها، مقادیر آن‌ها را بین صفر و یک نگاشت نموده‌ایم.

در نهایت، پس از پیش‌پردازش داده‌ها، مقادیر تعدادی از آن‌ها به صورت شکل ۱۳ در آمده است.

[12] *Normalization*

	Gender	Bracing_age	Before.bracing.Cobb	C_DAR	Best.In_brace.Correction	Bracing.Risser.sign_0	Bracing.Risser.sign_1	Bracing.Risser.sign_2	Twoyears.follow_up.Cobb	Outcome
47	1	0.8	0.45	0.382514	0.338235	0	0	1	0.545455	0
26	0	0.8	0.25	0.316940	0.250000	0	0	1	0.204545	0
73	0	0.2	0.25	0.316940	0.475000	0	1	0	0.318182	0
76	1	0.2	0.25	0.426230	0.250000	0	1	0	0.840909	1
82	1	0.4	0.15	0.382514	0.517857	0	0	1	0.318182	0

شکل ۱۳. مقادیر تعدادی از داده‌ها پس از پیش‌پردازش

بخش ۲-۳. طراحی مدل رگرسیون

همان‌طور که در بخش ۲-۱ گفته شد، برای حل این مسئله کافیست تنها برای متغیر رگرسیون شبکه‌ای طراحی کنیم که بتواند با خطای کم، مقدار *Two Years Follow-up Cobb* را تشخیص دهد. دقت این مدل، بر عمل‌کرد آن در تعیین *Outcome* نیز تاثیر مستقیم دارد؛ بنابراین لازم است دقیق‌ترین مدل رگرسیون ممکن را برای این مجموعه‌ی داده‌ها طراحی کنیم.

در ادامه ابتدا به جست‌وجوی ابرپارامتر[۱۳]های مناسب، به کمک *k-Fold Cross Validation* می‌پردازیم و سپس عمل‌کرد مدل نهایی را بررسی می‌کنیم.

پیش از شروع طراحی مدل، لازم است ابتدا روشی مناسب برای ارزیابی و مقایسه‌ی مدل‌ها پیدا کنیم. با توجه به حجم کم داده‌ها، بهتر است از روش *k-Fold Cross Validation* استفاده نماییم؛ با استفاده از این روش، می‌توانیم میانگین عمل‌کرد مدل را بر روی داده‌هایی که قبلا توسط مدل دیده نشده‌اند به دست آوریم. هنگام استفاده از این روش، دو سوال پیش می‌آید؛ جداسازی داده‌ها در هر *Fold* به چه شکلی باشد؟ و مقدار *k* چه عددی باشد؟

بخش ۲-۳-۱. نحوه‌ی استفاده از *k-Fold Cross Validation*

در نمودارهای شکل ۹ دیدیم که توزیع داده‌های هر دو متغیر هدف، غیریکنواخت هستند؛ نکته‌ی دیگری که از این نمودارها و نمودار شکل ۱۱ به دست می‌آید این است که داده‌های کلاس ۱ متغیر *Outcome*، متناظر با داده‌هایی با مقدار بزرگ‌تر از ۵۱ در متغیر *Two Years Follow-up Cobb* هستند. برای این که تا حدودی این عدم یک‌نواختی داده‌های کلاس ۱ متغیر *Outcome* (و داده‌های بزرگ‌تر از ۵۱ متغیر *Two Years Follow-up Cobb*) جبران شود، می‌توان جداسازی داده‌ها در هر *Fold* را طوری انجام داد که مطمئن باشیم حداقل یک داده از کلاس ۱ متغیر *Outcome* در *Validation* حضور داشته باشد. به این منظور از کلاس

[13] *Hyper-parameter*

StratifiedKFold موجود در کتاب‌خانه‌ی *sklearn* استفاده نموده‌ایم و داده‌ها را با هدف عدم تغییر توزیع کلاس‌های متغیر *Outcome* جداسازی نموده‌ایم.

همان‌طور که اشاره شد، در متغیر *Outcome*، ۸ داده متعلق به کلاس ۱ هستند. در همه‌ی داده‌های مربوط به *Validation* تعداد داده‌های کلاس ۱ برابر شوند، باید مقدار k مضربی از ۸ باشد. به منظور نزدیک‌شدن *k-Fold* به بهترین حالت ممکن، یعنی *Leave-one-out Cross Validation*، بیش‌ترین مقدار ممکن، یعنی ۸ را برای k در نظر می‌گیریم. در این حالت، در همه‌ی *Fold*ها دقیقا ۱ داده از کلاس ۱ متغیر *Outcome* در *Validation* وجود دارد.

همچنین به منظور یکسان شدن جداسازی در هر بار اجرای کد، یک مقدار ثابت برای *seed* جداساز در نظر گرفته شده است.

بخش ۲-۳-۲. تابع هزینه و معیار ارزیابی

یکی از توابع هزینه‌ی پر استفاده برای رگرسیون، تابع *MSE*[14] است که در معادله (۲) نشان داده شده است؛ در این پروژه نیز از همین تابع به عنوان تابع هزینه استفاده شده است.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad (۲)$$

برای ارزیابی و مقایسه‌ی مدل‌ها نیز از دو معیار *RMSE*[15] و R^2 *Score* استفاده نموده‌ایم؛ این دو معیار در معادله‌های (۳) و (۴) نشان داده شده‌اند. در معادله (۴)، $\hat{y}_i$ مقدار پیش‌بینی نمونه‌ی iام و $\bar{y}$ مقدار میانگین متغیرهای هدف را نشان می‌دهد.

$$RMSE = \sqrt{MSE} \qquad (۳)$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \qquad (۴)$$

دلیل استفاده از *RMSE* بجای *MSE* برای ارزیابی این است که *RMSE* تفکیک مقادیر بین صفر و یک را بهتر نشان می‌دهد؛ با توجه به این که مقادیر خطا معمولا کم‌تر از یک است، از *RMSE* استفاده شده است.

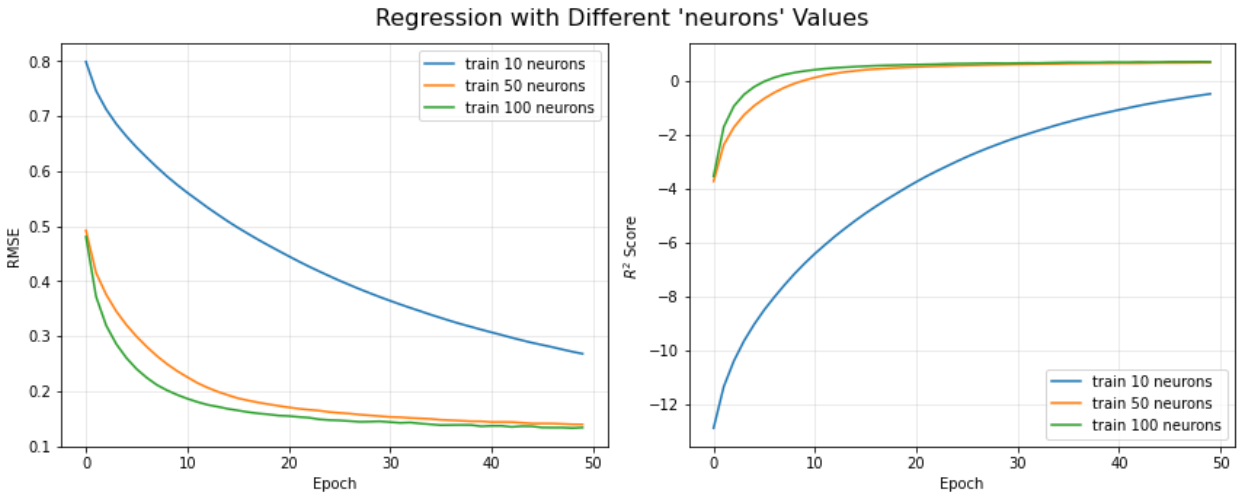
[14] *Mean Squared Error*
[15] *Root Mean Squared Error*

معیار R^2 *Score* نیز نشان می‌دهد که مدل ما به چه میزان از یک خط افقی پیش‌بینی بهتری دارد؛ اگر مقدار این معیار کم‌تر از صفر شود، یعنی مدل ضعیف‌تر از یک خط افقی عمل کرده است و اگر بیش‌تر از صفر شود، یعنی بهتر از خط افقی بوده است. بهترین R^2 *Score* زمانی است که مقدار آن برابر با ۱ باشد.

برای مقایسه‌ی مدل‌ها حین جست‌وجوی ابرپارامترها، نمودار میانگین *MSE* و R^2 *Score* بین هر هشت *Cross Validation* برای هر مدل بر حسب *Epoch* رسم شده است.

