بسمه تعالی

دانشکده مهندسی کامپیوتر

مهر ۱۴۰۰

یادگیری عمیق

نام استاد: دکتر محمدی

تمرین سوم

آرمان حیدری

شماره دانشجویی: ۹۷۵۲۱۲۵۲

۱. پاسخ سوال اول

الف) این نمودار نشان دهنده‌ی مقدار کاهش loss، با زیاد شدن epoch یا همان iteration روی دیتاست می باشد. همانطور که در درس خواندیم، بهینه ساز Adam نسبت به بقیه عملکرد بهتری داشته است. به همین دلیل امروزه پرکاربرد ترین بهینه ساز است و با دیتاست ها و شرایط مختلف، هم در شروع تمرین مدل و هم در اواخر آن به خوبی عمل میکند.

در این بهینه ساز ویژگی های مثبت AdaGrad و RMSProp را ترکیب میکنیم و همانطور که در نمودار مشخص است وضعیت الگوریتم Adam به شکل معناداری از هردو آن ها بهتر است. و علاوه بر آنکه سریعتر به نتایج خوبی رسیده است، در نهایت هم نتیجه بهتری داشته است.

اگر احیانا در مسئله ای Adam به نتایج خوبی نمیرسید، یا به هر دلیلی قابل پیاده سازی نبود یا مدل ساده ای نیاز بود یا ...، میتوانیم از SGD+Nesterov به جای آن استفاده کنیم که محبوب ترین جایگزین است. البته در این الگوریتم برنامه ریزی برای نرخ یادگیری اهمیت بیشتری دارد که باید به آن دقت کنیم. هرچند در adam هم این موضوع بی اهمیت نیست.

ب) تعیین بهینه ساز یکی از ابرپارامترهای شبکه ماست. اما هرکدام از optimizer ها میتوانند برتری/ایرادی نسبت به بقیه داشته باشند. اولین و ساده ترین آن ها که gradient decent است، مشکل اصلیش گیر کردن در مینیمم های محلی بود و همچنین به این دلیل که تمام دیتا مستقیم به مدل داده می شود، بسیار مموری اشغال میکند. برای حل مشکل دوم از sgd میتوان استفاده کرد، که البته همچنان مشکل اول را داراست (کمتر). و اگر داده ها نویز زیادی داشته باشد، ممکن است شبکه با batch ها دچار خطا شود و وزن های نویزی بگیرد. که البته اگر دیتا معقول و نویز کم باشد مشکل جدی نیست.

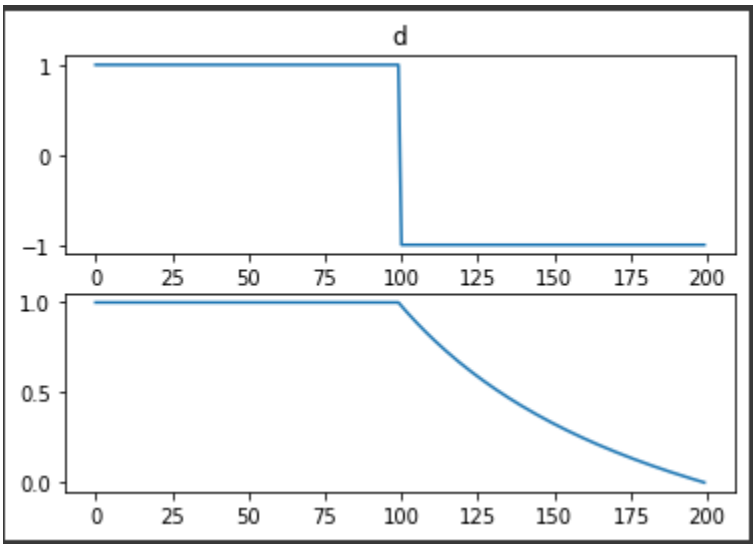
میتوان گفت که بهینه ساز Adam به دلیل ترکیب ویژگی های momentum و تغییر نرخ آموزش و همچنین استفاده از مشتق دوم در آپدیت کردن وزن ها زودتر همگرا می شود و در مجموع پراستفاده تر از سایر روش هاست. اما روش های adaptive شامل AdaGrad و RMSProp و Adam اگرچه مشکل گیر کردن در مینیمم های محلی را ندارند، اما در مموری مشکلات جدی تری دارند. چون هرکدام پارامتر های مختلفی را خارج

از شبکه نیاز دارند. بهینه ساز AdaGrad که برای هر پارامتر نرخ یادگیری مخصوص داریم، با epoch بالا اصلا توصیه نمی شود چون هرچه جلو برویم مقدار نرخ یادگیری کوچک شده و به نقطه ای میرسیم که iteration ها بیهوده است و عملا آپدیت خاصی روی وزن ها صورت نمیگیرد. البته از مزیت های مهم این سه روش این است که در داده های sparse عملکرد خوبی دارند. پس به طور خلاصه، با توجه به سخت افزار اجرای شبکه و همچنین دیتای ما و ... باید بهترین این‌ها را برگزینیم.

