تمرین اول پایتون

شبکه‌های عصبی بهار ۱۴۰۲

آریا ابراهیمی ۹۸۲۲۷۶۲۱۷۵

هدف از این تمرین، بررسی تاثیر تغییر تعداد لایه‌ها، تعداد نورون‌ها و تغییر optimizer در یک شبکه عصبی می‌باشد. دیتاست استفاده شده در این تمرین، دیتاست MNIST می‌باشد و برای راحتی کار، کلاسی برای شبکه عصبی چندلایه تعریف شده است که مشخصات شبکه (نرخ یادگیری، تعداد لایه ها، تعداد نورون های هر لایه و ...) را از طریق یک فایل json دریافت کرده و شبکه ای با مشخصات داده شده ایجاد می‌کند. سپس در فایل main.py یک شی از کلاس MLP برای کانفیگ انتخابی (می‌تواند نام یک کانفیگ یا تمامی کانفیگ ها باشد؛ برای آموزش تمامی کانفیگ ها باید از مقدار ورودی all استفاده کرد) ساخته شده و یادگیری می‌شود. مقادیر loss و accuracy در هر ۱۰۰گام با استفاده از SummaryWriter ذخیره می‌شوند (هم برای داده های train و هم برای داده های test که در نمودار های نمایش داده شده از مقادیر برای داده های test استفاده شده است) تا بتوان نتایج را در Tensorboard مشاهده کرد.

۱- بخش اول

۱.۱- بررسی تغییر تعداد لایه ها

میتوان در شکل ۱ مشاهده کرد که با افزایش تعداد لایه ها، روند یادگیری کندتر شده است. برای شبکه‌های سه‌لایه، روند یادگیری تقریبا بعد از ۵۰۰ گام converge شده است ولی شبکه عمیق‌تر با ده لایه و با استفاده از RMSProp optimizar تقریبا در گام های نهایی converge شده است.

با افزایش تعداد لایه‌ها، توانایی مدل برای شناسایی الگوهای پیچیده‌تر بیش‌تر می‌شود، ولی از طرف دیگر ریسک overfit شدن را افزایش میدهم و همچنین باعث می‌شود که شبکه سخت تر optimize شود.

همچنین ممکن است مشکل vanishing gradients نیز پیش بیاید. این مشکل بیان‌گر این است که در شبکه‌های عمیق، اگر گرادیان ها کوچک باشند، در back propagation هرچقدر به عقب بروند کوچک‌تر می‌شوند که باعث می‌شود وزن های لایه های ابتدایی به خوبی تغییر نکنند. از آنجایی که در شبکه استفاده شده در این تمرین از تابع فعال‌ساز ReLU استفاده شده است، احتمال رخداد این مشکل کمتر است و احتمالا دیر converge شدن به‌خاطر زیاد بودن پارامتر ها است.

میتوان مشاهده کرد که شبکه های کوچک تر هم سریع‌تر یادگیری شده‌اند و هم نتایج بهتر یا تقریبا برابری را می‌دهند که میتواند برای این باشد که داده های استفاده شده، پیچیدگی زیادی ندارند و میتوان آنها را در یک فضای کوچکتر جداسازی کرد.(diminishing returns)

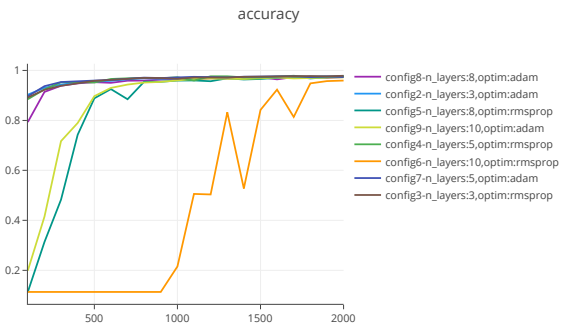

Figure 1: Accuracy of MLP for different number of layers.

