امیرحسین احمدی ۹۷۵۲۲۲۹۲ – تمرین دوم

۱- الف) همانطور که در شکل مشخص است، Adam با زیاد شدن تعداد iteration هایش زودتر از بقیه خطای کمتری میدهد. در واقع آدام با دربر گرفتن مزیت های AdaGrad و RMSProp سریع تر از بقیه optimizer ها به جواب رسیده است.

ب) به عوامل زیادی که به ازای هر optimizer باید به صورت جدا بررسی شود. مثلا sgd راحت تر از بقیه در لوکال مینیموم ها گیر میکند و به همین دلیل برای داده های دارای نویز زیاد مناسب نیست. در adagrad برای هر پارامتر learning_rate جدا داریم و اگر تعداد epoch ها زیاد باشد روند همگرایی کند میشود و بهتر است که از optimizer های دیگر استفاده کنیم. هر کدام از optimizer های دیگر نیز نقاط ضعف و قوت خود را دارند ولی معمولا از optimizer، adam در بیشتر مسائل استفاده میشود به دلیلی که در قسمت اول گفته شد.

۲- الف) در ۱۰۰ داده ی اول که همه یک هستند طبیعتن میانگین یک است. با اضافه شدن منفی یک ها میانگین به سمت صفر که میانگین کل داده هاست میرود، ابتدا شیب بیشتری دارد و کم کم شیبش کاهش میابد به این دلیل که هرچه داده ها بیشتر میشوند تاثیر داده ها کمتر میشود در میانگین و وارد شدن یک داده ی جدید تاثیر کمتری میگذارد.

ب) برای قسمت ب، مقدار اولیه صفر است و با زیاد شدن یک ها میانگین به سمت یک رفته و بعد با وارد شدن منفی یک ها میانگین به سمت منفی یک میرود. دلیل این است که در این نوع میانگین گیری تاثیر داده های جدید بیشتر است، زیرا فقط یک ضریب ۰.۱ دارند و داده های قدریمی با هر ایتریشن یک ضریب ۰.۹ نیز در آن ها ضرب میشود و تاثیرشان را کم میکند. برای همین میانگین به سمت داده های جدید میرود.

ج) در این نوع میانگین گیری، ابتدا که مقدار بتا به توان تی نزدیک به یک است و در نتیجه مخرج نزدیک به صفر است، بنابراین بعد از مقدار اول یه جهش زیاد داریم. ولی کم کم با بالا رفتن توان بتا، مخرج زیاد شده و به سمت یک میرود و روی نمودار نیز مشاهده میکنیم که از مقادیر ۳۰ به بعد m_2 به m_1 میل میکند و رفتارش شبیه به m_1 میشود.

د) در این قسمت میبینیم که با زیاد شدن بتا، یک منهای بتا کاهش میابد و تاثیر هر داده ی جدید به خودی خود کم است و به همین دلیل حرکت m_1 به سمت داده های جدید با سرعت کمتری انجام میشود به طوری که تقریبا خط ساف میبینیم. برای m_2 باز مانند قسمت ج، یک جهش در ابتدا داریم و با از بین رفتن تاثیر مخرج در حدود ایتریشن ۳۰ به بعد، نمودار سعی میکند به سمت m_1 حرکت کند ولی با به دلایل گفته شده سرعت بسیار کمتری دارد.

۳- Fashion MNIST دیتاستی از عکس های انواع لباس های فشن است که ۶۰۰۰۰ داده ی آموزشی و ۱۰۰۰۰ داده ی تست دارد. عکس ها ۲۸ در ۲۸ هستند و هر نقطه از عکس یک عدد از ۰ تا ۲۵۵ است که grayscalar هستند. در این دیتا ست ۱۰ لیبل از ۰ تا ۹ وجود دارد.

