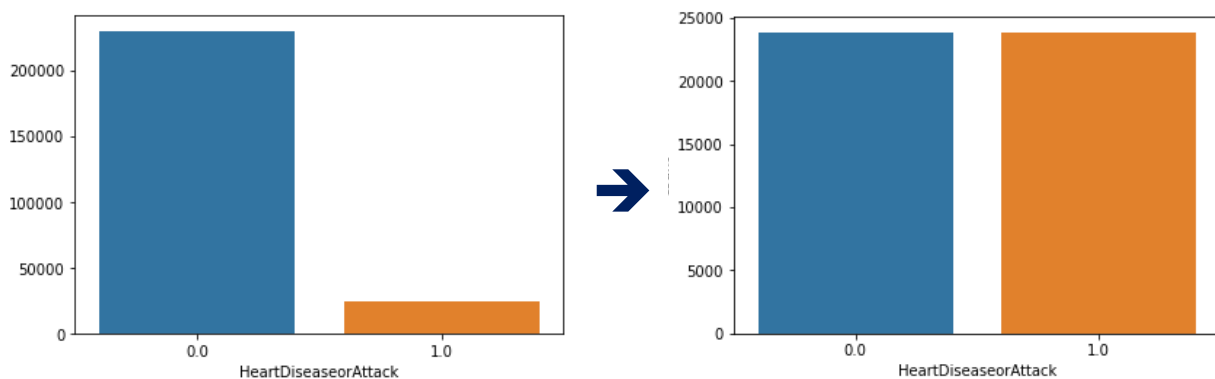


داده ها و پیش پردازش

مجمومه دادگان فایل `heart_disease_health_indicators_BRFSS2015.csv` حاصل جواب های یک پرسشنامه ست که در آن از افراد درباره ی فاکتورهایی سوال می شود که از نظر پزشکی عامل بالابرنده ی احتمال ابتلا به بیماری های قلبی شمرده می شود. در نهایت از افراد پرسیده شده که آیا مبتلا به بیماری قلبی هستند یا نه. این مجموعه شامل ۲۵۳۶۸۰ ردیف اطلاعات است که همه فرمت عددی دارند اما به لحاظ ماهیت اکثر ستون های آن متغیرهای `categorical` هستند که پاسخ به سوالات بله/خیر هستند. مثلا دیابت، فشارخون، کلسترول، مصرف میوه... و از طرف دیگر تعداد دیگری متغیر هم هستند که همین حالت را دارند ولی چندکلاسه هستند، مثل تحصیلات، درآمد یا شاخص BMI و حتی سن هم به صورت تعلق به گروه های سنی گزارش شده و نه عدد واقعی سن. پس می توان ادعا کرد تمام ۲۲ ستون اطلاعات `categorical` هستند. دانستن این مطلب از این جهت مهم است که در ادامه ی کار برخلاف روال معمول شبکه های عصبی به پیش پردازش های `scaling` نیاز پیدا نمی شود. ستون اول `HeartDiseaseorAttack` را به عنوان متغیر هدف تعیین می شود. با یک نمودار فراوانی می توان فهمید که داده ها روی متغیر هدف به شدت نامتقارن است. در توضیحات هم آمده که مقدار هدف ۰ یعنی افراد سالم در این داده ها ۲۲۹۷۸۷ ردیف هست در حالی که به ازای هدف ۱ که افراد بیمار هستند ۲۳۸۹۳ داده موجود هست. این نسبت ۱ به ۱۰ کار یادگیری را دشوار می کند. اگر مدلی داشته باشیم که تمام افراد را سالم تشخیص دهد مدل با اینکه عملا هیچ کاری نکرده اما نتیجه ی `accuracy` و `Loss` خوبی خواهد داشت! برای حل این مشکل از نمونه گیری تصادفی استفاده می کنم تا داده ها متقارن شوند و برای این کار از `imblearn.under_sampling` استفاده کردم که تعداد افراد سالم را کاهش داد و هم مجموعه داده ها را کوچک کرد و هم اینکه مشکل عدم تقارن را از بین برد و حالا با یک مجموعه حدودا ۵۰۰۰۰ تایی از داده ها کار می کنیم.



چرا دور ریختن این داده ها مشکل زیادی ندارد؟ هدف اصلی این مدل سازی تشخیص الگوی بیماری است و همه ی تمرکز روی این است که این الگوها از الگوهای سالم جدا شوند. حالا همانطور که قبلا گفته شد روی ۲۱ ستون متغیر وابسته همه ی مقادیر `categorical` هستند و تعداد مشخصی هم می توانند مقدار بگیرند. با یک حساب تخمینی می توان دید که اندازه ی این مجموعه داده نسبت به کاری که قرار است انجام شود بزرگ هست و قطعاً تعداد زیادی مقادیر تکراری دارد. به علاوه ی اینکه روی این متغیرها تنوع الگوی سالم از الگوی بیماری کمتر است چون همانطور که در توضیحات داده ها هم آمده تمام این ها فاکتورهای افزایش خطر هستند. مجموعه داده ی حاصل از مرحله ی پیش پردازش را در دیتافریم جداگانه ی `X` و `y` قرار می دهیم برای ادامه ی کار.

روش کار

گفته شد که هدف اصلی شناخت و تشخیص الگوی بیماری در افراد است. بیماری قلبی یک بیماری کشنده ست و طبیعتاً سنگین ترین هزینه ی اشتباه تشخیص مدل جان فرد است و دومین هزینه مربوط به اقدامات پزشکی است که صرف افرادی می شود که سالم هستند و بعد از معاینات و آزمایش های بیشتر مشخص می شود که سالم بوده اند. هدف اول را اینطوری تعین می کنم که مدل به ازای مجموعه ی داده های بیمار چند درصد از آن ها را می تواند شناسایی کند و در حالتی که تمام نمونه ها را بیمار تشخیص دهد این هدف بیشینه تامین شده اما کارایی ندارد چون هزینه ی دوم سنگین می شود و یعنی تمام افراد سالم ناچار باید روال معاینات و غیره را بگذرانند. برای کاهش هزینه ی دومی هدف اینطور تعین می شود که از بین تمام افرادی که مدل به عنوان بیمار شناسایی کرده چند درصدشان واقعاً بیمار هستند. پس برای ارزیابی مدل این دو فاکتور را در نظر می گیرم که اولی را میتوان به **recall** و دومی را به **precision** تعبیر کرد و واضح است که در بیشینه کردن هر دو باهم یک نقطه تعادل لازم است و هر دو باهم نمی توانند ماکسیمم بشوند.

شیوه ی کار این است که در چند مرحله چند دسته بند مختلف می سازم و هر بار سعی می کنم مدل هایی که در مرحله قبلی عملکرد خوبی داشته اند را با تغییراتی در معماری شبکه عصبی یا افزایش عمق یا یکی دو روش دیگر بازآمایی کنم تا ببینم نتیجه ها قابلیت بهبود دارند یا نه. چون مدل های شبکه عصبی وابستگی زیادی به مقداردهی اولیه وزن ها دارند هر بار ۳ نتیجه از روش **3fold cross validation** گرفتم که اگر با تکیه بر یک نتیجه ی خوب از یک مرحله جلو رفتم تا حدی مطمئن باشم که نتیجه تصادفی نبوده. چون در نهایت شاید اختلاف مدل هایی که قرار است به عنوان کاندید مدل برتر نهایی انتخاب شوند کم خواهد بود. در آخرین قدم هر مدلی که بهترین نتایج را داشته باشد در یک فرایند **10 fold cross validation** قرار می گیرد و بهترین نتیجه ی آن به عنوان مدل نهایی و میانگین آن هم گزارش می شود.

برای آنکه درک سریع تری از معماری هر شبکه به دست بدهم از دو تابع کمکی استفاده کردم تا هر بار یک شکل نظیر شبکه را رسم کند. این شکل هر بار با **visualize_model_nn** رسم شده که در دل خود از **utils_model_nn_config** استفاده می کند. در شکل ها بایاس رسم نشده.