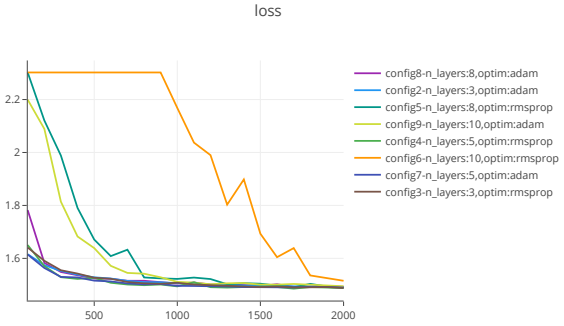

Figure 2: Loss of MLP for different number of layers.

در نتیجه استفاده از شبکه با سه لایه میتواند گزینه بهتری باشد زیرا هم دقت خوبی برای جداسازی داده های MNIST ارائه می‌کند و هم مشکلاتی که گفته شد را ایجاد نمی‌کند.

۱.۲- بررسی تغییر تعداد نورن‌ها

همان‌طور که در شکل۳ مشاهده می‌شود، در این مسئله با افزایش تعداد نورون‌ها به دقت بیشتر قابل توجهی نمی‌رسیم. با افزایش تعداد نورون‌ها در لایه های مخفی، میتوان توابع پیچیده‌تری را تخمین زد ولی در این مسئله نیازی به این کار نیست زیرا شبکه های کوچک هم دقتی نزدیک به شبکه های بزرگ‌تر خواهند داشت و استفاده از شبکه بزرگ فقط زمان یادگیری را افزایش می‌دهد و باعث می‌شود که نیاز به توان محاسباتی بیشتری داشته باشیم. همچنین اگر نمودار مقایسه train و test را در شبکه های با نورون های زیاد بررسی کنیم درمی‌یابیم که خطای train در آنها به ۰ هم رسیده است ولی در test هنوز خطا دارند و انتهای نمودار train و test در حال فاصله گرفتن از همدیگر می‌باشد که نشان دهنده overfit شدن است.

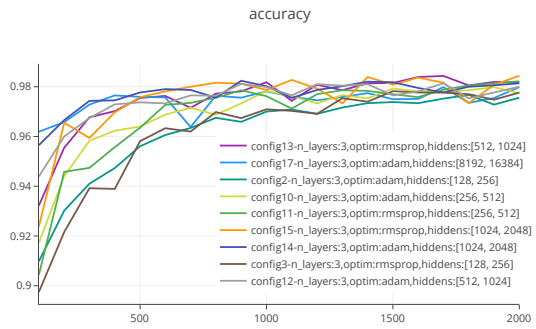

Figure 3: Accuracy of MLP for different number of neurons in hidden layers.

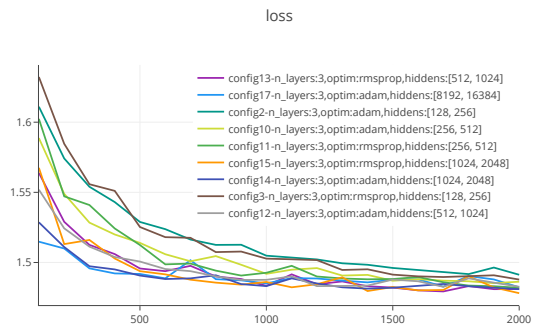

Figure 4: Loss of MLP for different number of neurons in hidden layers.

۱.۳- بررسی تغییر optimizer

در این قسمت به بررسی تاثیر optimizer های مختلف می‌پردازیم. برای این کار از optimzier های زیر استفاده شده است:

۱. SGD : در Stochastic Gradient Descent به جای استفاده از کل داده ها یا یک batch کوچک از داده ها برای آپدیت وزن‌ها، از تنها یک داده استفاده می‌کنیم که باعث سرعت بخشیدن به یادگیری می‌شود ولی ممکن است باعث نویزی شدن آپدیت ها شود.