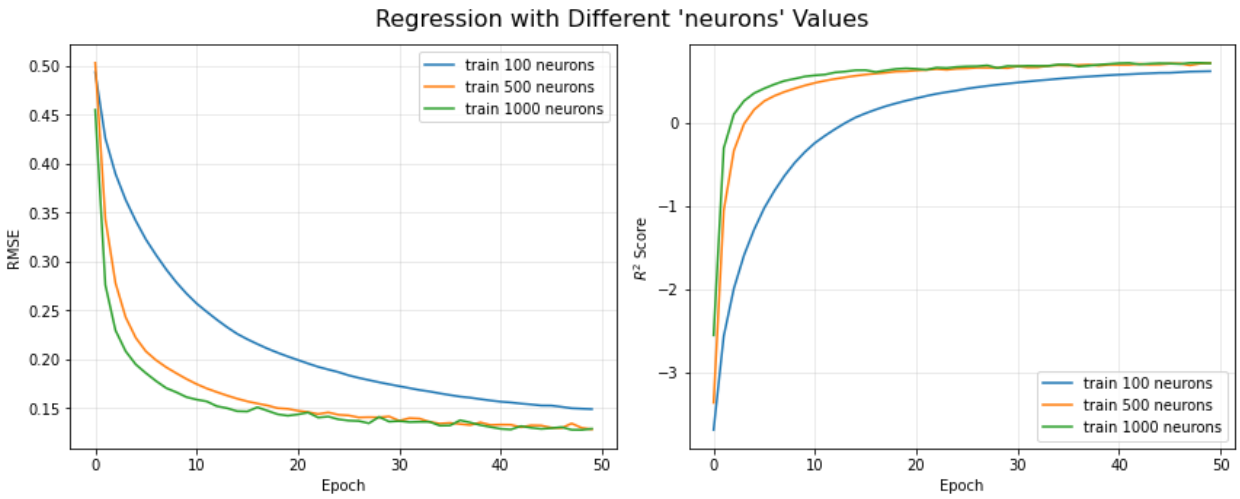
بخش ۲-۳-۳. بررسی عمل‌کرد شبکه با یک لایه‌ی مخفی

ابتدا از یک شبکه با یک لایه‌ی مخفی شروع می‌کنیم و پارامترهای مناسب برای این شبکه را می‌یابیم. ابتدا تعداد نورون‌های مختلف را بررسی می‌کنیم؛ نمودارهای شکل ۱۴ عمل‌کرد شبکه را بر روی داده‌های آموزشی برای تعداد ۱۰، ۵۰ و ۱۰۰ نورون نشان می‌دهد. مشاهده می‌کنیم که افزایش تعداد نورون‌ها باعث بهبود عمل‌کرد شبکه می‌شود؛ به منظور یافتن مقدار بهینه برای تعداد نورون‌ها، باز هم مقدار آن را افزایش داده‌ایم و در نمودار شکل ۱۵ نشان داده‌ایم.



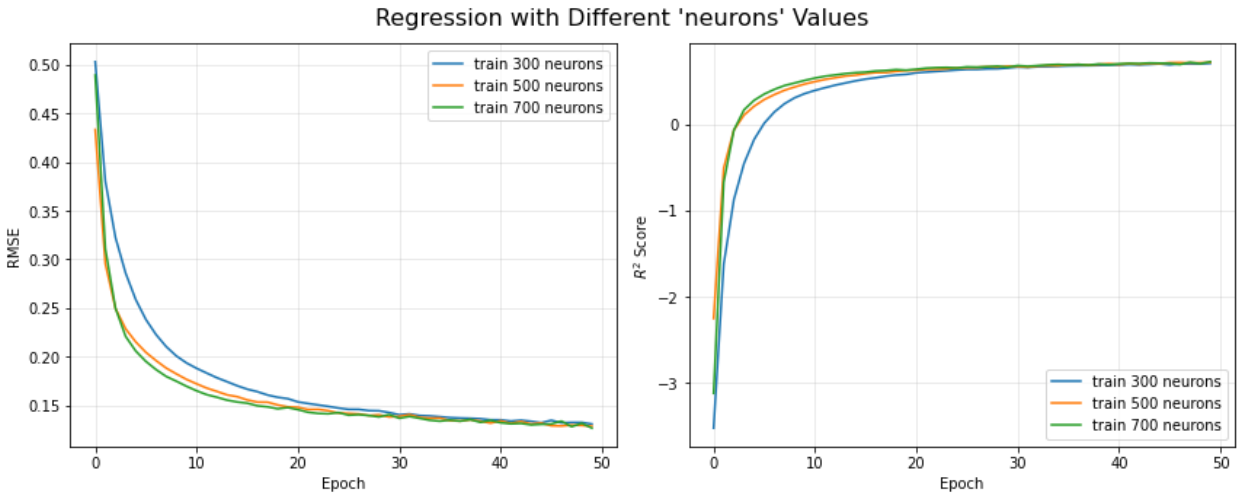
شکل ۱۴. مقایسه‌ی تعداد نورون‌های ۱۰، ۵۰ و ۱۰۰ در شبکه‌ای با یک لایه‌ی مخفی

طبق شکل ۱۵ مشاهده می‌کنیم که ۱۰۰۰ نورون با ۵۰۰ نورون پس از حدود ۲۰ *Epoch*، عمل‌کرد نسبتا مشابهی دارد.



شکل ۱۵. مقایسه‌ی تعداد نورون‌های ۱۰۰، ۵۰۰ و ۱۰۰۰ در شبکه‌ای با یک لایه‌ی مخفی

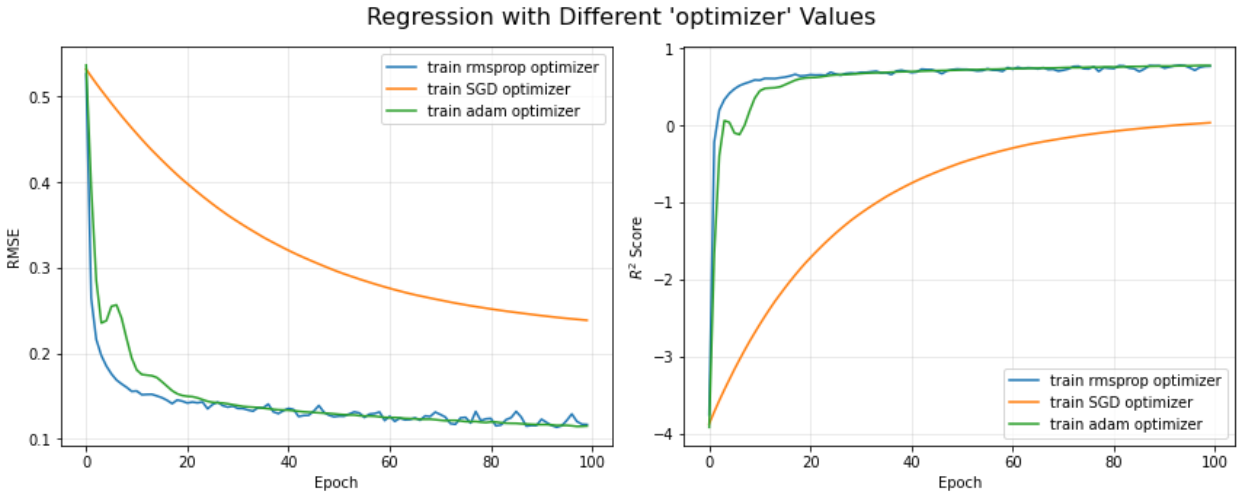
برای اطمینان از این که ۵۰۰ نورون مقدار بهینه‌ای است، آن را با مقدار ۳۰۰ و ۷۰۰ نورون نیز در شکل ۱۶ مقایسه نموده‌ایم. مشاهده می‌شود که هر سه پس از ۵۰ *Epoch* به یک مقدار می‌رسند؛ اما مدل‌های ۵۰۰ و ۷۰۰ نورونه سریع‌تر به این مقدار می‌رسند. با این حال سرعت هم‌گرایی مدل ۵۰۰ و ۷۰۰ نورونه نسبتا به یک‌دیگر نزدیک است؛ بنابراین تعداد کم‌تر، یعنی ۵۰۰ نورون را برای این شبکه انتخاب نمودیم.