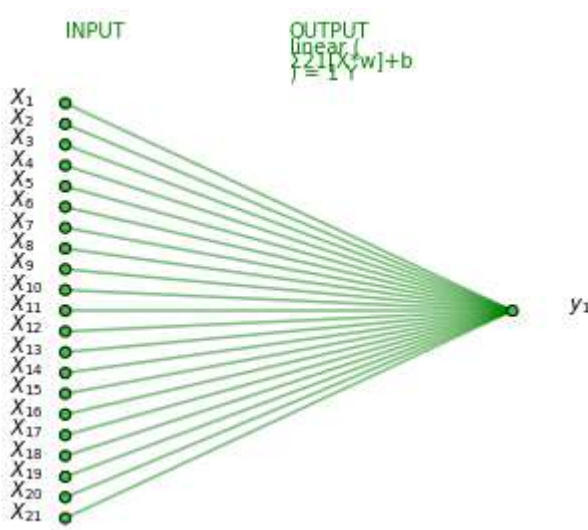
برای ساخت مدل ها از **tensorflow.keras** و برای دسته بندی تست و آموزش در **kfold cv** هم از **KFold** **sklearn.model_selection** استفاده شده.

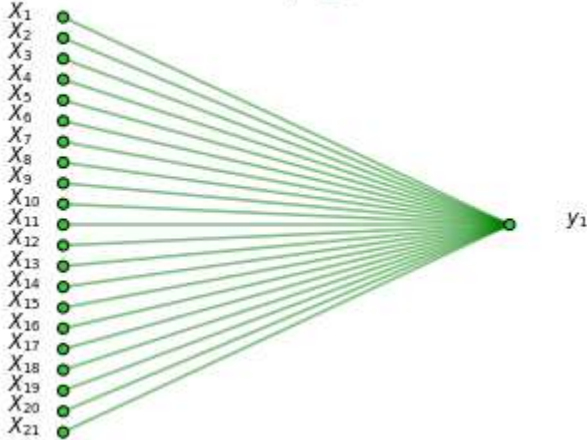
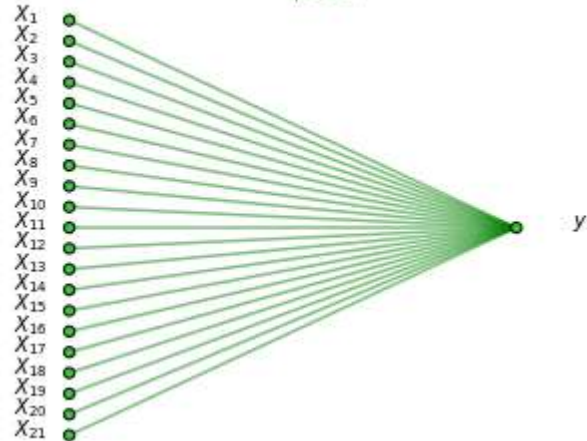
در تمام مدل ها برای بهینه سازی وزن ها از روش **adam** استفاده کردم. انتخاب بعدی روش گرادیان کاهشی بود که به خاطر گسستگی داده ها از آن صرف نظر کردم. برای تابع **loss** از **binary_crossentropy** استفاده کردم که انتخاب مناسب و پیشنهادشده ای برای دسته بندی های دوحالته با مدلهای غیراحتمالی هست. تمام مدل ها با **epoch = 100** و **batch_size = 64** آموزش داده شدند.

در هر بار اجرا ۳۰ دصد داده ها برای ارزیابی نگه داشته می شود و نتیجه ی هر بار آموزش مدل ۲ نمودار است که خط مشکی برابر **loss** و دوتای دیگر **recall** و **precision** روی دو دسته داده ی آموزش و ارزیابی روی ۱۰۰ تا ایپاک انجام شده هستند. علاوه بر این مدل نهایی حاصل هر یک از ۳ آموزش با سه عدد به ترتیب **loss, precision, recall** برای مجموعه آموزش در سطر اول و برای مجموعه ی تست (روی هر **fold**) در سطر دوم گزارش می شود. نمودار ها با استفاده از تابع **plot_train_val** رسم شده اند.

شناخت بیشتر داده ها

در قدم اول ۳ دسته بند پرسپترون می‌سازم که بیشتر داده ها را بشناسم. همه ی آن ها ۲۱ ورودی همراه یک بایاس میگیرند و هر کدام از یکی از توابع سیگموئید، خطی و **relu** به عنوان فعال ساز استفاده می‌کنند. واضح است که داده ها جداپذیر خطی نیستند هدف اصلی این کار مقایسه ی رفتار یک پرسپترون با تابع فعالساز متفاوت است که بدانم کدام یک بهتر می‌تواند این عدم خطی بودن را بهم بریزد تا برای اولین مدلی که می‌سازم به عنوان تابع فعالساز لایه اول از همان واحد استفاده کنم.

		Loss	Precision	Recall
model_p1				
	میانگین آموزش	4.52	0.77	0.46
	میانگین تست	4.53	0.77	0.46
	نتایج روی آموزش	2.25	0.72	0.85
		4.72	0.79	0.37
		6.60	0.81	0.16
	نتایج روی تست	2.22	0.71	0.84
		4.68	0.78	0.37
		6.70	0.82	0.17
پرسپترون با تابع فعالساز خطی	نمودار آموزش و ارزیابی پرش های زیادی دارد. فقط در مورد اول همگرایی دیده می‌شود. در دو حالت دیگر نتایج غیرقابل اعتماد است. تفاوت عملکرد در هر آموزش نسبت به دیگری زیاد است.			
model_p2				
	میانگین آموزش	0.49	0.76	0.78
	میانگین تست	0.49	0.76	0.78
	نتایج روی آموزش	0.49	0.75	0.80
		0.48	0.76	0.79
		0.48	0.78	0.74

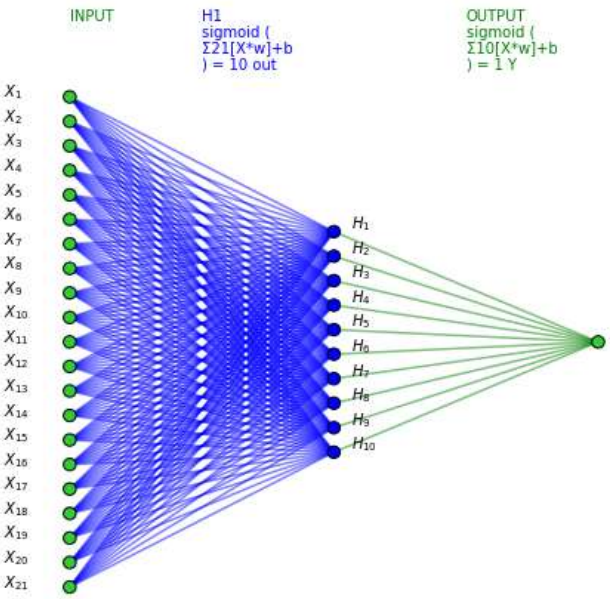
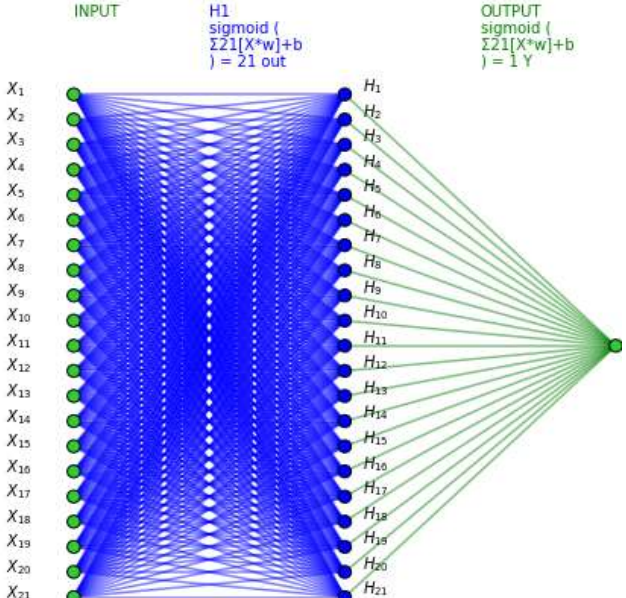
<div><div>INPUT</div><div>$y_i = \text{sigmoid}(\sum_{j=1}^n x_j w_j + b)$</div></div>	نتایج روی تست	0.48 0.49 0.49	0.76 0.76 0.77	0.80 0.79 0.74
پرسپترون با تابع فعالساز sigmoid	در هر ۳ مدل همگرایی وجود دارد. تفاوت معیارهای ۳ مدل نسبت به حالت اول کمتر شده. نتایج قابل اعتماد تری به دست می‌دهد.			
model_p3				
<div><div>INPUT</div><div>$y_i = \text{relu}(\sum_{j=1}^n x_j w_j + b)$</div></div>	میانگین آموزش	7.71	0	0
	میانگین تست	7.71	0	0
	نتایج روی آموزش	7.68	0	0
		7.74	0	0
		7.72	0	0
	نتایج روی تست	7.78	0	0
		7.66	0	0
		7.70	0	0
پرسپترون با تابع فعالساز relu	در هر ۳ اجرا نتیجه ی به دست آمده فاقد اعتبار است. فرایند آموزش به درستی صورت نگرفته.			