در ابتدا ۳۰ درصد را برای اعتبار سنجی در نظر گرفتم تا نه خیلی داده های train کم شود و نه به دقت ولیدیشن ضربه ای وارد شود. در ادامه درصد های ۵۰ و ۱۰ نیز مدل را نیز تست کردم که برای مثال با تعداد ۱۲۸ نورون میانی، آمار برای ۳۰ درصد:

train data accuracy: 0.873 loss: 0.367
validation accuracy: 0.864 loss: 0.393
test data accuracy: 0.853 loss: 0.418

برای ۵۰:

train data accuracy: 0.869 loss: 0.379
validation accuracy: 0.860 loss: 0.406
test data accuracy: 0.849 loss: 0.430

و برای ۱۰:

train data accuracy: 0.879 loss: 0.348
validation accuracy: 0.871 loss: 0.371
test data accuracy: 0.859 loss: 0.399

که همانطور که میشود هرچه درصد ولیدیشن پایین تر باشد، دیتای train بیشتری در اختیار داریم و مدل دقت بالا تری دارد و loss پایین میاید. ولی باید توجه داشت اگر درصد ولیدیشن خیلی پایین بیاید، اعتبار اعداد مربوط به ولیدیشن کاهش میابد چون دیتای کمی برای تست در اختیار بوده.
در ضمن با تعداد نرون های میانی متفاوت نیز نتایج مشابهش رخ داد که در نوت بوک موجود است.

حال با ثابت در نظر گرفتن درصد بخش اعتبار سنجی(۳۰ درصد)، accuracy و loss را برای تعداد نرون های متفاوت در لایه مخفی در نظر میگیریم. برای ۱۶ نورون:

train data accuracy: 0.861 loss: 0.400
validation accuracy: 0.853 loss: 0.423
test data accuracy: 0.843 loss: 0.449

۳۲ نورون:

train data accuracy: 0.866 loss: 0.384
validation accuracy: 0.858 loss: 0.410
test data accuracy: 0.848 loss: 0.432

۶۴ نورون:

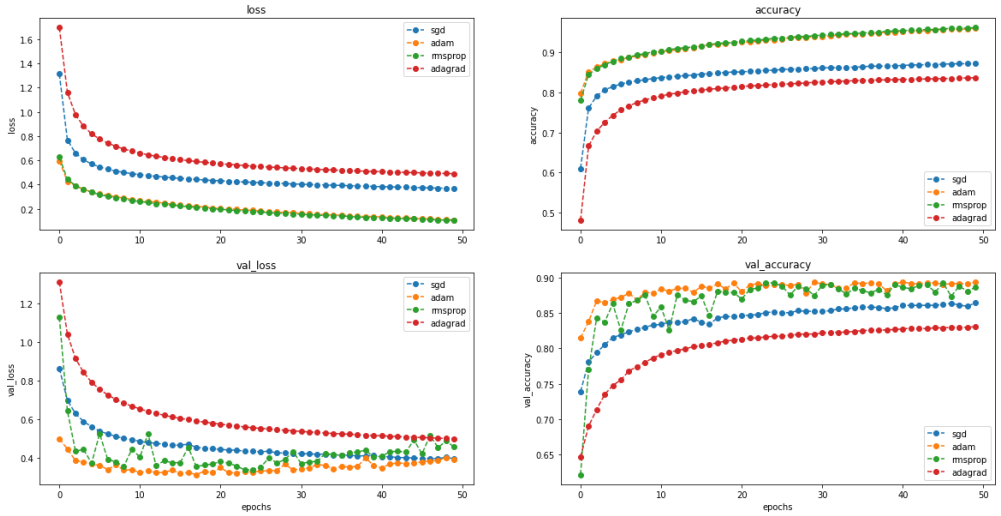
train data accuracy: 0.870 loss: 0.373
validation accuracy: 0.861 loss: 0.401
test data accuracy: 0.851 loss: 0.422

۱۲۸ نورون:

train data accuracy: 0.873 loss: 0.367
validation accuracy: 0.864 loss: 0.393
test data accuracy: 0.853 loss: 0.418

همانطور مشاهده میشود، با زیاد شدن تعداد نورون های لایه میانی دقت بالاتر و loss پایین تر میاید زیرا با زیاد شدن نورون ها مدل میتواند اطلاعات بیشتری ذخیره کند و فانکشنالیتی بهتری داشته باشد ولی از طرفی در مسائل واقعی زیاد کردن نورون ها هزینه زمانی بیشتری نیز میبرد. در آخر این کار را با درصد ولیدیشن های متفاوت نیز انجام دادیم و نتیجه مانند مثال بالا با درصد ۳۰ بود که در نوت بوک نتایج موجود است. پس تفاوتی در روند انتخاب لایه میانی نگذاشت، زیرا تعداد داده ها برای همه ی مدل ها تغییر میکند و شرایط یکسان است.