شکل ۱۶. مقایسه‌ی تعداد نورون‌های ۳۰۰، ۵۰۰ و ۷۰۰ در شبکه‌ای با یک لایه‌ی مخفی

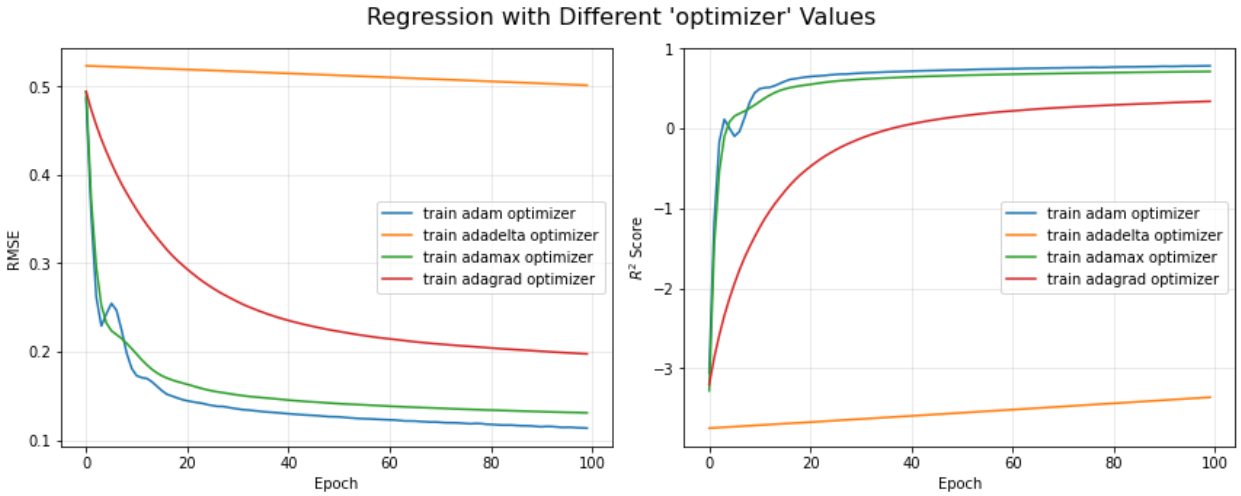
در مرحله‌ی بعد، *Optimizer*های *rmsprop*، *SGD* و *Adam* برای این شبکه با یک‌دیگر مقایسه شدند. طبق شکل ۱۷ می‌بینیم که *Adam* و *rmsprop* نتایجی نزدیک به هم دارند و *SGD* بسیار کند بوده است. با توجه

به این که نوسانات نمودار یادگیری برای *Adam* کم‌تر از *rmsprop* بوده است، این *Optimizer* را انتخاب نمودیم.



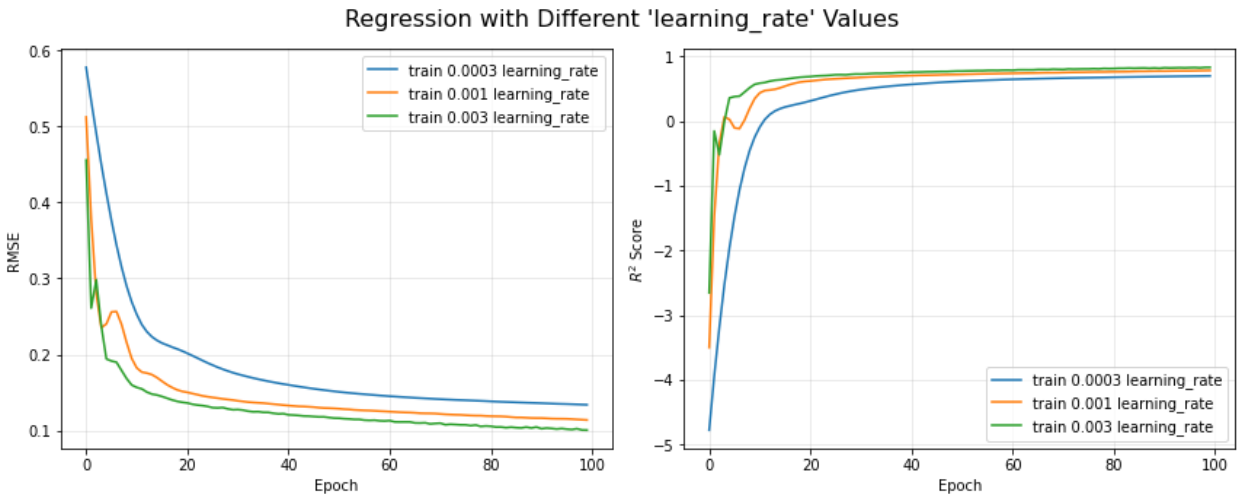
شکل ۱۷. مقایسه‌ی *Optimizer*های *rmsprop*، *SGD* و *Adam* در شبکه‌ای با یک لایه‌ی مخفی

برای اطمینان از این که *Adam* بهترین *Optimizer* برای این شبکه است، *Optimizer*های مشابه با *Adam*، یعنی *AdaDelta*، *AdaMax* و *AdaGrad* نیز در شکل ۱۸ با *Adam* مقایسه شده‌اند. مشاهده می‌شود که همچنان *Adam* بهترین عمل‌کرد را دارد.



شکل ۱۸. مقایسه‌ی *Optimizer*های *Adam*، *AdaDelta*، *AdaMax* و *AdaGrad* در شبکه‌ای با یک لایه‌ی مخفی

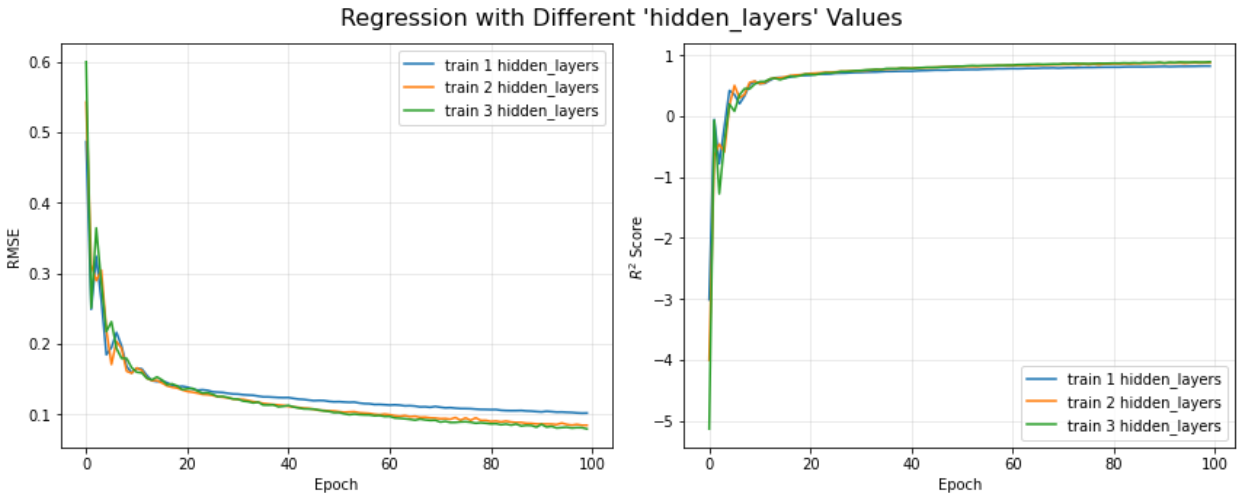
برای یافتن بهترین نرخ یادگیری، مقادیر ۰.۰۰۰۳، ۰.۰۰۱ و ۰.۰۰۳ با یک‌دیگر مقایسه شدند. طبق شکل ۱۹، مشاهده می‌شود که ۰.۰۰۳ بهترین نرخ یادگیری است.



شکل ۱۹. مقایسه‌ی نرخ یادگیری متفاوت در شبکه‌ای با یک لایه‌ی مخفی

بخش ۲-۳-۴. افزایش تعداد لایه‌های مخفی

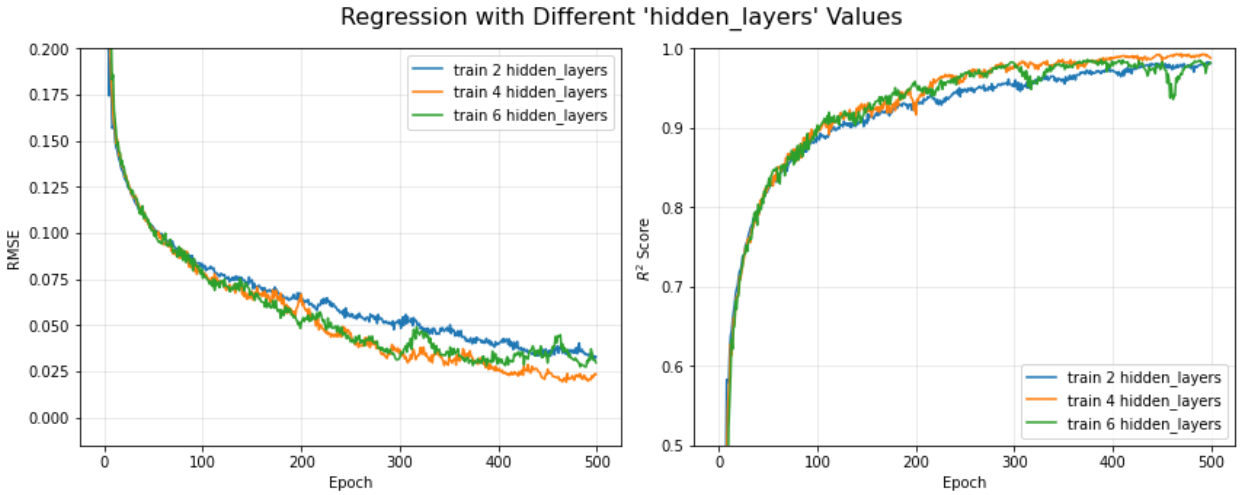
در این بخش لایه‌های مخفی بیش‌تری به مدل اضافه می‌کنیم و عمل‌کرد مدل را بررسی می‌کنیم. برای سادگی کار، تعداد نورون‌ها در همه‌ی لایه‌ها ثابت، و برابر با همان مقدار ۵۰۰ در نظر گرفته شده است.