در همه ی این ها نتایج تست و آموزش به هم نزدیک اند ولی چون این مدل ها فاقد پیچیدگی لازم اند این بررسی های مربوط به برازش و خطا اینجا انجام نمی دهیم. هدف اصلی این بخش این که برای قسمت بعدی که اولین شبکه ی عصبی دسته بندی را تعریف می کنیم برای تابع فعالسازی لایه اول از سیگموئید استفاده می کنیم.

شبکه عصبی با یک لایه پنهان

در این بخش دو مدل model_nn1 و model_nn2 ساخته شده. اولین سوال این بخش این است که وجود یک لایه پنهان نسبت به یک پرسپترون چقدر عملکرد را بهتر می‌کند؟ و دوم این که این لایه چند گره داشته باشد؟ معماری این بخش دو لایه نود (یکی خروجی و یکی میانی یا پنهان) و یک لایه ورودی دارد. تابع فعالساز لایه میانی سیگموئید است. (نتیجه مشاهدات بخش قبلی) لایه ورودی ۲۱ متغیر وابسته را می‌گیرد و لایه خروجی شامل یک نود خروجی است که با تابع سیگموئید نتایج نهایی را تولید می‌کند و کار تخصیص کلاس صفر یا یک (سالم یا بیمار) را انجام می‌دهد. در model_nn0 لایه میانی ۵ نود دارد. در نمودار آموزش مدل‌ها همگرایی هست و ۳ نتیجه هم به یکدیگر نزدیک اند و هم اینکه برای شروع نتایج خوبی هستند. با این مشاهده model_nn1 و model_nn2 را با ۱۰ و ۲۱ نود در لایه میانی تعریف کردم تا ببینم آیا بیشتر کردن این عدد نتایج را بهتر هم می‌کند یا خیر ولی نتیجه در هر دو حالت بسیار نزدیک به هم بودند و فقط اندکی از model_nn0 روی مقدار recall بهتر. هر دو مدل این بخش در قسمت‌های بعدی استفاده خواهند شد تا شاید در تغییراتی که بعداً انجام می‌شوند یکی بر دیگری برتری داشته باشد ولی اگر همین‌جا متوقف شویم model_nn1 چون سادگی بیشتر دارد (وزن‌های کمتر ۲۳۱ در مقابل ۴۸۴) ارجح است.

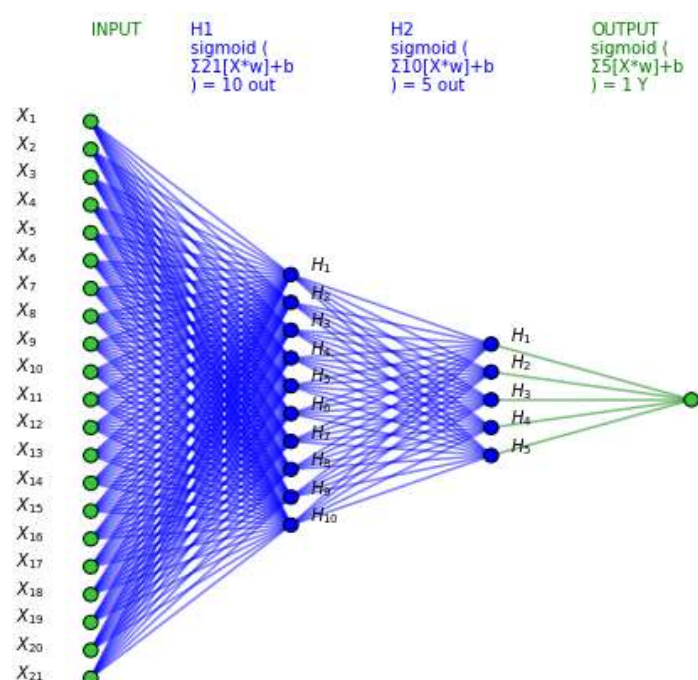
		Loss	Precision	Recall	
model_nn0					
<div><div>INPUT</div><div>H1 sigmoid ($\sum_{21}[X*w]+b$) = 5 out</div><div>OUTPUT sigmoid ($\sum_5[X*w]+b$) = 1 Y</div></div>	میانگین آموزش	0.47	0.76	0.80	
	میانگین تست	0.48	0.75	0.80	
	نتایج روی آموزش	0.48	0.75	0.82	
		0.48	0.77	0.77	
		0.47	0.75	0.83	
	نتایج روی تست	0.48	0.74	0.83	
		0.48	0.77	0.76	
		0.48	0.75	0.82	
	model_nn1				
		میانگین آموزش	0.48	0.74	0.83
میانگین تست		0.48	0.74	0.83	
نتایج روی آموزش		0.48	0.75	0.81	
		0.48	0.74	0.84	
		0.48	0.73	0.85	

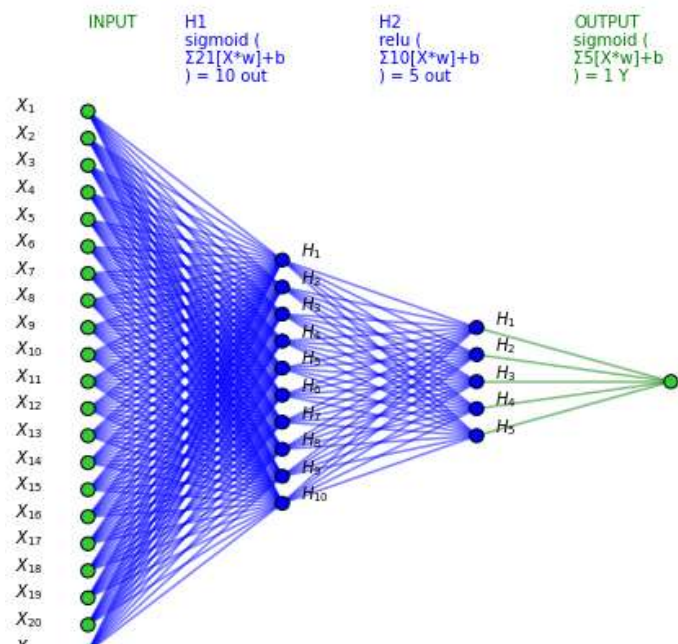
	نتایج روی تست	0.48 0.48 0.48	0.75 0.74 0.72	0.82 0.83 0.85
mode_nn2				
	میانگین آموزش	0.47	0.75	0.83
	میانگین تست	0.48	0.74	0.82
	نتایج روی آموزش	0.48	0.75	0.80
		0.47	0.74	0.83
		0.47	0.74	0.84
	نتایج روی تست	0.48	0.75	0.80
		0.48	0.75	0.82
		0.48	0.73	0.84

شبکه عصبی با دو لایه پنهان

در این بخش شبکه را یک لایه عمیق تر می‌کنم. در لایه میانی دوم ۵ نود قرار می‌دهم و با دو تابع سیگموئید و relu روی این لایه شبکه را آموزش می‌دهم. مدل‌های این بخش model_dn1 و model_dn2 هستند. هذ دوی این‌ها نتیجه ی عمیق کردن model_nn1 از بخش قبل هستند. نتیجه‌ها روی اولی بهتر است. همانطور که روی نتیجه‌ها پیداست model_dn1 نتیجه بهتری روی recall دارد و البته مقدار precision آن کمتر است ولی اولاً این تفاوت کمتر از مقدار بهبود recall است و دوماً هدف اول که در بخش‌های پیش تعیین شد با recall هست. در این بخش یک سوال دیگر هم ممکن است پیش بیاید. اگر در