۲. پاسخ سوال دوم

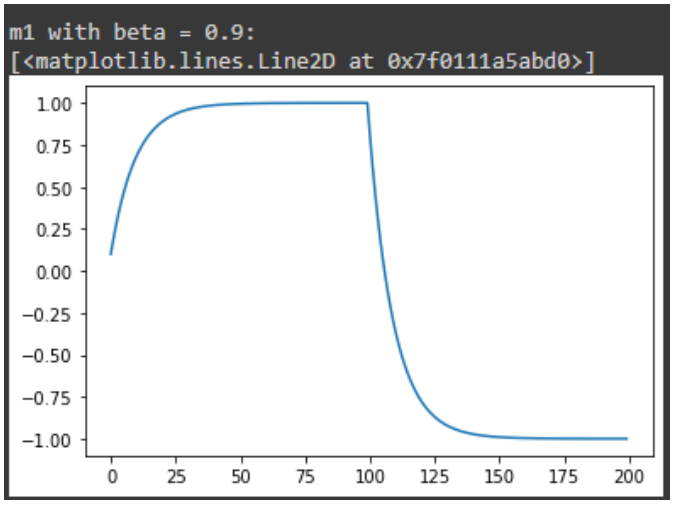
کدهای مربوط به این سوال در فایل HW3.ipynb، section=Question 2 آمده است.

الف)



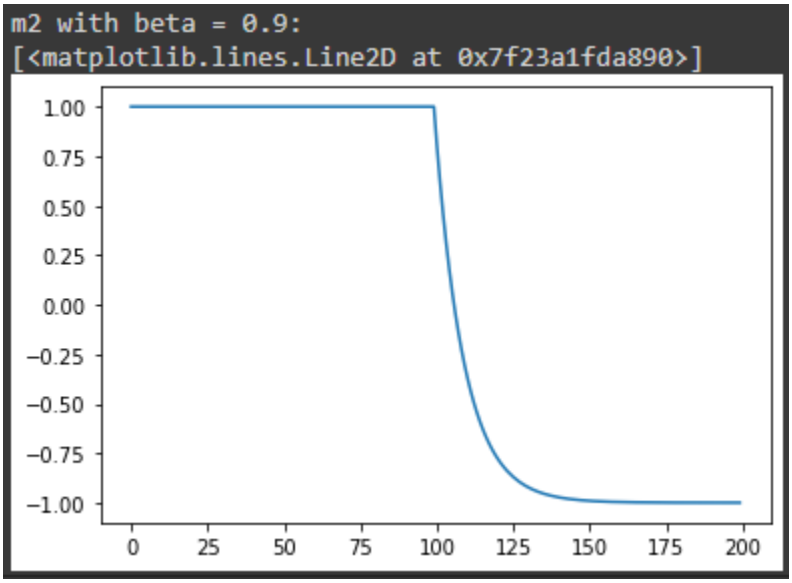
در این قسمت میانگین تجمعی را حساب کرده ایم. که میبینیم چون ۱۰۰ داده اول برابر هستند میانگین ثابت بوده و سپس که داده ها همگی منفی میشوند میانگین هم در هر زمان با اضافه شدن یک مقدار منفی، کمی کاهش می یابد. تا این که در پایان چون مجموع همه ۱ ها و −۱ ها برابر صفر می شود، پس میانگین هم صفر می شود.

ب)



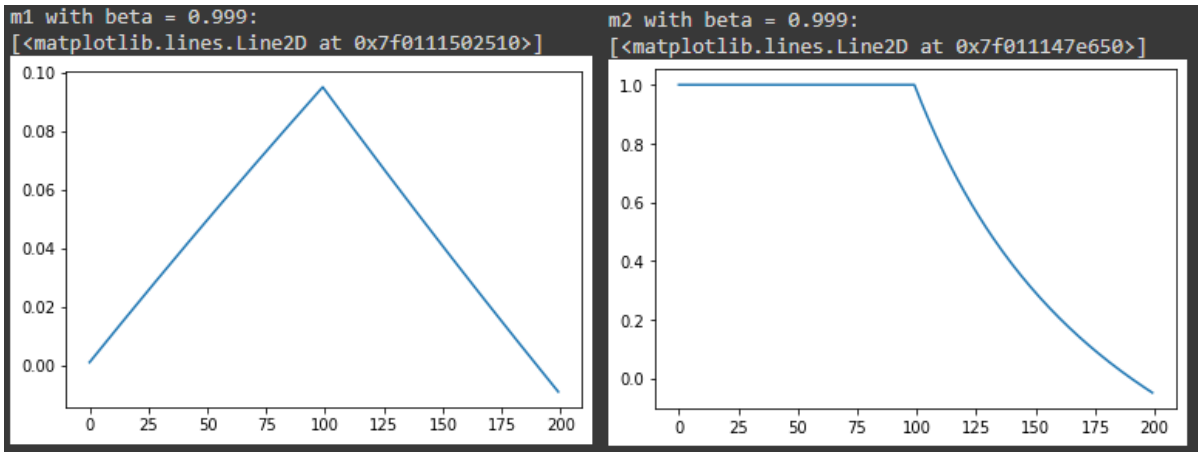
از رابطه به دست آوردن m1 مشخص است که نسبت به میانگین معمولی که در قسمت قبل محاسبه کردیم، بیشتر به داده های جدید توجه می کند. چون با ضریب ۰.۱ آن ها را جمع میکند و در محاسبه میانگین می آورد. پس در ابتدا که میانگین صفر است و داده های ۱ را محاسبه میکند، زود به ۱ می رسد و مادامی که ۱ میبیند میانگین حدود کمی کمتر از ۱ می ماند و به ۱ میل میکند. سپس بسیار سریع تر از قسمت قبل از ۱ به ۰ و سپس حتی به−۱ میل میکند. چون هر چه پیش میرویم −۱ را میبیند و با ضریب ۰.۱ آن را در مقایسه با ۰.۹ که تمام داده های قبلی دارند حساب میکند.