شکل ۲۰. مقایسه‌ی تعداد لایه‌های مخفی ۱، ۲ و ۳ عدد

طبق نمودار $RMSE$ در شکل ۲۰، مشاهده می‌شود که تعداد لایه‌های مخفی بیش از ۱، عمل‌کرد مدل را به میزان قابل توجهی بهبود می‌بخشد. برای یافتن بهترین تعداد لایه، مقادیر ۲، ۴ و ۶ برای تعداد لایه‌های مخفی در شکل ۲۱ با یک‌دیگر مقایسه شده‌اند. مشاهده می‌شود که با ۶ لایه، پس از مدتی نمودار واگرا می‌شود و احتمالا به نرخ

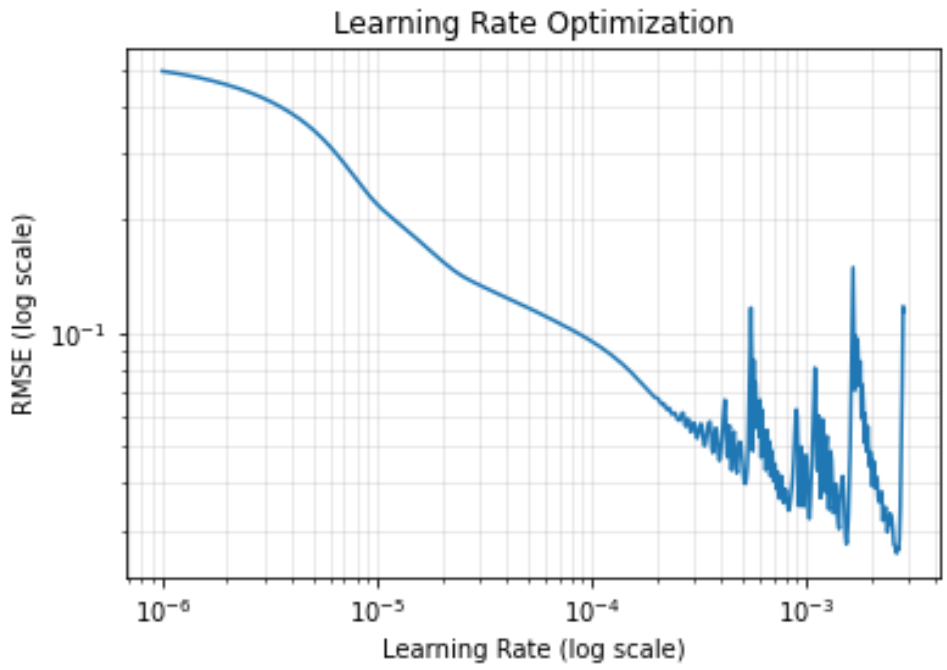
یادگیری پایین‌تری نیاز دارد؛ در نتیجه آموزش آن کندتر خواهد بود. با توجه به بهبود قابل توجه شبکه‌ی ۴ لایه نسبت به ۲ لایه، این تعداد لایه را انتخاب نمودیم.



شکل ۲۱. مقایسه‌ی تعداد لایه‌های مخفی ۲، ۴ و ۶ عدد

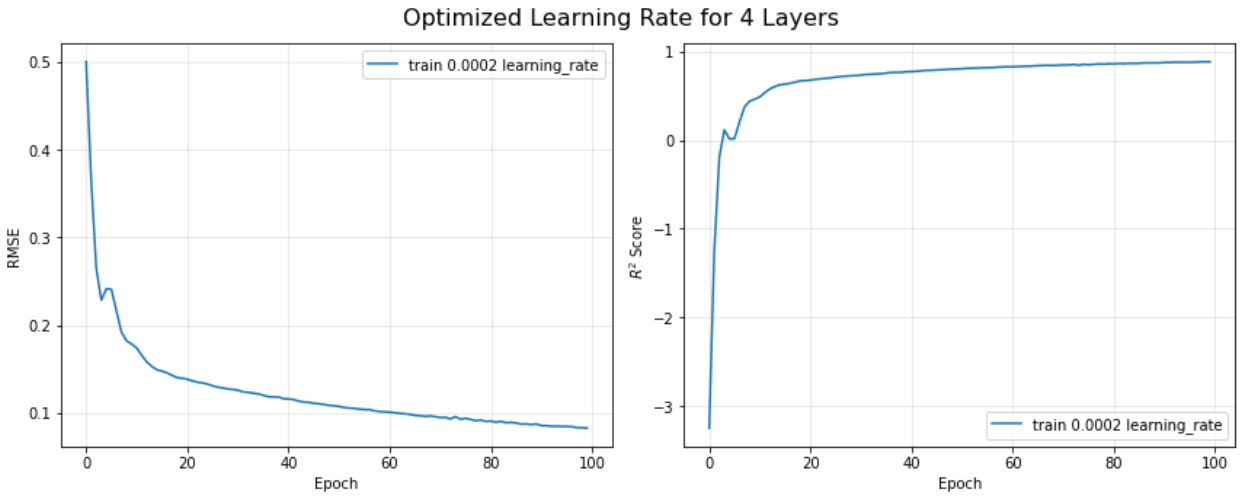
به منظور یافتن بهترین نرخ یادگیری، از روش تست بازه‌ای[۱۶] استفاده نمودیم[2]. در این روش، از یک نرخ یادگیری کوچک شروع می‌کنیم و به تدریج در هر *Iteration* مقدار آن را اندکی افزایش می‌دهیم؛ این کار را تا زمانی که عمل‌کرد مدل شروع به واگرایی کند ادامه می‌دهیم. سپس نمودار نرخ یادگیری را بر حسب معیار ارزیابی مدل رسم می‌کنیم.

[16] *Learning Rate Range Test*



شکل ۲۲. اجرای تست بازه‌ای نرخ یادگیری بر روی مدل کنونی

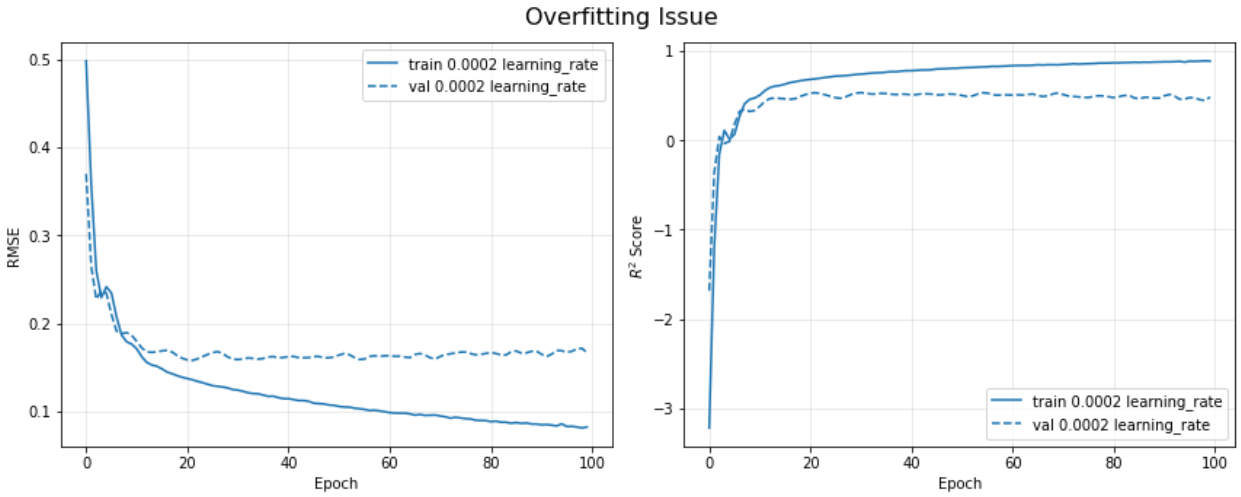
طبق *شکل ۲۲* مشاهده می‌شود که نرخ یادگیری بیش از حدود $3 \cdot 10^{-4}$ باعث واگرایی *RMSE* مدل می‌شود. شکل ۲۳ عمل‌کرد مدل را با نرخ یادگیری $2 \cdot 10^{-4}$ بر روی داده‌های آموزش نشان می‌دهد. می‌بینیم که مدل به خوبی عمل کرده است و R^2 *Score* آن به نزدیکی ۱ رسیده است.