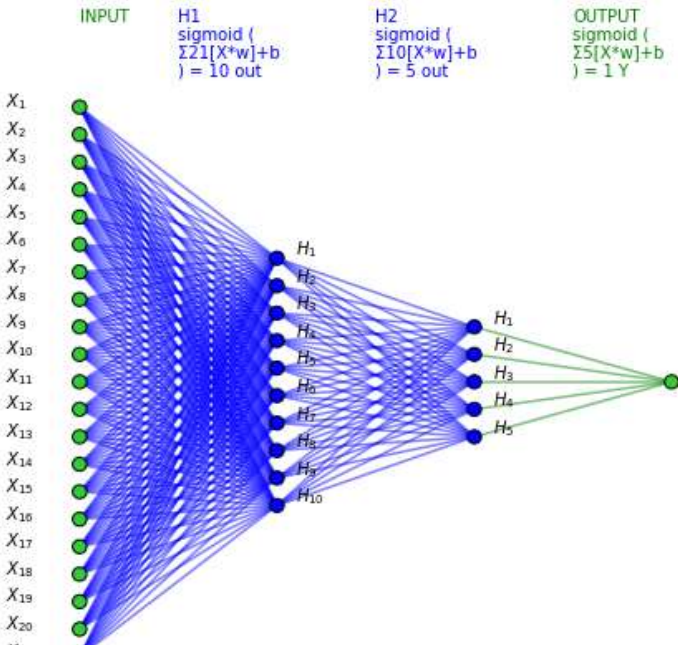
هر بار به جای اینکه وزن ها را مطابق معمول شبکه عصبی تصادفی بدهم، از وزن های نهایی لایه ی model_nn1 استفاده کنم آیا نتیجه ای که به دست می آید در نهایت بهتر خواهد بود یا نه؟ این فرض را روی model_dn3 امتحان کردم و تغییر معناداری دیده نشد. در تمام موارد روند آموزش روی نمودار همگرایی تابع loss را نشان می دهد. اگر خط سیاه را در نمودار تست و ارزیابی قیاس کنیم در هیچ کدام نشان بیش برآزش دیده نمی شود. و علاوه بر این انطباق بسیار زیادی روی نتایج تست و آموزش هست. از اینجا به بعد بهترین نتیجه ای که ساختیم مربوط به model_dn1 هست و تلاش این است که این نتیجه را هم می توان بهبود داد یا نه؟

		Loss	Precision	Recall	
model_dn1					
	میانگین آموزش	0.48	0.74	0.85	
	میانگین تست	0.48	0.73	0.84	
	نتایج روی آموزش	0.48	0.73	0.86	
		0.48	0.74	0.84	
		0.48	0.74	0.83	
	نتایج روی تست	0.48	0.73	0.86	
		0.48	0.73	0.84	
		0.48	0.73	0.83	
	افزودن یک لایه پنهان دوم به model_nn1 با تابع فعالساز sigmoid				
	model_dn2				
	میانگین آموزش	0.48	0.76	0.80	
	میانگین تست	0.48	0.75	0.80	
	نتایج روی آموزش	0.48	0.76	0.80	
		0.48	0.76	0.79	
		0.48	0.75	0.82	

	نتایج روی تست	0.48 0.49 0.48	0.76 0.76 0.74	0.79 0.78 0.81
---	---------------	----------------------	----------------------	----------------------

افزودن یک لایه پنهان دوم به model_nn1 با تابع فعالساز relu

mode_dn3

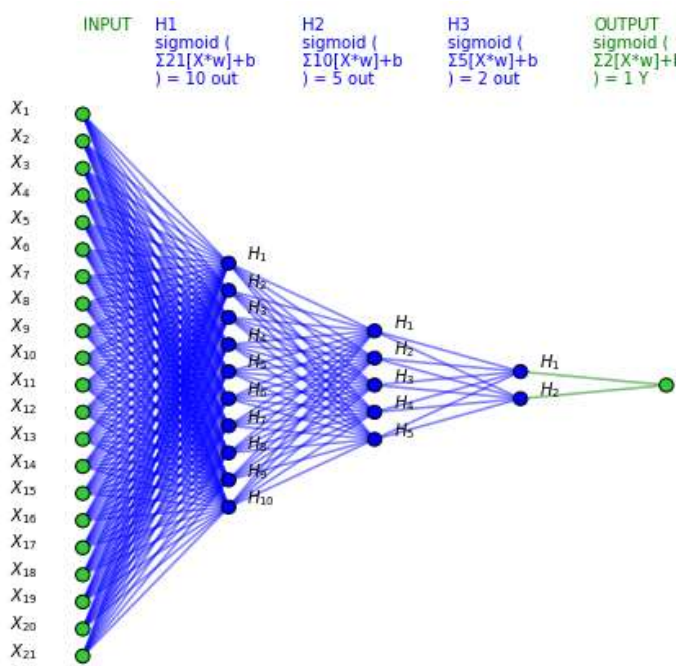
	میانگین آموزش	0.47	0.75	0.82
	میانگین تست	0.48	0.75	0.81
	نتایج روی آموزش	0.48	0.75	0.82
		0.47	0.76	0.81
		0.48	0.74	0.83
	نتایج روی تست	0.48	0.74	0.82
		0.49	0.75	0.80
		0.47	0.75	0.82

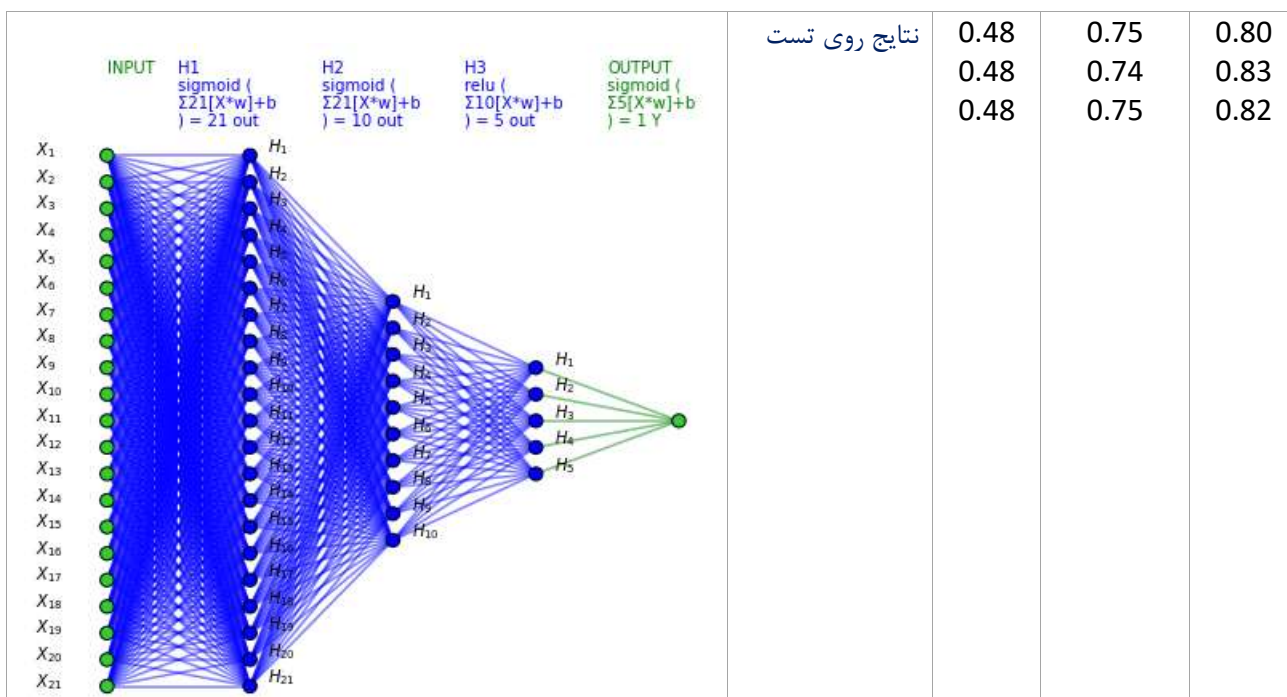
معماری model_dn1 با وزن های لایه پنهان model_nn1 به