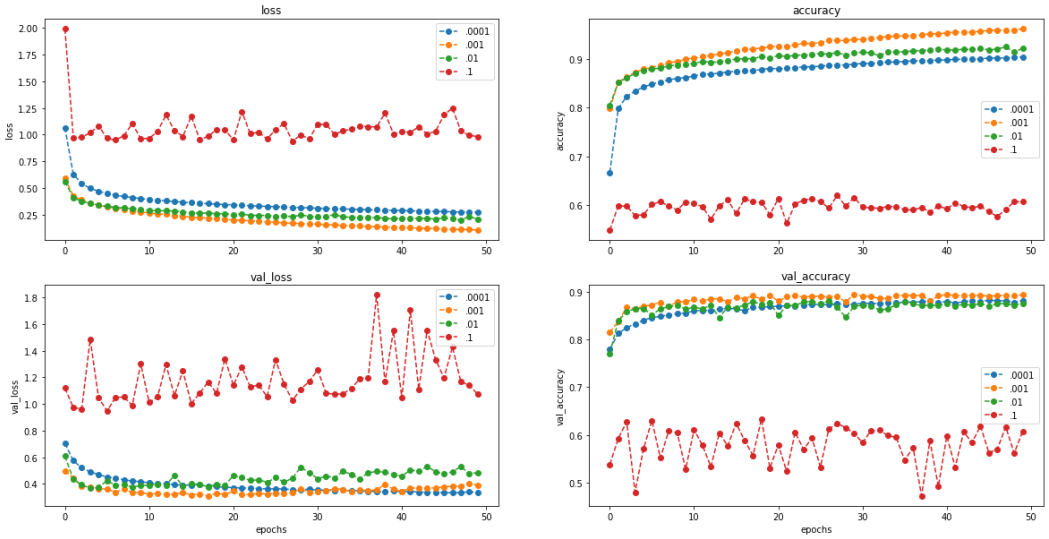
حال که تعداد نورون های لایه مخفی را پیدا کردیم (۱۲۸) میتوانیم optimizer ها را برای این مدل با هم مقایسه کنیم:



مشاهده میکنیم که در هر چهار نمودار که به ترتیب loss و دقت برای داده ی train و loss و دقت برای داده ی ولیدیشن است، adagrad بدترین عملکرد را داشته حتی از sgd هم بدتر و میتوان نتیجه گرفت که برای این مسئله optimizer مناسبی نیست و بعد از آن sgd بدترین عملکرد را داشته.

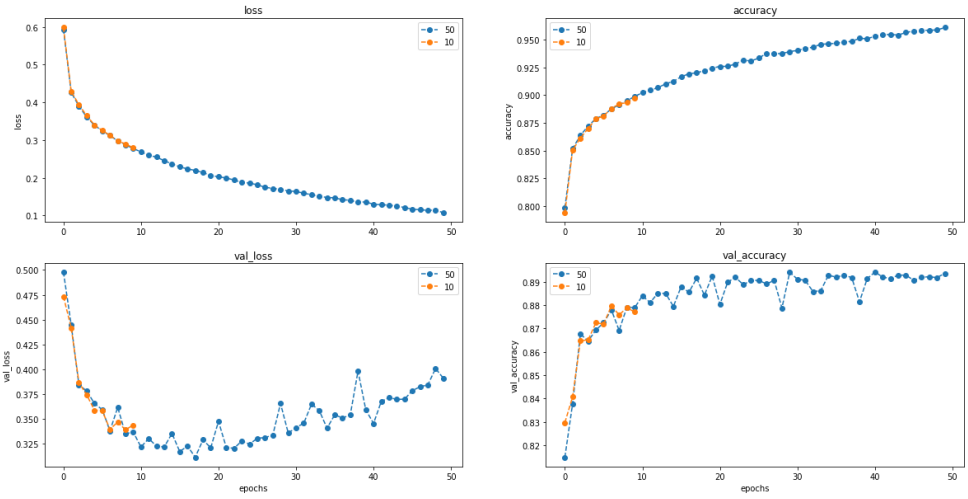
به نظر میاید rmsprop و adagrad برای train عملکرد مشابهی دارند و خوب عمل کرده اند ولی با بررسی ولیدیشن میتوان دید که adam عملکرد بهتری دارد و شبیه تر به train خود عمل کرده و میتوان نتیجه گرفت که rmsprop کمی روی داده ی train، overfit شده که نتوانسته به خوبی آن در ولیدیشن عمل کند. به همین دلیل میتوان نتیجه گرفت که adam برای این مسئله عملکرد مناسبتری از بقیه دارد.