شکل ۲۳. عمل‌کرد مدل با نرخ یادگیری بهینه

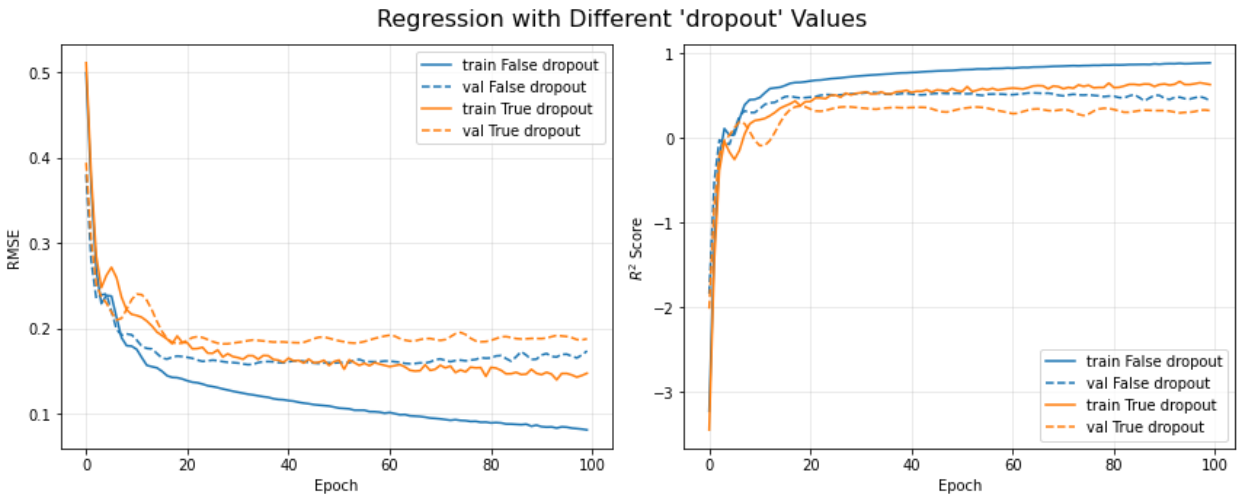
بخش ۲-۳-۵. مشکل *Overfitting*

نمودارهای شکل ۲۴ عمل‌کرد مدل بر روی داده‌های *Validation* را نیز نشان می‌دهد. مشاهده می‌کنیم که معیار *RMSE* آن در حدود *Epoch* ۲۰ به کمینه‌ی خود رسیده است و پس از آن با شیب کمی افزایش یافته است. در این بخش تلاش می‌کنیم تا جای ممکن عمل‌کرد مدل را روی داده‌های *Validation* بهبود بخشیم.



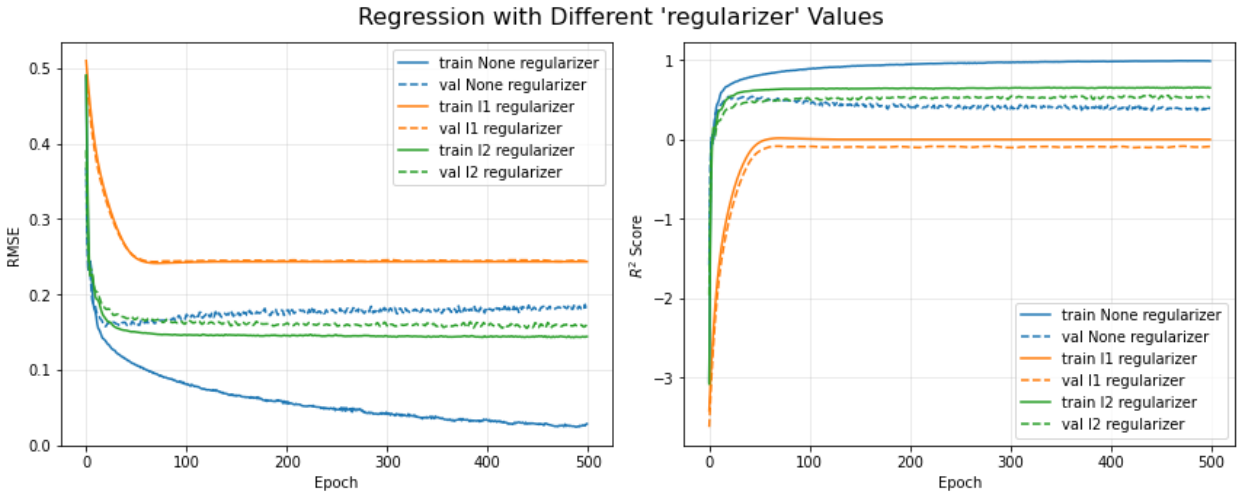
شکل ۲۴. عمل‌کرد مدل کنونی بر روی داده‌های آموزشی و *Validation*

شکل ۲۵ تاثیر اضافه کردن لایه‌ی *Dropout* پس از هر لایه‌ی مخفی، با نرخ ۰.۳ را نشان می‌دهد. مشاهده می‌کنیم که نه تنها عمل‌کرد مدل بر روی داده‌های آموزش بدتر شده است، بلکه بر روی داده‌های *Validation* نیز عمل‌کرد مدل ضعیف‌تر گردیده است. بنابراین به نظر می‌رسد افزودن لایه‌های *Dropout* کمکی به این مدل نمی‌کند.



شکل ۲۵. تاثیر اضافه کردن لایه‌ی *Dropout* با نرخ ۰.۳ پس از هر لایه‌ی مخفی

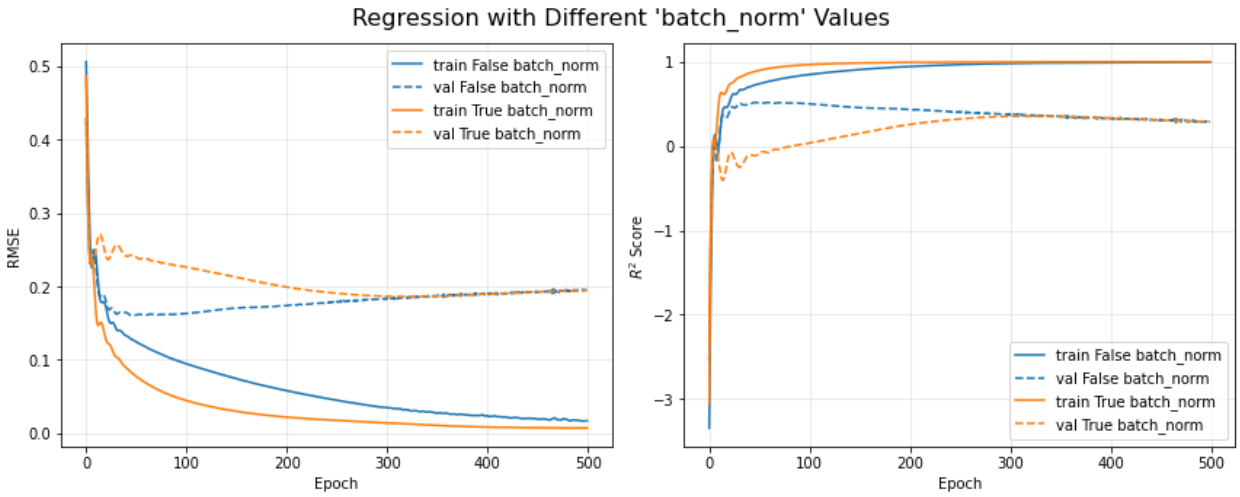
نمودارهای شکل ۲۶، عمل‌کرد مدل را در صورت استفاده از *Regularizer*های *L1* و *L2* نشان می‌دهد. مشاهده می‌کنیم که *L1* عمل‌کرد مدل را بر روی هر دو حالت آموزش و *Validation* ضعیف‌تر کرده است؛ اما *L2*، با این که باعث کاهش عمل‌کرد داده‌های آموزش شده است، اما عمل‌کرد *Validation* را بهبود بخشیده است؛ هدف اصلی ما نیز بهبود عمل‌کرد *Validation* است؛ بنابراین در مدل نهایی از این *Regularizer* استفاده خواهیم نمود.