ج)



تفاوت m2 با m1 این است که بر 1-beta^t تقسیم می شود. که می دانیم فقط در زمان های اولیه تاثیر گذار خواهد بود چون به تدریج که t زیاد می شود، این مخرج به ۱ میلی میکند و بی تاثیر در شکل نمودار خواهد بود. اما همانطور که انتظار می رود باعث شده است خیلی زود به میانگین ۱ برسیم و در واقع از همان ابتدا میانگین ۱ باشد و رسیدن از صفر به یک زمان بر نباشد. تغییرات سریع از میانگین ۱ به میانگین −۱ در این روش هم دیده می شود چون فرق خاصی در t های بزرگتر از ۵۰ با قبلی ندارد.

د)



چون بتا از قبل بزرگتر شده است، در هر زمان داده جدید تاثیرش بسیار کمتر از میانگین داده های قبلی است. نسبت (تاثیر داده جدید/تاثیر میانگین) در قسمت های قبل ۰.۱۱ بود اما در این بخش ۰.۰۰۱ است. و این یعنی میانگین های ما خیلی تدریجی تغییر میکنند که این قضیه در شکل سمت چپ کاملا مشخص است و میانگین در زمان ۱۰۰ تازه به ۰.۱ می رسد و سپس شروع به کاهش میکند. که البته کاهش آن نیز مانند افزایشش کند است و فرق خاصی ندارد.

شکل سمت راست هم در مقایسه با m2 با beta=0.9، تغییرش مطابق انتظار در همان سرعت کاهش میانگین بوده است. چون داده های بعد از زمان ۱۰۰ که -۱ هستند در ضریب کوچک ۰.۰۰۱ ضرب می شوند و در زمان ۲۰۰ تازه میانگین را به صفر می رسانند. اما از زمان شروع تا زمان ۱۰۰ تفاوتی ندارد چون m2 با ۱ شروع می شود و داده های ۱ ابتدایی اثری رویش ندارند.

۳. پاسخ سوال سوم

کدهای مربوط به این سوال در فایل HW3.ipynb، section=Question 3 آمده است.

ابتدا با شرایط صورت سوال پیاده سازی را انجام میدهیم و ۱۰ درصد داده‌ها را به عنوان validation در نظر میگیریم. تا ببینیم بهترین تعداد نورون‌ها برای تنها hidden layer ما چیست.

```
==============================] - 3s 2ms/step - loss: 0.4100 - accuracy: 0.8603 - val_loss: 0.4152 - val_accuracy: 0.8523
backs.History at 0x7f4f2d1d4590>

] model.evaluate(test_images, test_labels)

313/313 [==============================] - 0s 1ms/step - loss: 0.4470 - accuracy: 0.8433
```

شکل ۱: number of hidden neurons=128

```
==============================] - 3s 2ms/step - loss: 0.4178 - accuracy: 0.8576 - val_loss: 0.4224 - val_accuracy: 0.8513
backs.History at 0x7f4f2c833910>

model.evaluate(test_images, test_labels)

313/313 [==============================] - 0s 1ms/step - loss: 0.4470 - accuracy: 0.8433
```

شکل ۲: number of hidden neurons=64

```
==============================] - 3s 2ms/step - loss: 0.4292 - accuracy: 0.8531 - val_loss: 0.4347 - val_accuracy: 0.8435
backs.History at 0x7f4f2c68a210>

[44] model.evaluate(test_images, test_labels)

313/313 [==============================] - 0s 1ms/step - loss: 0.4636 - accuracy: 0.8384
```

شکل ۳: number of hidden neurons=32

```
==============================] - 3s 1ms/step - loss: 0.4397 - accuracy: 0.8480 - val_loss: 0.4425 - val_accuracy: 0.8452
backs.History at 0x7f4f2c551690>

[48] model.evaluate(test_images, test_labels)

313/313 [==============================] - 0s 1ms/step - loss: 0.4723 - accuracy: 0.8334
```

شکل ۴: number of hidden neurons=16

که میبینیم مطابق انتظار با ۱۲۸ عملکرد در داده های validation و test بهتر بوده است. و نورون های بیشتر باعث overfit کردن شبکه نشده است.

الف) مجموعه داده شامل عکس هایی grayscale از لباس های فشن هست که شامل ۶۰۰۰۰ داده آموزشی و ۱۰۰۰۰ داده ی آزمون بوده است. این داده ها تصاویر دوبعدی بوده ۲۸*۲۸ که هر کدام از خانه های آن ها عددی بین ۰ تا ۲۵۵ است. بنابراین داده های تمرینی رنک ۳ تنسور با shape=(60000,28,28) هستند و داده های تست با shape=(10000,28,28) هستند. همچنین label های داده های آزمون به صورت (60000,) و label های داده های تستی به صورت (10000,) داده شده اند. که تعلق هر کدام از عکس ها به کلاس های ۰ تا ۹ را نشان می دهند. پس این دیتاست ۱۰ کلاس دارد.