عنوان مقدار اولیه وزن های لایه اول

شبکه عصبی با سه لایه پنهان

آیا بازهم عمیق تر کردن شبکه نتیجه را بهتر می‌کند؟ در این بخش اول شبکه ی `model_dnn1` که بهترین نتیجه را تا اینجا داشته با اضافه کردن یک لایه با دو نود یک لایه عمیق تر می‌کنم. این لایه از تابع سیگموئید استفاده می‌کند تا شبکه ی `model_dnn1` تعریف شود. دومین معماری که با ۳ لایه استفاده می‌شود `model_dnn2` حاصل دو لایه اضافه و عمیق کردن شبکه ی `model_dnn1` است. این شبکه عمیق همان لایه پنهان `model_nn1` شامل ۲۱ نود با تابع سیگموئید را دارد و علاوه بر آن در دو لایه پنهان بعدی به ترتیب ۱۰ و ۵ نود با توابع سیگموئید و `relu` دارد. با معماری ۳ لایه پنهان نتیجه ای که تا اینجا بود بهبود نیافت. در `model_dnn2` نمودار تابع ارزیابی با جلو رفتن نشانه های جزئی از واگرایی نشان می‌دهد ولی تاثیرگذار نیست. پارامترهای این مدل زیاد است. در بخش بعد با اعمال تغییراتی روی این مدل باز هم برای بهبود بهترین نتیجه تلاش می‌شود. ولی این بخش پایان کار عمیق کردن شبکه ست و بیش از ۲ لایه میانی برای این مساله مناسب نیست.

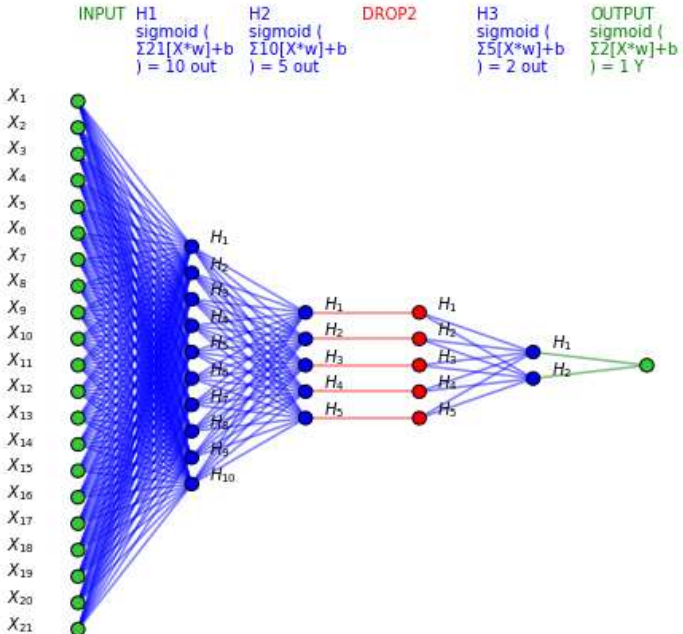
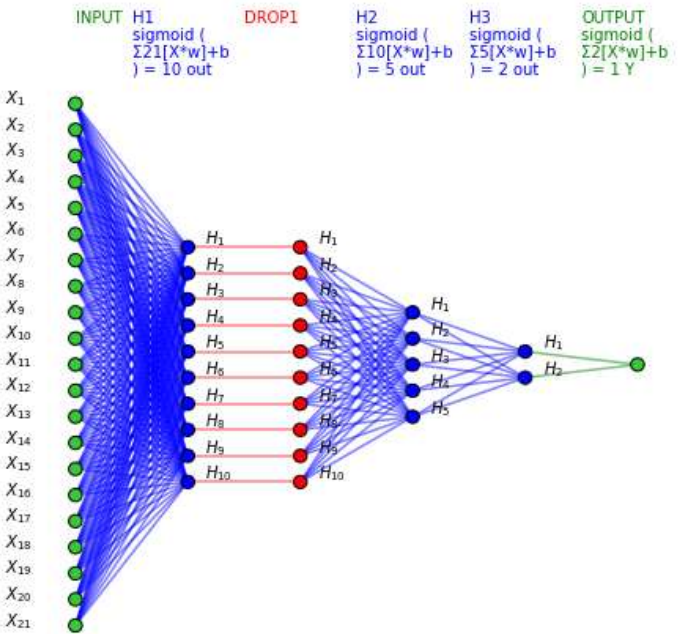
		Loss	Precision	Recall	
model_ddn1					
	میانگین آموزش	0.48	0.74	0.83	
	میانگین تست	0.48	0.74	0.83	
	نتایج روی آموزش	0.47	0.75	0.82	
		0.47	0.74	0.84	
		0.48	0.74	0.84	
	نتایج روی تست	0.48	0.74	0.82	
		0.49	0.74	0.83	
		0.48	0.74	0.83	
	model_ddn2				
		میانگین آموزش	0.47	0.75	0.82
میانگین تست		0.48	0.75	0.82	
نتایج روی آموزش		0.47	0.75	0.81	
		0.47	0.74	0.83	
		0.47	0.75	0.82	



چند آزمایش روی بهترین نتایج بدست آمده

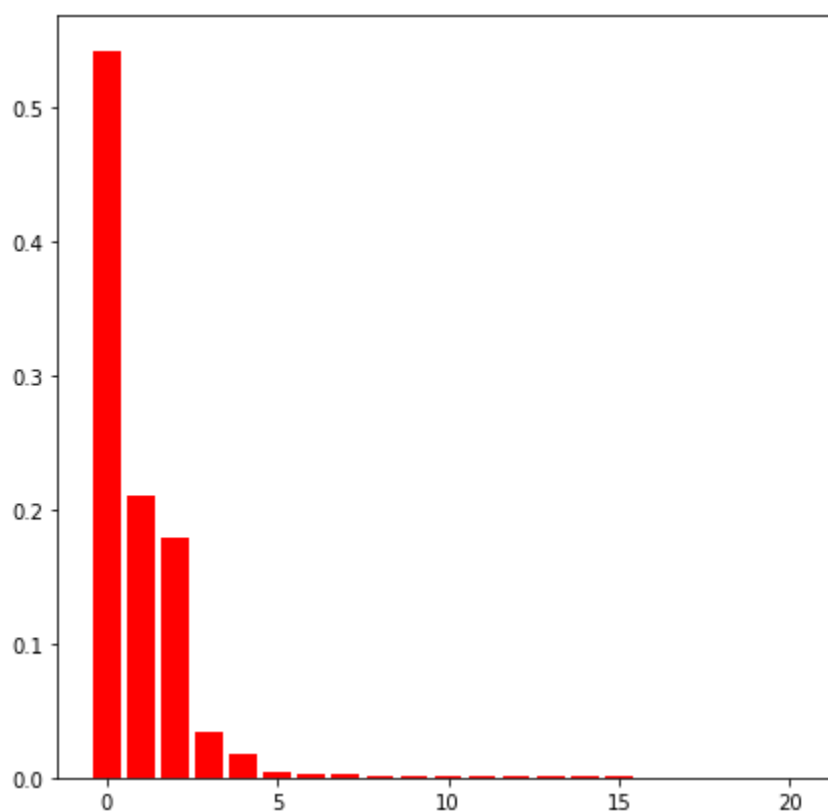
عمیق ترین مدلی که ساخته شد و نتیجه ای در خوب هم داشت شبکه ی `model_ddn1` بود که ۳ لایه پنهان داشت ولی دیدیم که نسبت به شبکه ی ۲ لایه پنهان نظیر خودش کاراتر نشده. صرفا برای آزمایش اینکه این معماری ۳ لایه میانی بهبودی در نتیجه می دهد یا نه باز همان را می سازم ولی این بار کمی آزادی مدل را کمتر می کنم. با اضافه کردن یک لایه `dropout` ابتدا لایه دوم و بعد در لایه اول و بعد در لایه دوم در این معماری سعی میکنم با هر بار روند آموزش به اندازه ی یک دهم از وزن های ورودی به لایه بعدی را به طور تصادفی صفر کنم این کار معمولا یک روش برای مقابله با بیش برازش در مدل هایی است که پیچیدگی زیادی دارند ولی اینجا قصد من این است که با کم کردن وزن های آموزشی معماری با ۳ لایه پنهان بهتر از ۲ لایه کار می کند یا همچنان بهبودی در کار نخواهد بود. در هر دو حالت نتایج بدست آمده مشابه و نزدیک اند و از خود `model_ddn1` بهتر هستند ولی از `model_dn1` کارایی بهتری مشاهده نمی شود. هر دو این مدل ها به لحاظ تعداد پارامترهای وزن (۲۹۰) به `model_dn1` که ۲۸۱ وزن داشت نزدیک اند.