شکل ۲۶. تاثیر افزودن *Regularizer*های مختلف به لایه‌های مخفی

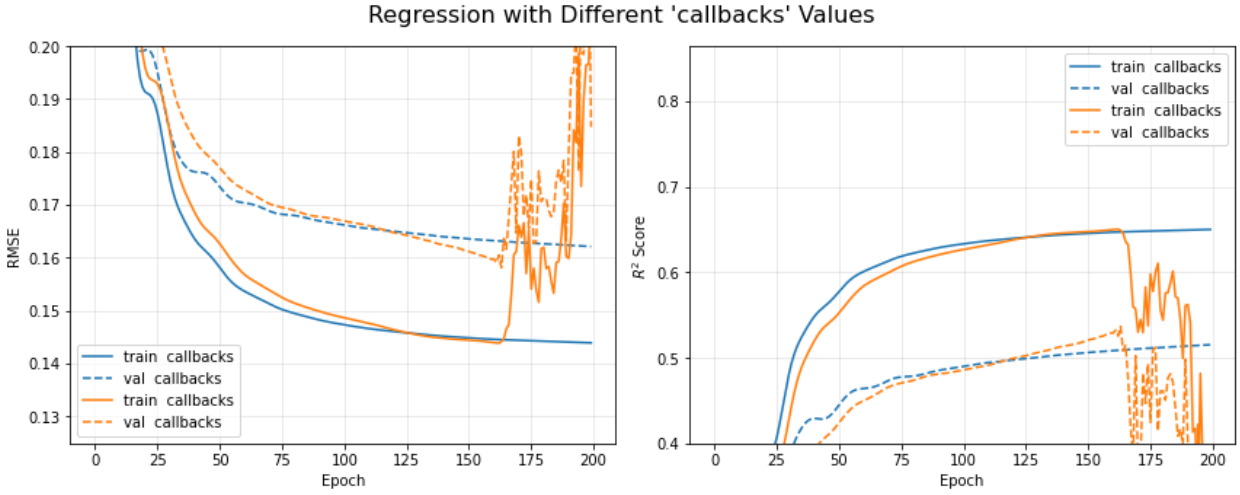
شکل ۲۷ عمل‌کرد مدل را در صورت استفاده و عدم استفاده از لایه‌ی *Batch Normalization* پس از ورودی شبکه نشان می‌دهد. مشاهده می‌کنیم که این لایه، با این که عمل‌کرد نهایی مدل را روی داده‌های *Validation*

تغییر نداده است، اما باعث بهبود عمل‌کرد مدل بر روی داده‌های آموزشی شده است؛ بنابراین استفاده‌ی همزمان از *L2 Regularizer* و لایه‌ی *Batch Normalization* می‌تواند به بهبود عمل‌کرد مدل بر روی داده‌های *Validation* کمک کند.



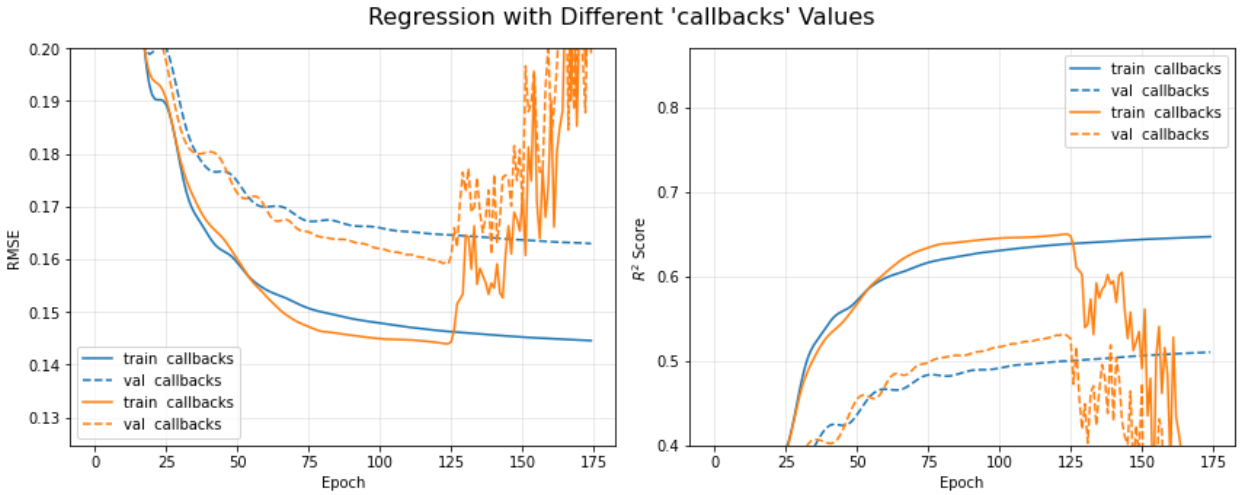
شکل ۲۷. تاثیر افزودن لایه‌ی *BatchNormalization* پس از لایه‌ی ورودی

در ادامه به بررسی تاثیر افزایش نرخ یادگیری بر عمل‌کرد مدل پرداخته‌ایم. به این منظور، پس از گذشت ۴۰ *Epoch*، مقدار نرخ یادگیری را در هر *Epoch* با نرخ ۱.۰۴ افزایش دادیم؛ شکل ۲۸ نشان می‌دهد که این افزایش تدریجی نرخ یادگیری، باعث بهبود عمل‌کرد مدل بر داده‌های *Validation* شده است؛ اما پس از افزایش بیش از حد نرخ یادگیری، عمل‌کرد مدل واگرا شده است.



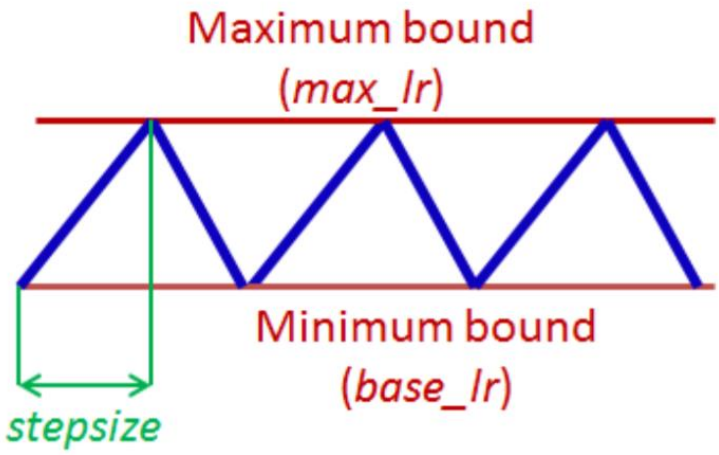
شکل ۲۸. تاثیر افزایش نمایی نرخ یادگیری پس از ۴۰ *Epoch* با نرخ ۱.۰۴

به منظور جلوگیری از این واگرایی، پس از رسیدن به *Epoch=80*، نرخ یادگیری را نصف کرده‌ایم و دوباره با همان نرخ ۱.۰۴ آن را افزایش دادیم. طبق شکل ۲۹ مشاهده می‌شود که عمل‌کرد مدل باز هم بهتر شده است.



شکل ۲۹. تاثیر افزایش نرخ یادگیری و سپس کاهش آن

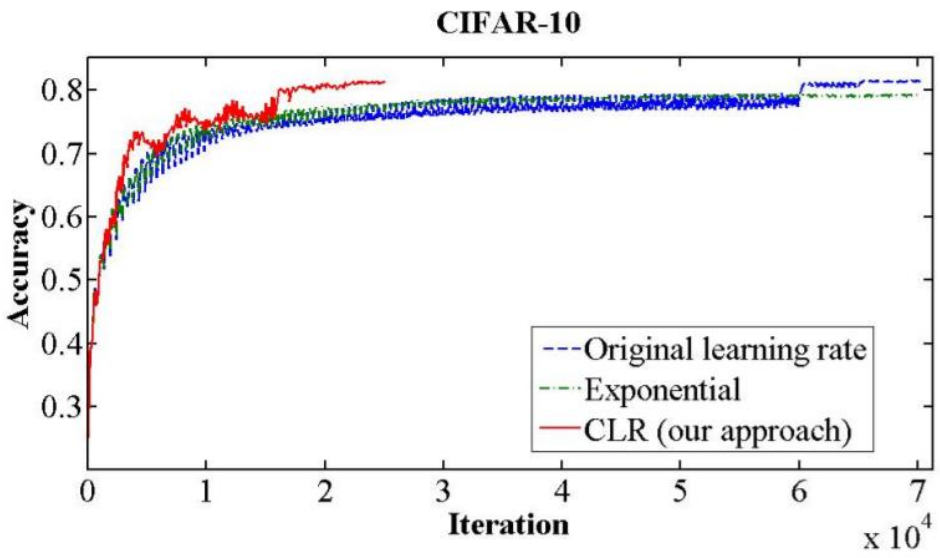
روشی مشابه با این روش در *[2]* معرفی شده است که با نام نرخ یادگیری متناوب[۱۷] از آن یاد می‌شود. در این روش، نرخ یادگیری بین دو مقدار همانند نمودار شکل ۳۰ نوسان می‌کند.



شکل ۳۰. نرخ یادگیری متناوب[2]

در مقاله‌ی یاد شده، این روش بر روی دیتاست *CIFAR-10* بررسی شده است و مشاهده شده است که سرعت همگرایی و عمل‌کرد نهایی مدل بسیار نسبت به روش‌های دیگر بهبود داشته است (شکل ۳۱).