برای بهتر حل کردن قسمت های بعدی از نرمال سازی داده های تست و آموزشی استفاده میکنیم.

ب) قاعدتا مدل ما باید مقدار قابل توجهی داده آموزشی داشته باشد و داده های اعتبارسنجی باید با هدف فهمیدن دقت مدل در هر epoch استفاده شوند. پس نیازی نیست که خیلی زیاد باشند و فقط باید تعدادشان در حدی باشد که دقت را معقول به دست آورند. با توجه به این که مجموعا ۶۰۰۰۰ داده برای آموزش و اعتبارسنجی داریم، آن ها را به نسبت ۹۰ درصد به ۱۰ درصد تقسیم کردم و ۶۰۰۰ داده برای اعتبارسنجی عدد مناسبی است. چون در حدی بزرگ هست که دقت مدل را حساب کند، و آن قدر هم زیاد نیست که به تعداد داده های آموزشی لطمه بزند.

ج)

Train accuracy = 86%, validation accuracy=85.2%, test accuracy=84.3%

اعداد بالا با شرایط گفته شده در صورت سوال، و یک لایه مخفی با ۱۲۸ نورون به دست آمده اند.

نکته ای که به نظرمان میرسد، این است که دقت داده های آموزشی باید اختلاف معنی داری با داده های اعتبارسنجی و تست که توسط شبکه دیده نشده اند، داشته باشد. و در اینجا اینطور نیست پس شبکه ایده آل نبوده است یا این که دیتا کم بوده است. نظر من این است که یک لایه مخفی احتمالا کفایت نمیکند و به همین دلیل دقت داده های آموزشی نسبتا پایین است. چون شبکه باید تا حد خوبی روی داده های آموزشی fit شود. البته مشکل overfit یا underfit نداشته ایم چون ابرپارامترها همگی حدودا مناسب انتخاب شده اند.

د) ابرپارامترهای دیگر به جز validation split و hidden layer neurons را روی مقادیر قبلی ثابت میگذاریم. سپس شبکه را در هر حالت آموزش و تست میکنیم و جدول زیر را تشکیل میدهیم:

Validation split = 0.1

HIDDEN LAYERS	TRAIN ACCURACY	VAL ACCURACY	TEST ACCURACY
16	**84.8**	**84.5**	**83.3**
32	**85.3**	**84.3**	**83.8**
64	**85.7**	**85.1**	**84.2**
128	**86**	**85.2**	**84.3**

Validation split = 0.2

HIDDEN LAYERS	TRAIN ACCURACY	VAL ACCURACY	TEST ACCURACY
16	**84.8**	**84**	**83**
32	**85**	**84.3**	**83.3**
64	**85.4**	**84.8**	**84**
128	**85.7**	**85**	**84**

Validation split = 0.3

HIDDEN LAYERS	TRAIN ACCURACY	VAL ACCURACY	TEST ACCURACY
16	**84.3**	**83.7**	**82.6**
32	**85.1**	**84.4**	**83.5**
64	**85**	**84.3**	**83.4**
128	**85.4**	**84.7**	**83.7**

میبینیم که همواره دقت برای 128>64>32>16. چون درواقع validation split ویژگی است که بر سایر ابرپارامترها تاثیر گذار نیست. و میتوانیم مستقل از آن حالت بهینه را به دست آوریم.

اما از نتایج واضح است که با تعداد نورون ۱۲۸ و validation split = 0.1 به بهترین دقت در تست رسیده‌ایم.

ه) پس از اجرای شبکه با ثابت نگه داشتن تمام پارامتر ها و صرفا تغییر دادن optimizer بین سه حالت گفته شده در صورت سوال، نتایج نهایی زیر به دست آمد:

Train loss: Adam < RMSprop < Adagrad

Train accuracy: Adam > RMSprop > Adagrad

Val loss: Adam < RMSprop < Adagrad

Val accuracy: Adam > RMSprop > Adagrad

Test loss: Adam < RMSprop < Adagrad

Test accuracy: Adam > RMSprop > Adagrad

علاوه بر دقت نهایی که Adam بهتر از بقیه عمل کرده است، در طول آموزش هم نوسان کمتری نسبت به دو روش دیگر داشت. و این موضوع که ویژگی های مثبت این دو الگوریتم را ترکیب کرده است مشخص بود. اما همه optimizer ها زود به دقت ۸۰ درصد رسیدند، و از نظر سرعت اولیه مناسب داشتن روش های خوبی هستند. البته در این مورد هم Adam کمی زودتر رسید.