		Loss	Precision	Recall
	model_tt1			
	میانگین آموزش	0.48	0.74	0.83
	میانگین تست	0.48	0.74	0.83
	نتایج روی آموزش	0.48	0.74	0.82
		0.48	0.74	0.85
		0.48	0.75	0.83

 <p>INPUT H1 sigmoid ($\sum 21[X*w]+b$) = 10 out</p> <p>H2 sigmoid ($\sum 10[X*w]+b$) = 5 out</p> <p>DROP2</p> <p>H3 sigmoid ($\sum 5[X*w]+b$) = 2 out</p> <p>OUTPUT sigmoid ($\sum 2[X*w]+b$) = 1 Y</p>	نتایج روی تست	0.48 0.49 0.48	.74 0.73 0.75	0.83 0.85 0.82
اضافه کردن dropout بعد از لایه ۲				
model_tt2				
 <p>INPUT H1 sigmoid ($\sum 21[X*w]+b$) = 10 out</p> <p>DROP1</p> <p>H2 sigmoid ($\sum 10[X*w]+b$) = 5 out</p> <p>H3 sigmoid ($\sum 5[X*w]+b$) = 2 out</p> <p>OUTPUT sigmoid ($\sum 2[X*w]+b$) = 1 Y</p>	میانگین آموزش	0.48	0.74	0.83
	میانگین تست	0.48	0.74	0.83
	نتایج روی آموزش	0.49 0.48 0.47	0.74 0.75 0.74	0.83 0.82 0.85
	نتایج روی تست	0.48 0.48 0.49	0.74 0.75 0.73	0.84 0.81 0.83
اضافه کردن dropout بعد از لایه ۱				

در بخش آزمایش ها دو مدل دیگر هم می سازم که بازهم هدف از ساختشان بیشتر شناخت داده هاست و اینکه آیا امکان ساده سازی بهترین مدل هایی که تا اینجا ساخته شده وجود دارد یا نه؟ اینجا با استفاده از تبدیل pca می خواهم تعداد ورودی های مدل های model_dn1 و model_ddn1 را کاهش دهم. این کار با وجود توضیح اول کار درباره ی گسسته بودن داده ها شاید

عجیب به نظر برسد. مخصوصا اینکه ابعاد داده ها چندان هم بزرگ نیست. Pca یک روش غیرنظارتی است و انجام تبدیل قبل از ورود اطلاعات به شبکه عصبی بیشتر متوجه ارتباط بین ستون های داده هاست و قصد دارم بدانم آیا در بین این ۲۱ فاکتور موثر بر بیماری قلبی می توان ترکیباتی معنادار یافت که مدلی با همین کارایی بهینه ای که تا اینجا به دست آمده آموزش دهد اما با سادگی بیشتر. اول یک بار pca را روی تمام داد ها اجرا می کنم تا توزیع واریانس ها را رسم کنم و بدانم تا چقدر می توانم اندازه ی ورودی را کوچک کنم. برای تبدیل های pca این قسمت از sklearn استفاده شده. نمودار به شکل زیر هست:



به نظر انتخاب عددی بین ۵ و ۱۰ مناسب خواهد بود من تبدیل را با ۱۰ بردار می گیرم و این یعنی لایه ورودی من از ۲۱ به ۱۰ در مدل های قبلی تغییر می کند و نصف می شود. حالا یک تبدیل دیگر به فضای ۱۰ بعد می سازم و در هر یک از ۳ بار آموزش مدل با استفاده از این تبدیل اول داده های آموزش به فضای جدید رفته (هر بار تبدیل pca با همان دسته داده ی آموزش فیت می شود) و بعد وارد شبکه می شود و در پایان برای ارزیابی داده های تست را با همین تبدیل در فضای جدید با شبکه دسته بندی می شوند. اول با یک شبکه ۴ لایه (۳ لایه پنهان) که تغییر یافته ی model_ddn1 هست و بعد با معماری سه لایه (۲ لایه پنهان) تغییر یافته ی مدل model_dn1 این عملیات را انجام دادم. مدل دومی با ۷۰ وزن از تمام مدل هایی که ساخته شده پیچیدگی کمتری دارد ولی جالب اینکه تفاوت عملکرد آن با مدل های بسیار پیچیده و حجیمی که پیش از این ساخته شد چشمگیر نیست. هر چند که هردوی این ها هم روی recall و هم روی precision ضعیف تر هستند.

		Loss	Precision	Recall
model_tt3				

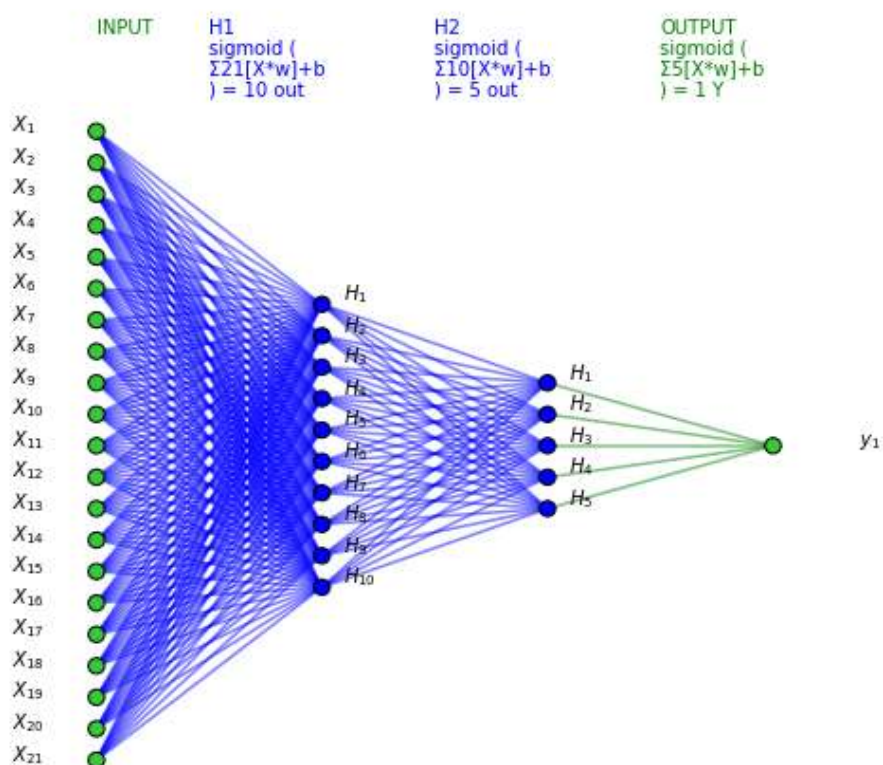
<div><div><div>INPUT</div><div>H1 sigmoid ($\Sigma 10[X*w]+b$) = 10 out</div><div>H2 sigmoid ($\Sigma 10[X*w]+b$) = 5 out</div><div>H3 sigmoid ($\Sigma 5[X*w]+b$) = 2 out</div><div>OUTPUT sigmoid ($\Sigma 2[X*w]+b$) = 1 Y</div></div></div> <tr><td>میانگین آموزش</td><td>0.49</td><td>0.74</td><td>0.82</td></tr> <tr><td>میانگین تست</td><td>0.49</td><td>0.73</td><td>0.81</td></tr> <tr><td rowspan="3">نتایج روی آموزش</td><td>0.49</td><td>0.73</td><td>0.82</td></tr> <tr><td>0.49</td><td>0.74</td><td>0.82</td></tr> <tr><td>0.49</td><td>0.74</td><td>0.82</td></tr> <tr><td rowspan="3">نتایج روی تست</td><td>0.49</td><td>0.74</td><td>0.82</td></tr> <tr><td>0.49</td><td>0.73</td><td>0.81</td></tr> <tr><td>0.49</td><td>0.73</td><td>0.81</td></tr>	میانگین آموزش	0.49	0.74	0.82	میانگین تست	0.49	0.73	0.81	نتایج روی آموزش	0.49	0.73	0.82	0.49	0.74	0.82	0.49	0.74	0.82	نتایج روی تست	0.49	0.74	0.82	0.49	0.73	0.81	0.49	0.73	0.81
میانگین آموزش	0.49	0.74	0.82																									
میانگین تست	0.49	0.73	0.81																									
نتایج روی آموزش	0.49	0.73	0.82																									
	0.49	0.74	0.82																									
	0.49	0.74	0.82																									
نتایج روی تست	0.49	0.74	0.82																									
	0.49	0.73	0.81																									
	0.49	0.73	0.81																									