[17] *Cyclic Learning Rate*



شکل ۳۱. عمل‌کرد روش نرخ یادگیری متناوب بر روی دیتاست *CIFAR-10*

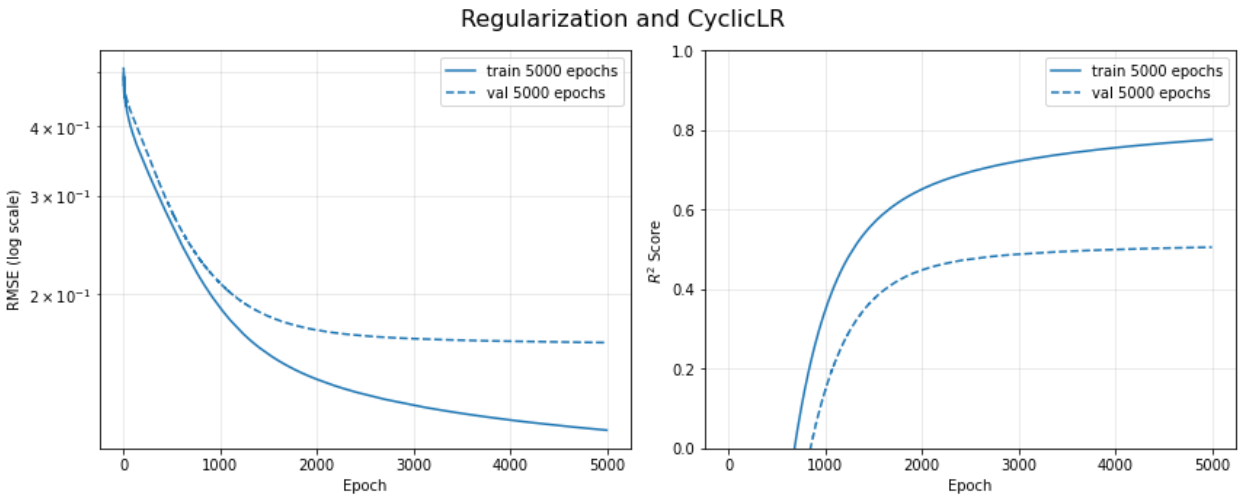
به منظور استفاده از این روش، از مخزن[۱۸] *CLR* در گیت‌هاب[۱۹] استفاده نموده‌ایم[3]؛ کلاسی[۲۰] که در آن تعریف شده است، دقیقا بر اساس مقاله‌ی ذکر شده عمل می‌کند.

پس از بررسی ابرپارامترهای مختلف برای *CLR*، مقادیر موجود در جدول ۱ را برای آن انتخاب نمودیم. طبق شکل ۳۲ می‌بینیم که عمل‌کرد *Validation* مدل پس از مدتی تقریبا ثابت می‌شود؛ اما عمل‌کرد مدل بر روی داده‌های آموزشی همچنان پس از ۵۰۰۰ *Epoch* نیز نزولی است. بنابراین می‌توانیم آموزش را تا هر میزان *Epoch* ادامه دهیم تا عمل‌کرد روی داده‌های آموزشی به بالاترین حد خود برسد و از بابت عدم واگرایی *Validation* اطمینان داشته باشیم.

جدول ۱. ابرپارامترهای انتخاب شده برای *CLR*

base_lr	*max_lr*	*step_size*	*mode*	*gamma*
1e-6	*3e-3*	*8*	*'exp_range'*	*0.999*

[18] *Repository*
[19] *GitHub*
[20] *Class*

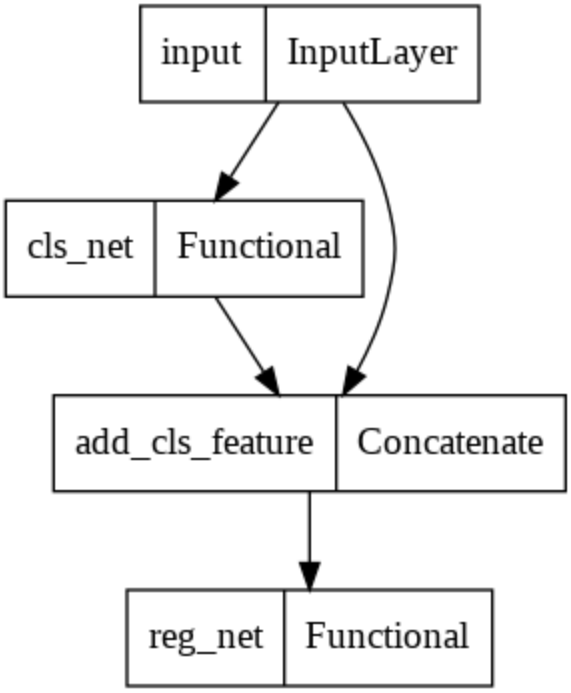


شکل ۳۲. عمل‌کرد مدل به همراه *Regularization* و نرخ یادگیری متناوب

بخش ۲-۳-۶. بحث در مورد مدل به دست آمده

در نهایت، با وجود بررسی ابرپارامترهای مختلف برای این دیتاست، مقدار R^2 *Score* مدل بر روی داده‌های *Validation* در حدود ۰.۵ باقی ماند. دلایل این موضوع می‌تواند کوچک بودن سایز دیتاست، وجود نویز در دیتاست، یا حتی مسیر اشتباه در انتخاب ابرپارامترها باشد.

روش دیگری که می‌توانستیم استفاده کنیم این بود که ابتدا مدلی را برای طبقه‌بندی متغیر *Outcome* طراحی کنیم، سپس خروجی آن را به همراه ویژگی‌های دیگر دیتاست، به یک مدل رگرسیون برای پیش‌بینی متغیر *Two Years Follow-up Cobb* بدهیم (شکل ۳۳). به دلیل کم‌بود زمان، عمل‌کرد این روش بررسی نشد.



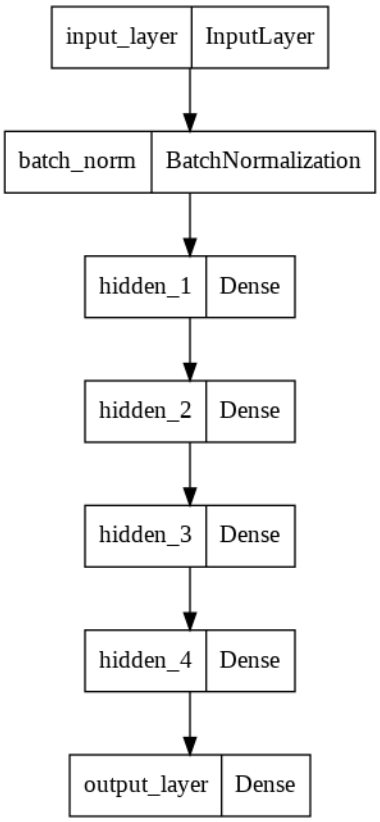
شکل ۳۳. مدل پیشنهادی دیگر برای حل این مسئله

بخش ۲-۳-۷. پیاده‌سازی مدل نهایی

پس از انتخاب ابرپارامترها در بخش قبل، یک مدل با ۴ لایه‌ی مخفی به شکل ۳۴ به دست آمد. می‌بینیم که تعداد پارامترهای قابل آموزش این مدل، بیش از ۷۵۰ هزار عدد است. بلوک دیاگرام مدل نیز در شکل ۳۵ نشان داده شده است.

```
Model: "reg_net"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 batch_norm (BatchNormalizat  (None, 8)                32
 ion)

 hidden_1 (Dense)            (None, 500)               4500

 hidden_2 (Dense)            (None, 500)               250500

 hidden_3 (Dense)            (None, 500)               250500

 hidden_4 (Dense)            (None, 500)               250500

 output_layer (Dense)        (None, 1)                 501

=================================================================
Total params: 756,533
Trainable params: 756,517
Non-trainable params: 16
_________________________________________________________________
```

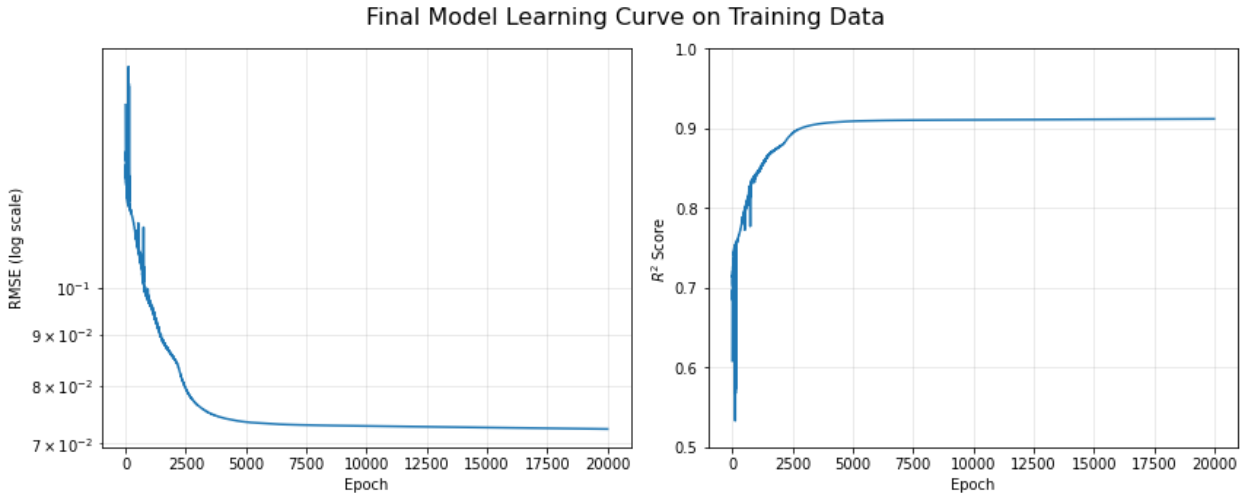
شکل ۳۴. خلاصه‌ای از مدل نهایی پیاده‌سازی شده



شکل ۳۵. بلوک دیاگرام مدل نهایی پیاده‌سازی شده

با توجه به این که می‌دانیم عمل‌کرد *Validation* این مدل هیچ‌گاه واگرا نمی‌شود، به جهت بالا بردن عمل‌کرد مدل بر روی داده‌های آموزش، تا ۲۰ هزار *Epoch* آموزش را ادامه دادیم. با این کار، اگر مدل در زمان تست با داده‌ای مشابه با داده‌های آموزش مواجه شود، می‌تواند عمل‌کرد بسیار خوبی داشته باشد و در عین حال عمل‌کرد آن روی داده‌های جدید نیز کاهش نیافته باشد.