حال با پیدا شدن بهترین optimizer میخواهیم نرخ یادگیری ها را با هم مقایسه کنیم:



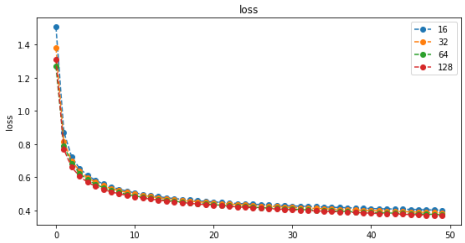
همانطور که مشاهده میشود، نرخ یادگیری ۰.۱ در هر چهار نمودار عملکرد خوبی ندارد و دلیل آن این است که به علت بالا بودن مدل در مینیموم ها نمیماند و نمیتواند همگرا شود. برای ۰.۰۱ همانطور که مشاهده میشود در ابتدای هر چهار نمودار عملکرد خوبی دارد و حتی بهتر از ۰.۰۰۱ عمل میکند ولی به همان دلیل نمیتواند دقت را از یک حدی بالاتر ببرد و در مینیموم ها گیر نمیکند ولی خب بهتر از ۰.۱ عمل میکند. ۰.۰۰۰۱ همانطور که مشاهده میشود به سختی همگرا میشود و خیلی دیرتر یاد میگیرد و احتمالن اگه تعداد epoch ها بیشتر بود میتوانست عملکردش را بهتر کند. ولی مشاهده میشود که در ولیدیشن ها عمل کرد بهتری دارد و حتی loss نهایی بهتری از ۰.۰۰۱ ارائه داده است که نشان میدهد خوب train شده است. ولی درکل میتوان نتیجه گرفت که لرنینگ ریت ۰.۰۰۱ به تناسب خوبی از سرعت همگرایی و دقت رسیده است و عملکرد بهتری نسبت به سایرین دارد.

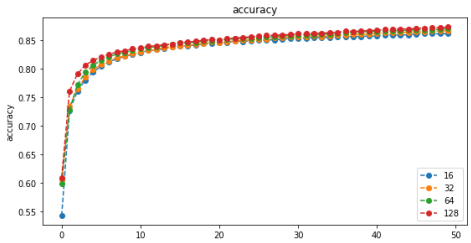
حال مدل را با epoch کمتر(۱۰) امتحان میکنیم تا نتیجه را بررسی کنیم:

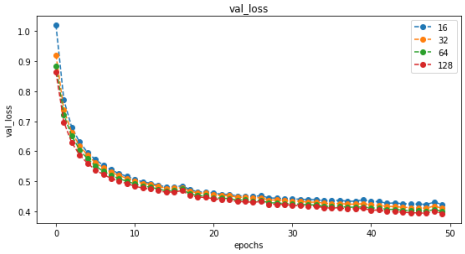


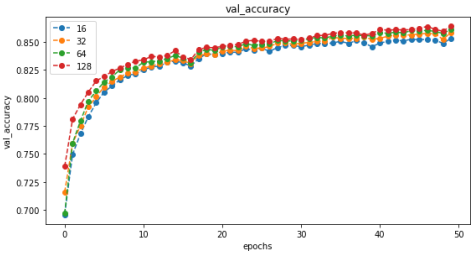
همانطور که مشاهده میکنید epoch ۱۰ برای این مسئله کم است و مدل وقت نمیکند که به همگرایی برسد و نتیجه ی مطلوبی ندارد. نکته ی جالب این است که در دیتای ولیدیشن دقت از epoch ۳۰ به بعد افضایش پیدا میکند که میتواند نشانه ی اور فیت شدن باشد و شاید بهتر باشد epoch را جایی بین ۱۰ و ۵۰ بگذاریم که داده هم به همگرایی برسد و هم اور فیت نشود.

در آخر نمودار مدل با تعداد نرون های لایه مخفی متفاوت را داریم که همانطور که مشاهده میشود روند نمودار ها هم مانند نتیجه ی نهاییست و مدل با نورون های بیشتر، طبق دلایلی که بالاتر گفته شد عملکرد بهتری دارد:


loss
16
32
64
128
loss


accuracy
16
32
64
128
accuracy


val_loss
16
32
64
128
val_loss
epochs


val_accuracy
16
32
64
128
val_accuracy
epochs