model_tt4																												
<div><div><div>INPUT</div><div>H1 sigmoid ($\Sigma 10[X*w]+b$) = 5 out</div><div>H2 sigmoid ($\Sigma 5[X*w]+b$) = 2 out</div><div>OUTPUT sigmoid ($\Sigma 2[X*w]+b$) = 1 Y</div></div></div> <tr><td>میانگین آموزش</td><td>0.49</td><td>0.73</td><td>0.81</td></tr> <tr><td>میانگین تست</td><td>0.49</td><td>0.74</td><td>0.81</td></tr> <tr><td rowspan="3">نتایج روی آموزش</td><td>0.49</td><td>0.73</td><td>0.81</td></tr> <tr><td>0.49</td><td>0.74</td><td>0.81</td></tr> <tr><td>0.49</td><td>0.73</td><td>0.82</td></tr> <tr><td rowspan="3">نتایج روی تست</td><td>0.49</td><td>0.74</td><td>0.80</td></tr> <tr><td>0.49</td><td>0.73</td><td>0.81</td></tr> <tr><td>0.49</td><td>0.74</td><td>0.81</td></tr>	میانگین آموزش	0.49	0.73	0.81	میانگین تست	0.49	0.74	0.81	نتایج روی آموزش	0.49	0.73	0.81	0.49	0.74	0.81	0.49	0.73	0.82	نتایج روی تست	0.49	0.74	0.80	0.49	0.73	0.81	0.49	0.74	0.81
میانگین آموزش	0.49	0.73	0.81																									
میانگین تست	0.49	0.74	0.81																									
نتایج روی آموزش	0.49	0.73	0.81																									
	0.49	0.74	0.81																									
	0.49	0.73	0.82																									
نتایج روی تست	0.49	0.74	0.80																									
	0.49	0.73	0.81																									
	0.49	0.74	0.81																									

انتخاب مدل نهایی

با توجه به نتایجی که به دست آمد تا اینجا بهترین عملکرد به طور میانگین روی model_dn1 بوده هم اینکه مقدار پیچیدگی و معماری آن نسبت به بقیه و هم نیبت به نتایجی که به دست داده مطلوب است. به عنوان نتیجه آخر انتخاب مدل یک بار 10fold cross vadidation اجرا می‌کنم و از بین ۱۰ نتیجه به دست آمده میانگین را به عنوان نتیجه قابل انتظار از این مدل

روی این داده ها و بهترین نتیجه را به عنوان مدل نهایی این گزارش انتخاب می‌کنم. در مرحله آخر این مدل انتخاب شده با نتیجه کار بقیه کسانی که در این داده ها مدلسازی انجام داده اند مقایسه خواهد شد.



Model: "DeepNN"

Layer (type)	Output Shape	Param #
input (InputLayer)	[(None, 21)]	0
h1 (Dense)	(None, 10)	220
h2 (Dense)	(None, 5)	55
output (Dense)	(None, 1)	6
Total params: 281		
Trainable params: 281		
Non-trainable params: 0		

Precision	Recall	
0.75	0.82	نتایج میانگین (در 10fold)
0.74	0.86	نتایج شبکه آموزش دیده ی نهایی (بهترین از 10fold)

مقایسه با نتایج سایر کاربرها

در بین کاربرهایی که روی این دیتاست کار طبقه بندی انجام داده بودند بیشتر به دو حالت برخورد و تفاوت این دو استفاده یا عدم استفاده از متوازن سازی داده ها با باز نمونه گیری تصادفی تفاوت محدوده ی نتایجشان را نشان می داد. در بیشتر کارها accuracy اعلام شده بود. بیشتر کارها با مدل های خطی رگرسیون و روش های دسته بندی بر پایه درخت انجام شده اند و شبکه عصبی استفاده ی کمی داشت که در آن معیارهای recall و precision که پایه های مدلسازی من بودند گزارش نشده بودند. تا جایی که امکان داشت مقایسه را با انواع مدل های ارایه شده انجام دادم. در هر مورد توضیح کوتاهی درباره ی تشابهات و اختلاف های روش خودم با روش مورد نظر می دهم. اصلی ترین ضعف کار من نسبت به این موارد کم توجهی به روش ها و ابزارهای آماری برای شناخت بیشتر داده هاست که کار من در آن مراحل بیشتر با فرض و حدس پیش رفته تا با بررسی دقیق و قابل استناد. و اصلی ترین تفاوت کار من با این روش ها هم همان تعیین دو هدف اصلی از کار با داده هاست که معیارهای گزینش مدل را متفاوت می کند.

۱. <https://www.kaggle.com/murattademir/heart-disease-binary-classification>

کار آماری زیادی برای شناخت داده ها انجام داده از جمله استفاده از ماتریس همبستگی و خوشه بندی و محاسبه ی چندک ها و واریانس های ستون ها و نمودار جعبه ای و ... در واقع بیشتر چیزهایی که من از روی مشاهده ی چشمی و از اطلاعات توضیحات داده ها درباره ی داده ها حدس زدم در این کار به صورت مشروح و با محاسبات و نتایج قابل اندازه گیری و دقیق آمده. استانداردسازی روی ستون ها انجام شده. برای طبقه بندی از ۳ روش KNeighbors و RandomForest و xgboost استفاده شده. معیار استفاده شده accuracy هست و در هر ۳ روش ۹۰ درصد گزارش شده. و اگر از روی confusion matrix های ارایه شده precision و recall محاسبه شود مقادیر زیر به دست می آیند :

	Precision	Recall
Kneighbors	0.36	0.08
Randomforest	0.43	0.10
Xgboost	0.52	0.11

واضح است که مدل من روی این مقادیر نتیجه ی بهتری داشتند چون هدف اصلی این کار افزایش accuracy بوده ولی هدف من که در بخش های اول توضیح داده شد استفاده از مدل جهت کار تشخیص بیماری بوده.

۲. <https://www.kaggle.com/abohelal/nueral-network-regression-vs-sklearn-algorithms>

همان مساله ی حدسیات من در مقابل منایج دقیق آماری روی این کار هم هست. اینجا هم مثل کار قبلی استاندارد سازی انجام شده. هم من و هم در اینجا از RandomUnderSampler استفاده شده و یک مدل شبکه عصبی ۱۰۴ و ۱۰۴ با توابع relu برای دو لایه اول و سیگموئید برای لایه خروجی استفاده شده و برای معماری دوم همین را با یک لایه عمیق تر با relu و به صورت ۶۰ و ۶۰ و ۱ ساخته ولی معیاری که در هر دو شبکه گزارش کرده accuracy برابر ۷۸ درصد هست و من این را محاسبه و لحاظ نکردم و در این کار هم معیارهای recall و precision من لحاظ نشده. نتیجه ی کار شبکه ها را نمی توان مقایسه کرد. مدل بعدی KNeighbors و logisticRegression و decisionTree هست که نتایج زیر را گزارش کرده:

	Precision	Recall	Accuracy
Kneighbors	0.66	0.60	0.76
logisticRegression	0.69	0.60	0.78
decisionTree	0.53	0.55	0.69

در این ۳ مدل نسبت به مدل نهایی من نتایج بهتری روی دو معیار precision و recall پیدا نشده.