نمودار یادگیری نهایی، در شکل ۳۶ نشان داده شده است. در نهایت بر روی داده‌های آموزش، مقدار *RMSE* به ۰.۰۷۲۴ و مقدار R^2 *Score* به ۰.۹۱۱۷ رسید.



شکل ۳۶. نمودار یادگیری نهایی

بخش ۲-۴. تست عمل‌کرد مدل

بخش ۲-۴-۱. روش تست مدل

برای تست مدل، کافیست ماژول *predict.py* را *import* نموده و تابع *test* را با آرگومان ورودی «مسیر فایل اکسل تست» اجرا نمایید (شکل ۳۷). این تابع دو خروجی دارد که خروجی اول پیش‌بینی ستون *Two Years Follow-up Cobb* و خروجی دوم پیش‌بینی ستون *Outcome* است. توجه کنید که ستون‌های فایل تست دقیقا باید مطابق با ستون‌های فایل آموزش باشد و فرمت فایل *xlsx* باشد. همچنین پوشه‌ی *reg_model* نیز باید در محل اجرای کد وجود داشته باشد.

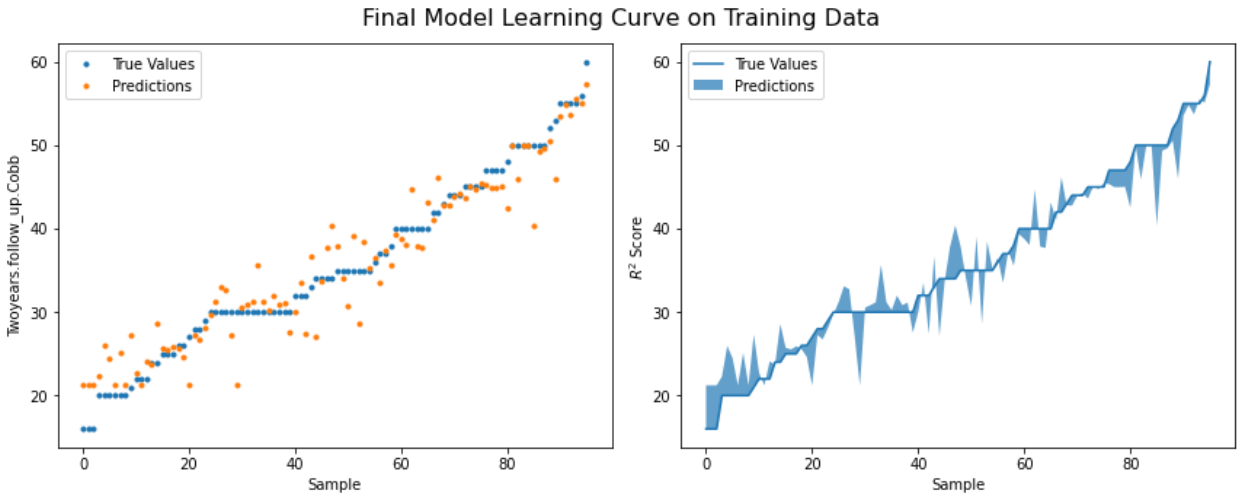
```
from predict import test
y_reg_hat, y_cls_hat = test('/content/traindataset_for_intro_to_ml.xlsx')
```

شکل ۳۷. نحوه‌ی تست مدل

توجه شود که برای اجرای ماژول *predict.py*، لازم است کتاب‌خانه‌های موجود در *requirements.txt* از قبل نصب شده باشند.

بخش ۲-۴-۲. عمل‌کرد مدل بر روی متغیر هدف رگرسیون

همان‌طور که گفته شد، مقدار R^2 *Score* مدل بر روی داده‌های آموزش به ۰.۹۱۱۷ رسید. نمودارهای شکل ۳۸ مقایسه‌ای از مقادیر حقیقی و مقادیر پیش‌بینی شده توسط مدل را نشان می‌دهند.



شکل ۳۸. عمل‌کرد مدل بر روی متغیر هدف رگرسیون

بخش ۲-۴-۳. عمل‌کرد مدل بر روی متغیر هدف طبقه‌بندی

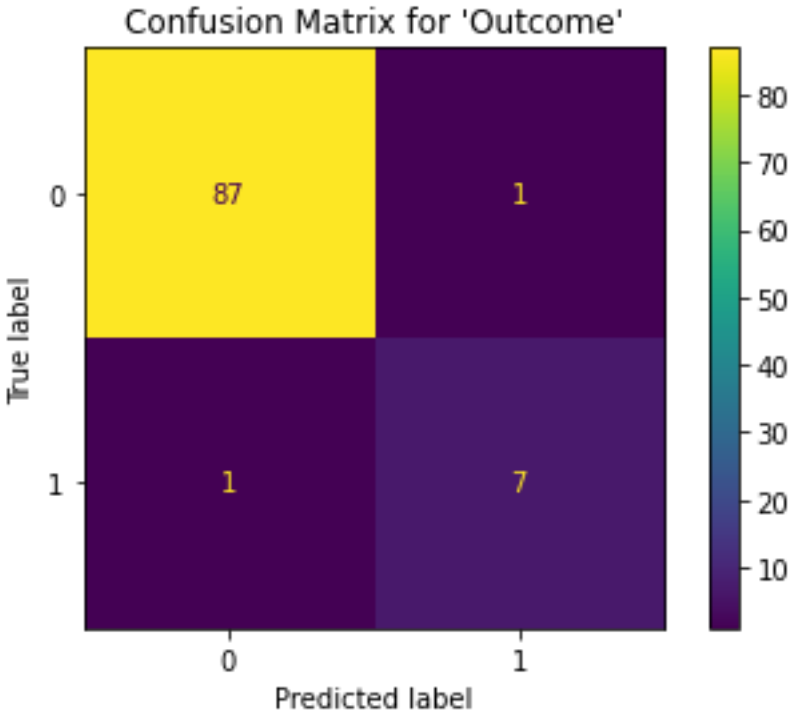
همان‌طور که قبلا اشاره شد، در این پروژه، برای طبقه‌بندی متغیر *Outcome*، از نتیجه‌ی پیش‌بینی رگرسیون استفاده می‌کنیم. پس از بررسی نتایج پیش‌بینی رگرسیون، بهترین مرز برای رسیدن به بیش‌ترین *Precision* و *Recall* روی کلاس‌ها، ۵۰ بود؛ یعنی هنگامی که مقدار *Two Years Follow-up Cobb* بیش از ۵۰ پیش‌بینی شود، *Outcome=1* و در غیر این صورت، *Outcome=0* خواهد بود.

در شکل ۳۹ مقادیر *Precision*، *Recall*، *F1-Score* برای هر کلاس و میانگین آن‌ها و همچنین صحت طبقه‌بندی نشان داده شده است. شکل ۴۰ نیز ماتریس درهم‌ریختگی[۲۱] این مدل را بر روی داده‌های آموزش نشان می‌دهد. مشاهده می‌کنیم که تنها یک پیش‌بینی اشتباه برای هر کلاس رخ داده است.

[21] *Confusion Matrix*

	precision	recall	f1-score
0	0.99	0.99	0.99
1	0.88	0.88	0.88
accuracy			0.98
macro avg	0.93	0.93	0.93
weighted avg	0.98	0.98	0.98

شکل ۳۹. عمل‌کرد مدل بر روی متغیر هدف طبقه‌بندی



شکل ۴۰. ماتریس درهم‌ریختگی مدل برای پیش‌بینی متغیر *Outcome*

بخش ۳. مراجع

[1] https://pythonmachinelearning.pro/perceptrons-the-first-neural-networks/

[2] arXiv:1506.01186 [cs.CV]

[3] https://github.com/bckenstler/CLR