حتی دقت نهایی با Adam و hidden units=128 و validation split = 0.1 به مقادیر زیر رسید که از SGD که قسمت های قبلی سوال را با آن حل کردیم هم بسیار بهتر است:

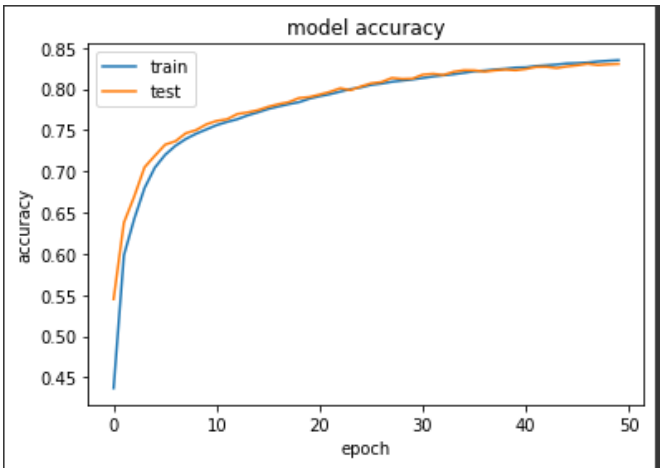
```
1688/1688 [==============================] - 4s 2ms/step - loss: 0.0943 - accuracy: 0.9651 - val_loss: 0.4694 - val_accuracy: 0.8920
<keras.callbacks.History at 0x7f3a75cbd210>
```

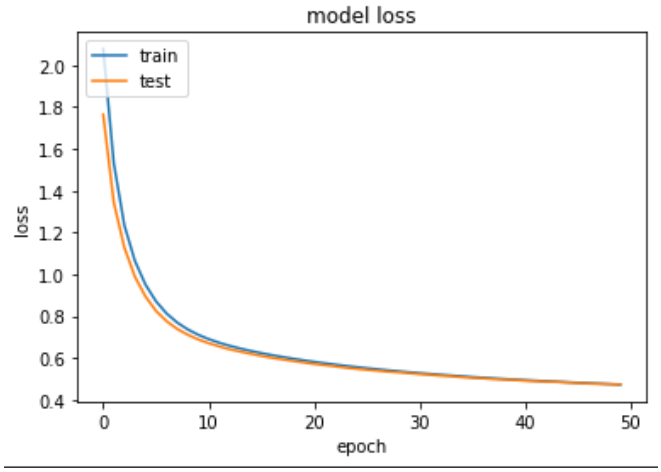
```
model.evaluate(test_images, test_labels)
```

```
313/313 [==============================] - 0s 1ms/step - loss: 0.5209 - accuracy: 0.8854
```

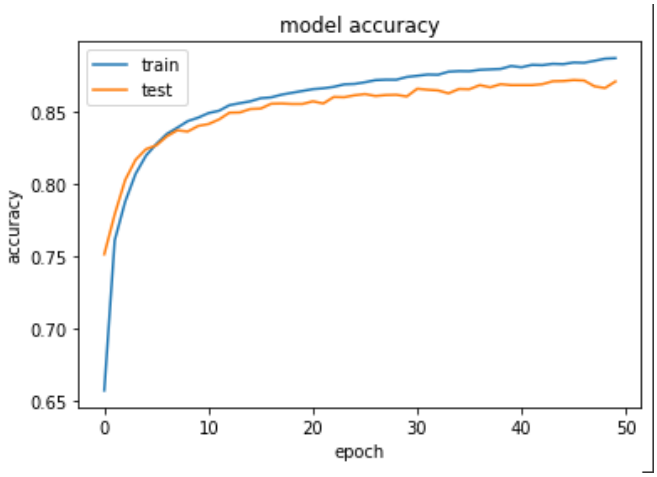
و) حال با ثابت نگه داشتن همه مجهولات از جمله بهینه ساز Adam، فقط نرخ آموزش را تغییر میدهیم و نمودار های زیر به دست می آید:

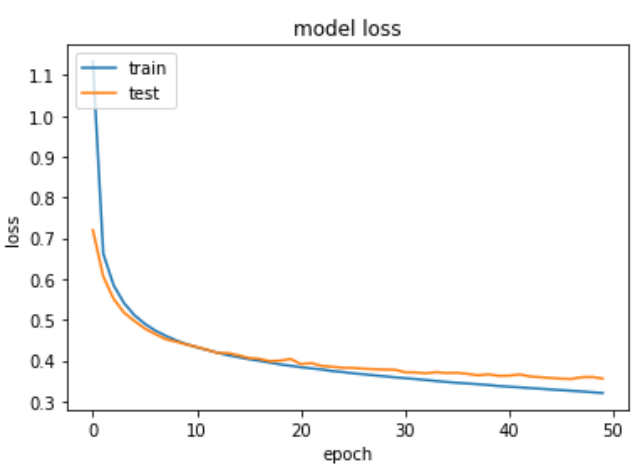
Learning rate = 0.0001:



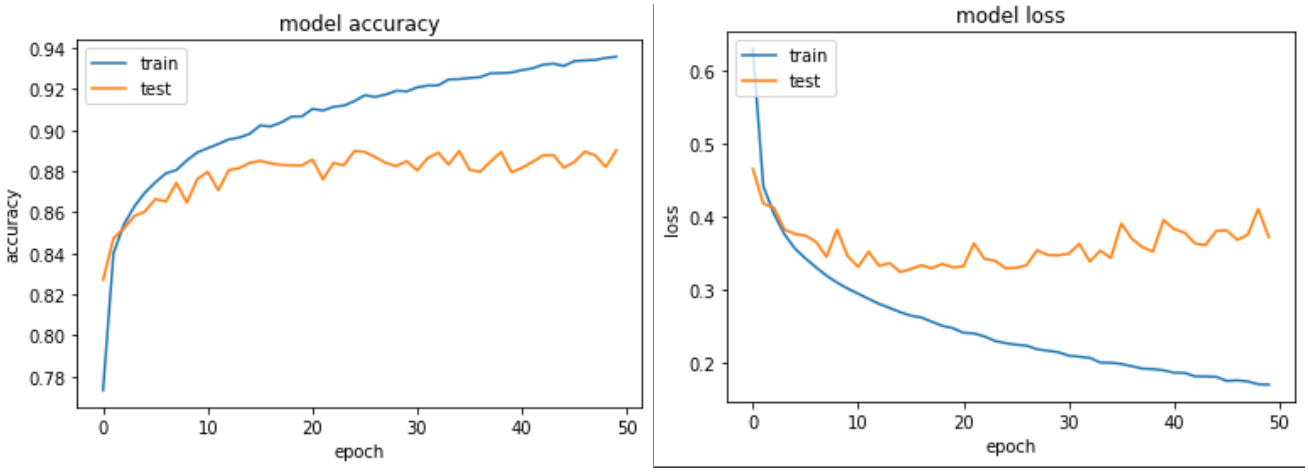


Learning rate = 0.001:

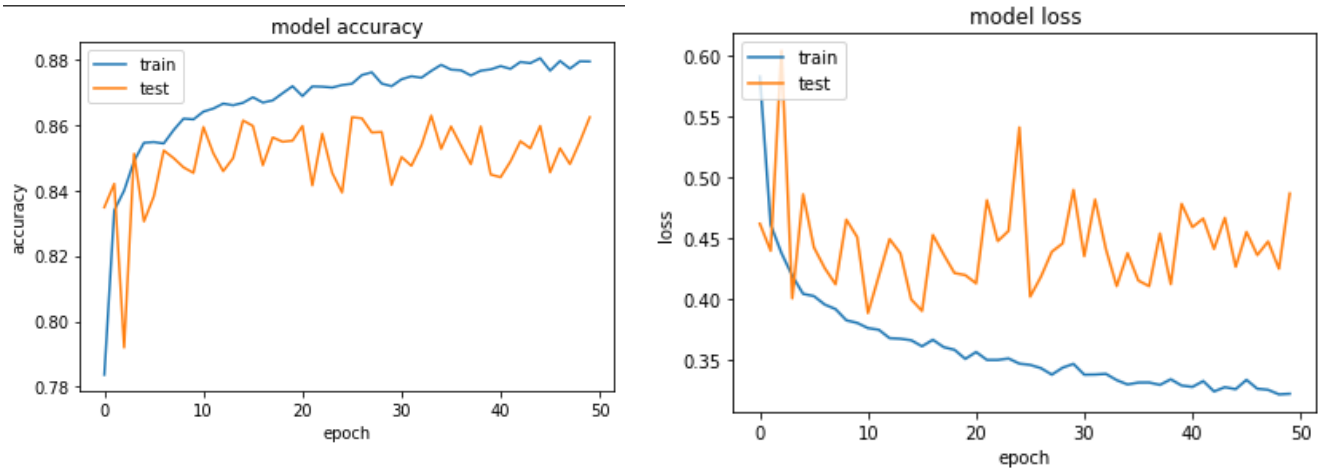




Learning rate = 0.01:



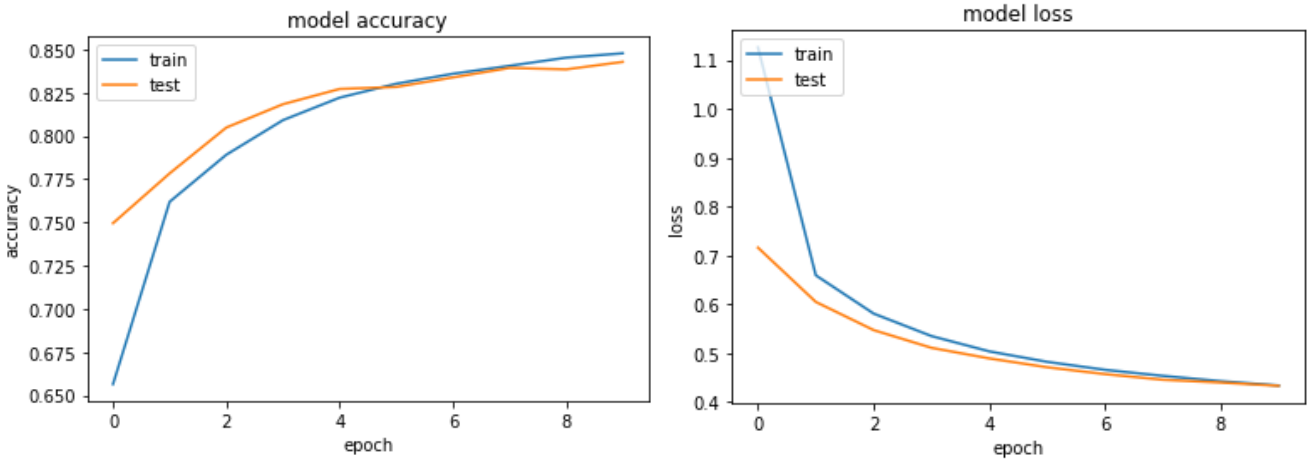
Learning rate = 0.1:



با مقایسه شکل های بالا متوجه میشویم که وقتی نرخ آموزش ۰.۰۰۰۱ بوده، شبکه به آرامی به نقطه بهینه نزدیک شده است و نوسان خاصی ندارد اما سرعتش کم است. نرخ آموزش ۰.۰۰۱ اگرچه بهترین عملکرد در داده های تست را نسبت به ۰.۰۱ نداشته، اما ثبات و عدم نوسان شبکه بسیار بهتر است و به مرور روی داده های validation هم پیشرفت کرده در حالی که ۰.۰۱ نرخ آموزش نسبتا بزرگی بوده و نوسان شبکه زیاد بوده است. به خصوص در validation loss عملکرد رو به بهبود در آن دیده نمی شود.

نرخ آموزش ۰.۱ هم که بسیار بزرگ بوده و شبکه به جای حرکت به ثبت نقطه بهینه فقط نوسان می کند.

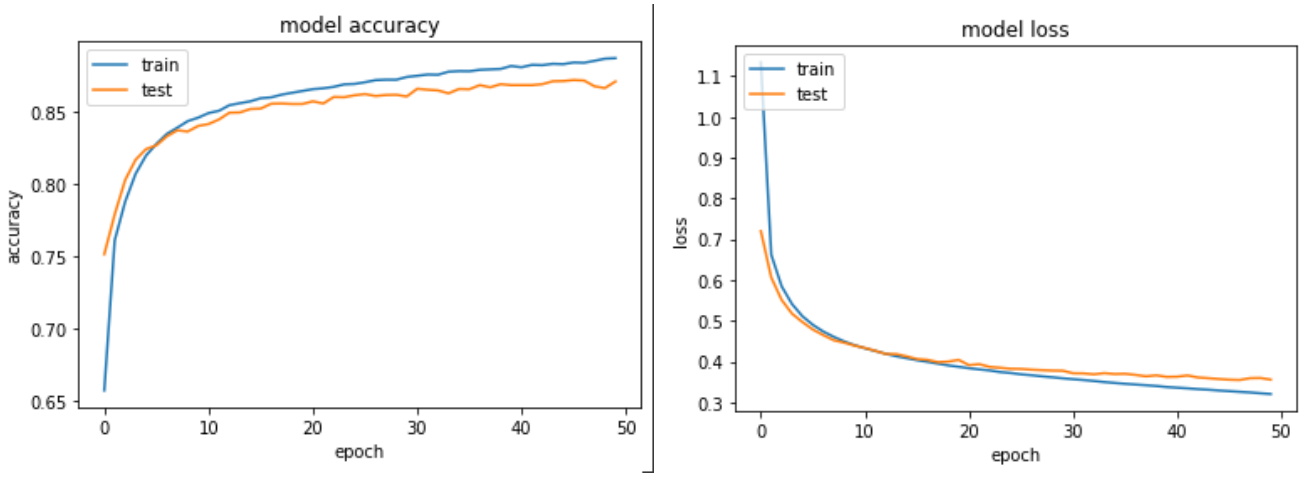
ز) خیر شبکه همگرا نمیشود چون 10 بار اجرا عدد بسیار پایینی برای تکمیل آموزش شبکه است. از نمودار زیر میتوان این را متوجه شد:



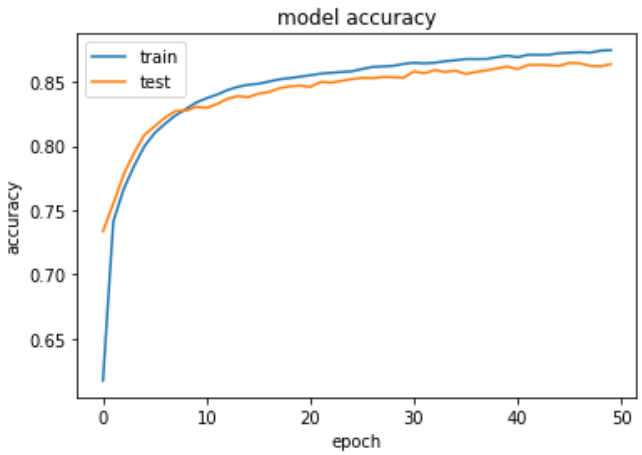
هنوز خط accuracy و همچنین خط loss حالت افقی که به معنی میل کردن و همگرایی به عددی خاص است به خود نگرفته اند. البته چون از بهینه ساز Adam و learning rate مناسب ۰.۰۰۱ استفاده شده است، نتایج آنچنان هم ایراد ندارد. اما بازهم میتوان از نمودار متوجه شد که آموزش باید ادامه پیدا کند.