۲. Adagrad : هدف در این الگوریتم این است که نرخ یادگیری را adaptive کند، به این معنی که براساس شیب جهت هر پارامتر، یک scaling انجام دهد و می‌توان گفت درواقع آپدیت‌ها را نرمال می‌کند. میتوان مشاهده کرد با استفاده از این الگوریتم، پارامتر هایی که گرادیان بزرگ‌تری دارند، آپدیت کوچک‌تری خواهند داشت و گرادیان های کوچک‌تر آپدیت هایی بزرگ‌تر.

$$v_t^w = v_{t-1}^w + (\nabla w_t)^2$$

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_t^w + \epsilon}} * \nabla w_t$$

$$v_t^b = v_{t-1}^b + (\nabla b_t)^2$$

$$b_{t+1} = b_t - \frac{\alpha}{\sqrt{v_t^b + \epsilon}} * \nabla b_t$$

۳. RMSprop : در Adagrad، پارامترها با شدت زیادی آپدیت می‌شوند. ایده RMSprop این است که به جای استفاده از مربع گرادیان ها از روش دیگری به نام exponentially decaying average مربع گرادیان ها استفاده شود.

$$S_t^w = \beta * v_{t-1}^w + (1-\beta)(\nabla w_t)^2$$

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{S_t^w + \epsilon}} * \nabla w_t$$

$$S_t^b = \beta * v_{t-1}^b + (1-\beta)(\nabla b_t)^2$$

$$b_{t+1} = b_t - \frac{\alpha}{\sqrt{S_t^b + \epsilon}} * \nabla b_t$$

۴. Adam: ترکیبی از دو optimizer مومنتوم و RMSprop می‌باشد. در الگوریتم مومنتوم از exponentially weighted averages استفاده

می‌کنیم تا آپدیت ها smooth تر شوند. الگوریتم Adam به صورت زیر است:

$$v_{t,w} = \beta_1 * v_{t,w} + (1 - \beta_1) * (\nabla w_t)$$
$$S_{t,w} = \beta_2 * S_{t,w} + (1 - \beta_2) * (\nabla w_t)^2$$
$$v_{t,w}^{corrected} = \frac{v_{t,w}}{1 - \beta_1^t}$$
$$S_{t,w}^{corrected} = \frac{S_{t,w}}{1 - \beta_2^t}$$
$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{S_{t,w}^{corrected} + \epsilon}} * v_{t,w}^{corrected}$$

$$v_{t,b} = \beta_1 * v_{t,b} + (1 - \beta_1) * (\nabla w_t)$$
$$S_{t,b} = \beta_2 * S_{t,b} + (1 - \beta_2) * (\nabla w_t)^2$$
$$v_{t,b}^{corrected} = \frac{v_{t,b}}{1 - \beta_1^t}$$
$$S_{t,b}^{corrected} = \frac{S_{t,b}}{1 - \beta_2^t}$$
$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{S_{t,b}^{corrected} + \epsilon}} * v_{t,b}^{corrected}$$

حال به بررسی تفاوت optimizer ها در مسئله طبقه‌بندی MNIST با استفاده از MLP می‌پردازیم.

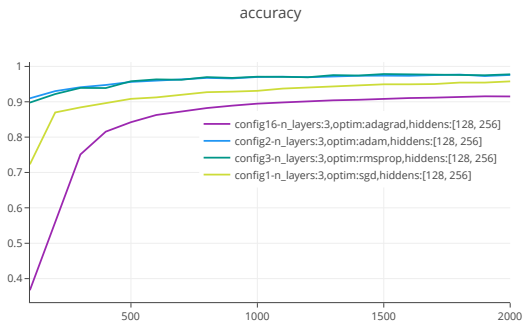

Figure 5: Accuracy of MLP for different optimizers.

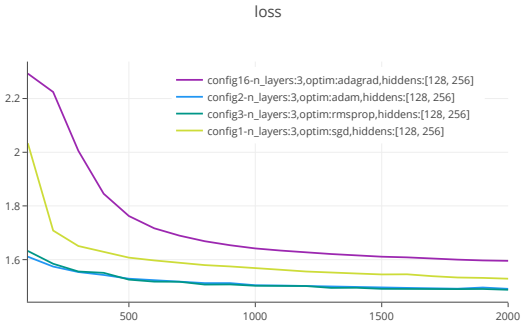

Figure 6: Loss of MLP for different optimizers.