۳. <https://www.kaggle.com/abohelal/log-knn>

برخلاف من استانداردسازی انجام داده. مثل من از RandomUnderSampler استفاده کرده و با دو مدل KNeighbors و logisticRegression به نتایج زیر رسیده. بخش قابل تمایز این کار نسبت به قبلی ها انجام gridsearch و 5fold برای یافتن بهترین k در مدل knn هست.

	Precision	Recall	Accuracy
logisticRegression	0.73	0.79	0.75
KNeighbors	0.76	0.79	0.77

مدل اول رگرسیون روی precision نزدیک مدل های بهینه ی من عمل می کنند و در مدل نزدیک ترین همسایه از بهترین مدلی که من ساختم روی این معیار نتیجه بهتری گرفته و در عین حال در هر دوی این ها recall هم نتیجه ی خوبی هست اما کار من روی recall با ۸۶ درصد نتیجه ی مطلوب تری هست.

۴. <https://www.kaggle.com/mahdishirmohammadi/heart-disease-binary-classification>

بدون نمونه گیری تصادفی برای حل مشکل عدم توازن با مدل لوجستیک رگرشن کار کرده و با دقت بیشتری برای انتهاب پارامترهای مدل سازی کار شده و نتایج نهایی به دست آمده به شرح زیر است:

	Precision	Recall	Accuracy
logisticRegression	0.54	0.12	0.90

مدل هایی که بدون رفع عدم توازن ساخته می شوند مثل این و مدل های کار اول همه نتایج خوب روی accuracy و precision حدود شانس ۵۰ ۵۰ و recall بسیار پایین دارند.

۵. <https://www.kaggle.com/davipedrosomartins/pucminas-visualizacao-de-dados>

کار آماری زیادی روی ستون ها انجام داده و به لحاظ شناخت داده ها و هم به لحاظ دیداری سازی این شناخت تلاش زیادی شده. در نهایت یک مدل دسته بندی رگرسیون لوجستیک با accuracy برابر ۸۴ ارایه شده که به خاطر تفاوت معیار با نتایج مدل نهایی خودم مقایسه پذیر نیست.

۶. <https://www.kaggle.com/funda06/two-layer-nn-heart-disease>

یک شبکه عصبی دو لایه ۳ و ۳ با تابع سیگموئید تعریف کرده و تمام مراحل ساخت و تعریف مدل و حتی آموزش به روش back propagation را تعریف کرده و با معیار accuracy و loss یادگیری را انجام داده. شیوه ی کار من بیشتر استفاده از کلاس keras.Models و الگوریتم های آماده و در دسترس بود تا راحت تر کار جستجو برای بهترین

مدل را با آزمایش و خطاهای خودم و نه با روش های سرچ الگوریتمی پیش ببرم. آموزش این کار با $acc = 40\%$ به پایان رسیده. به دلیل قبل قابل مقایسه با مدل من نیست.

۷. <https://www.kaggle.com/mahmoudlimam/undersampling-dimensionality-reduction>

در شناخت داده ها با نمودار های توزیع احتمال خوب عمل کرده و با شناخت بهتری نسبت به من از داده ها وارد مرحله مدل سازی شده. مانند کار من از RandomUnderSampler استفاده شده و از جنگل تصادفی برای دسته بندی استفاده کرده که روی داده های آموزش کاملاً فیت شده و روی داده های تست نتیجه زیر را دارد :

	Precision	Recall	Accuracy
RandomForest	0.79	0.87	0.82

که نسبت به مدل من برتری کامل هر دو معیار precision و recall دارد. هم شبکه عصبی و هم جنگل تصادفی از مدل هایی هستند که حجیم هستند و با پیچیدگی زیاد همراهند ولی اینطور که اینجا دیده می شود استفاده از جنگل روی این داده ها نتیجه ی بهتری می دهد.

در مرحله بعد از RobustScaler برای پیش پردازش استفاده از SVM استفاده کرده و نتایج روی SVM :

	Precision	Recall	Accuracy
SVM	0.75	0.83	0.77

به غیر از accuracy که در کار من غایب است ، شبکه عصبی های من در میانگین حالتشان معادل این SVM عمل خواهند کرد. در پله سوم از pca استفاده شده و داده ها را به دو بعد تقلیل داده. در قسمتس که من از pca استفاده کردم با ۱۰ بردار کار کردم و نتایجم را بهبود نداد. اینجا هم کار pca ادامه پیدا نکرده و هدف صرفاً دیداری سازی داده ها در دوبعد بوده و نتیجه دوم SVM بعد از یک کاهش بعد حاصل lda اتفاق می افتد که به صورت زیر است و جز اینکه اندکی روی recall کاهش داشته بقیه بدون تغییر هستند. هدف بررسی همین کاهش بعد و نتیجه ی آن روی عملکرد مدل بوده.

	Precision	Recall	Accuracy
LDA+ SVM	0.75	0.81	0.78

شاخص قابل توجهی که در این کار استفاده شده منحنی roc بود.

۸. <https://www.kaggle.com/mahmoudlimam/model-evaluation-tutorial>

از RobustScaler برای پیش پردازش استفاده کرده، نمودارهای توزیع را بررسی کرده ولی نمونه گیری تصادفی و متوازن سازی انجام نداده. از یک درخت تصمیم برای دسته بندی استفاده کرده و نتیجه همانطور که در ۴ انتظار می رفت به صورت زیر است.

	Precision	Recall	Accuracy
DecisionTree	0.24	0.27	0.85

نتیجه ی رگرسیون روی precision قوت بهتری داشت. این کار هدف آموزشی داشته.

۹. <https://www.kaggle.com/faridrizqis/hertdisease-eda-prediction>

در پیش پردازش و شناخت داده ها نمودارهای جالب و مفیدی رسم شده. MinMaxScaler را روی ۳ ستون انجام داده و با بقیه ستون ها که یا ordinal بودند با کیفی دوحالته کاری نکرده. بدون تغییر پارامترهای دیفالت با مدل های مختلف طبقه بندی انجام داده و روی همه تنها accuracy را گزارش کرده که با مدل من قابل قیاس نیست.

۱۰. <https://www.kaggle.com/cienkang/using-logistic-regression>

بدون پیش پردازش و متوازن سازی و .. از رگسیون لجستیک نتایج زیر را گرفته و مشابه این نتایج را قبلا در ۴ بررسی کردیم. کار جالب انتخاب ویژگی است (البته بدون بررسی های آماری و صرفا از روی خدس و گمان هایی شبیه من دربارهی داده ها) دو تا مدل ساخته و در اولی از تمام ویژگی ها و دو رومی با حذف ۵ ستون همین مدل را آموزش داده. و در آخر کار نتایج را با نامتوازن بودن داده ها توضیح داده.

	Precision	Recall	Accuracy
logisticRegression	0.51	0.13	0.91
feature selection + logisticRegression	0.51	0.11	0.91

۱۱. <https://www.kaggle.com/alexteboul/example-model-easyensembleclassifier>

از کتابخانه ی imblearn از دسته بند EasyEnsembleClassifier استفاده کرده و پیش پردازشی انجام نشده. نتایج را با معیارهای جزیی و زیادی شرح داده که من فقط همین ۳ تایی که اینجا بررسی می شد را می اورم :

	Precision	Recall	Accuracy
EasyEnsembleClassifier	0.25	0.79	0.76

که نتیجه ی جالبی هست چون تمام مدل هایی که تا اینجا ساختم و در کار دیگران مطالعه کردم با precision عمل بهتری داشتند تا recall در خالی که اینجا برعکس است. فکر می کنم اگر مدلسازی من بدون توجه به هدف دوم (precision) انجام می شد و فقط در جهت افزایش recall کلاس ۱ پیش می رفت با این شرایط نزدیکی پیدا می کرد.