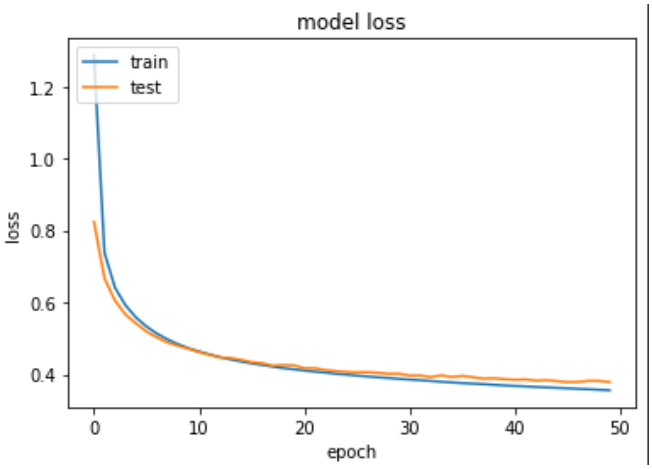
ح) توضیحات و پاسخ این بخش را قبل از بخش الف و همچنین در قسمت د آورده ام. شکل نمودارها:

Hidden units = 128:

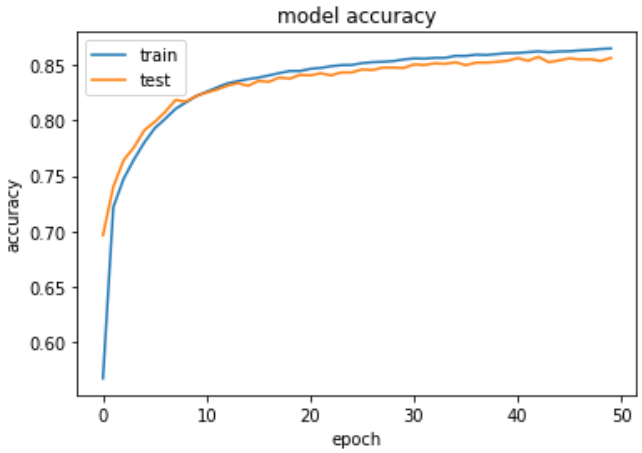


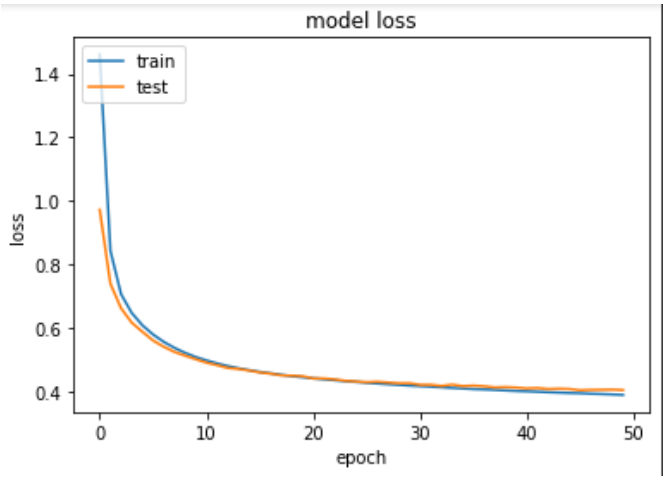
Hidden units = 64:



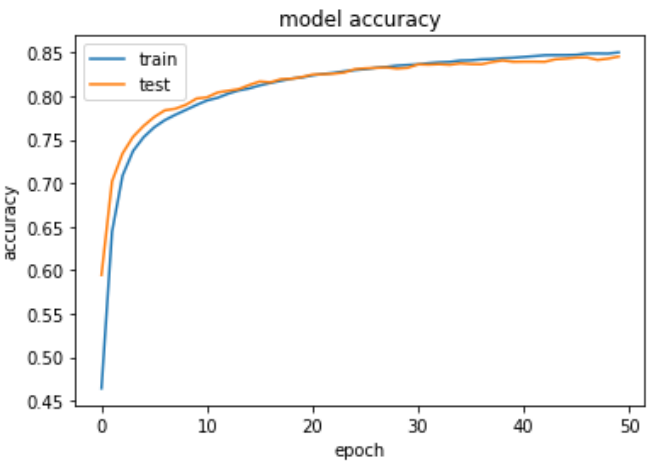


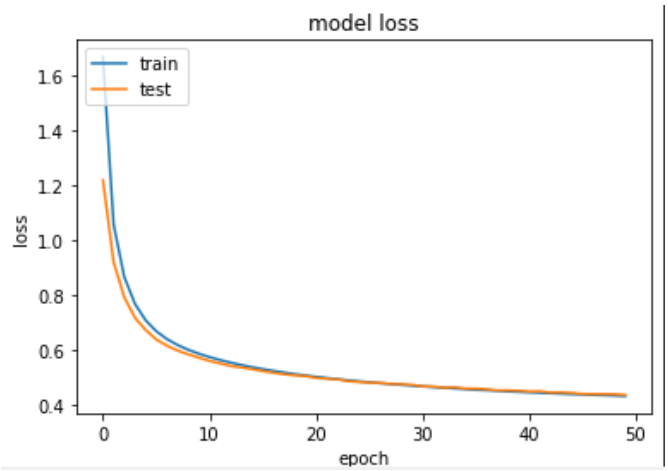
Hidden units = 32:





Hidden units = 16:





میبینیم که شبکه در هیچ حالتی عملکرد خیلی بدی ندارد، چون تمام پارامترهای optimizer و learning rate و validation split و epoch را در بهترین حالت خود گذاشته ایم و داده ها هم نرمال شده اند. اما بازهم دقت با کاهش تعداد نورون های لایه مخفی، کم شده و مطابق توضیحات اولیه‌ام، ۱۲۸ بهترین است.