همان‌طورکه مشاهده می‌شود، Adam و RMSprop تقریبا همانند یک‌دیگر عمل کرده‌اند و نتیجه بهتری دارند ولی SGD و Adagrad در iteration های بیشتری converge شده اند.

در نتیجه بهترین شبکه‌ای که trade off بین توان محاسباتی و دقت را رعایت کند می‌تواند به صورت زیر باشد:

برای optimizer از adam استفاده می‌کنیم چون در شبکه سه‌لایه به همراه RMSprop بهترین عملکرد را داشت و در شکل ۱ هم مشاهده شد که اگر شبکه عمیق‌تر شود بهتر از RMSprop عمل می‌کند.

برای عمق شبکه همان‌طور که اشاره شد، شبکه عمیق نیازی نداریم زیرا برای داده های MNIST نیازی به پیچیدگی زیادی نداریم و داده ها به یک شبکه ۳لایه قابل جداسازی هستند.

برای تعداد نورون ها در هر لایه هم می‌توان در شکل ۷ مشاهده کرد که با افزایش آنها accuracy خیلی افزایش پیدا نمی‌کند برای همین از کانفیگ شماره ۱۰ به عنوان شبکه انتخابی استفاده می‌کنیم که ۳ لایه دارد. لایه اول شامل ۲۵۶ نورون است و لایه دوم ۵۱۲ نورون دارد و optimizer استفاده شده Adam است.

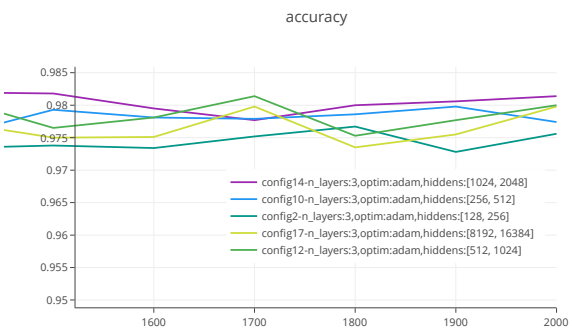

Figure 7: Accuracy of Adam optimizer for 3-layer MLPs with different number of neurons in hidden layers.

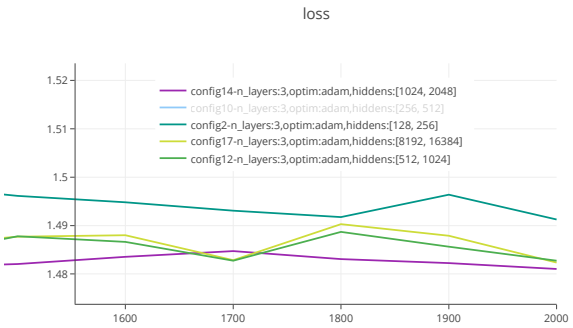

Figure 8: Loss of Adam optimizer for 3-layer MLPs with different number of neurons in hidden layers.

این مدل دوباره با وزن های ابتدایی random uniform ران شده و به نتایج زیر رسیده‌ایم :(در فایل test_properties.ipynb قابل مشاهده است)

Accuracy	Precision	Recall	F1 score
۰.۹۸۰	۰.۹۸۰	۰.۹۸۰	۰.۹۷۹

نمودار ROC در شکل۹ و نمودار یادگیری در شکل ۱۰ نمایش داده شده اند.

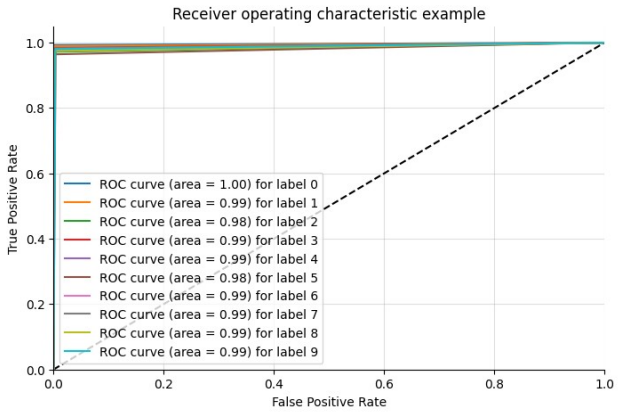

Figure 9: ROC curve of the proposed configuration.

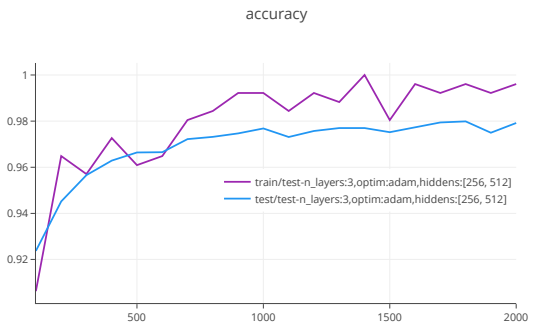

Figure 10: Learning curve of the proposed configuration.

۲- بخش دوم

با آموزش شبکه روی داده‌های ۰ تا ۴، عمل‌کرد شبکه برای همین کلاس ها در تست خوب است و به accuracy تقریبا برابر با ۹۹ درصد می‌رسیم ولی برای داده‌هایی که در مرحله آموزش مشاهده نشده‌اند عمل‌کرد خوبی ندارد و به اشتباه جز کلاس های ۰ تا ۴ تشخیص می‌دهد. برای مثال عدد ۹ که شباهت زیادی به عدد ۴ دارد را در اکثر مواقع ۴ تشخیص میدهد ولی در حالت کلی نمیتواند این اعداد را تمایز دهد و برای ما Black-Box است که چگونه تشخیص می‌دهد. (کد مربوط به این قسمت در test_zero_to_four.ipynb قابل مشاهده است)

برای رفع این مشکل اگر برای آموزش داده‌های ۵ تا ۹ را نداشته باشیم می‌توان از روش های unsupervised برای تشخیص داده‌های دورافتاده استفاده کرد. برای مثال می‌توان نماینده هر کلاس را در حین یادگیری پیدا کرد و در نهایت در زمان تست اگر داده‌ای به شبکه داده شد، نزدیک بودن به مرکز کلاس تخمین زده شده نیز بررسی شود و اگر فاصله از یک threshold بیشتر بود می‌توان داده را به عنوان داده دورافتاده اعلام کرد.

اگر به داده های دیگر (۵ تا ۹) در زمان آموزش دسترسی داشته باشیم میتوان یک خروجی دیگر به شبکه اضافه کرد. (یک لایه Dense با یک نورون و تابع فعال‌سازی sigmoid که درواقع binary classification انجام می‌دهد) تسک مربوط به این خروجی جدید، جداسازی کلاس های ۰ تا ۴ از بقیه کلاس ها است به این صورت که اگر اعداد در بازه ۰ تا ۴ به شبکه داده‌شدند، این خروجی باید عددی نزدیک به ۱ و اگر داده دیگری داده شد باید عددی نزدیک به ۰ خروجی داده شود. میتوان این خروجی جدید را با داده های MNIST یادگیری کرد به این صورت که به کلاس ۰ تا ۴ لیبل های ۱ و به کلاس های دیگر لیبل ۰ اختصاص داده می‌شود و با استفاده از این لیبل ها میتوان تابع loss را محاسبه کرد و شبکه را آموزش داد.

در مرحله تست ابتدا قبل از اعلام لیبل پیش‌بینی شده برای ورودی، مقدار خروجی جدید را بررسی می‌کنیم و اگر مقداری نزدیک ۱ داشت میتوان لیبل پیش‌بینی شده را گزارش کرد و اگر مقدار نزدیک ۰ بود میتوان داده ورودی را به عنوان others پیش‌بینی کرد و به لیبل گزارش شده اهمیت نداد.