به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین Extra

نام و نام خانوادگی	فربد سیاهکلی – سعید شکوفا
شماره دانشجویی	810198418 - 810198510
تاریخ ارسال گزارش	1401.09.28

فهرست

پاسخ 1. تشخیص تقلب با استفاده از شبکه عصبی 1

۱-۱. پیاده سازی مقاله 1

پاسخ ۲ – Liveness Detection 2

۲-۱. پیاده سازی 2

پاسخ ۳ – تشخیص کاراکتر نوری 3

۳-۱. پیاده سازی 3

شکل‌ها

شکل 1 معماری شبکه Autoencoder 1
شکل 2 معماری شبکه کلاسیفیکیشن 2
شکل 3 تاثیر Sampling 3
شکل 4 معماری شبکه Denoising 3
شکل 5 نحوه آموزش شبکه Denoising 3
شکل 6 نمودار خطا برای داده آموزش و تست 3
شکل 7 نمودار Accuracy و Recall به ازای Threshold های مختلف 4
شکل 8 معادلات به روز رسانی برای SGD 3
شکل 9 معادلات به روز رسانی برای SGD به همراه Momentum 3
شکل 10 نحوه یادگیری در Adagrad 4
شکل 11 پیاده سازی Adadelta 4
شکل 12 میانگین وزنی نمایی برای شیب ها 5
شکل 13 میانگین وزنی نمایی برای گرادیان ها 5
شکل 14 معادلات به روز رسانی وزن های و بایاس در Adam 5
شکل 15 مقادیر خطا در هنگام آموزش برای 30 ایپاک 6
شکل 16 تمامی نتایج خواسته شده برای بهینه ساز Adam 6
شکل 17 مقادیر خطا در هنگام آموزش برای 30 ایپاک 7
شکل 18 تمامی نتایج خواسته شده برای بهینه ساز SGD 7
شکل 19 مقادیر خطا در هنگام آموزش برای 30 ایپاک 7
شکل 20 تمامی نتایج خواسته شده برای بهینه ساز Adadelta 8

جدول‌ها

No table of figures entries found.

پاسخ 1. تشخیص تقلب با استفاده از شبکه عصبی

۱-۱. پیاده سازی مقاله

1- تعدادی از چالش‌های مرتبط با شناسایی کارت اعتباری:

- نمایه رفتار متقلبانه پویا است، یعنی تراکنش‌های متقلبانه شبیه تراکنش‌های قانونی هستند.
- مجموعه داده های تراکنش کارت اعتباری به ندرت در دسترس هستند و به شدت نامتعادل هستند.
- انتخاب بهینه ویژگی (متغیرها) برای مدل ها.
- معیار مناسب برای ارزیابی عملکرد تکنیک ها بر روی داده های تقلب کارت اعتباری.
- عملکرد شناسایی تقلب در کارت اعتباری تا حد زیادی تحت تأثیر نوع روش نمونه‌گیری مورد استفاده، انتخاب متغیرها و تکنیک‌های شناسایی مورد استفاده قرار می‌گیرد.

2- معماری شامل دو بخش است:

- شبکه Denoised Autoencoder: آنها یک Autoencoder 7 لایه برای فرآیند حذف نویز داده طراحی کرد. پس از اینکه مجموعه داده آموزشی متعادلی از Oversampling بدست آورده شد، نویز گاوسی را به مجموعه داده آموزشی اضافه می کنند، سپس مجموعه داده آموزشی را به این Autoencoder وارد می کنند. پس از آموزش این مدل، این Autoencoder قابلیت حذف داده‌های آزمایشی در فرآیند پیش‌بینی را دارد.

Dataset with noise (29)
Fully-Connected-Layer (22)
Fully-Connected-Layer (15)
Fully-Connected-Layer (10)
Fully-Connected-Layer (15)
Fully-Connected-Layer (22)
Fully-Connected-Layer (29)
Square Loss Function

شکل 1 معماری شبکه **Autoencoder**

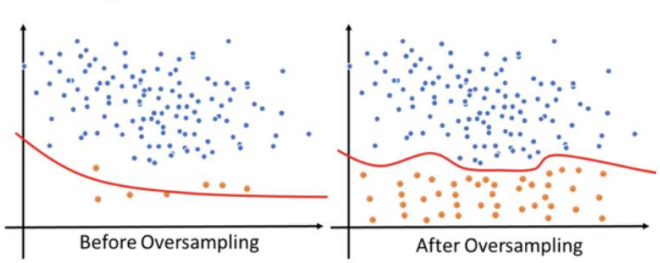
- شبکه Classifier: گروه ما یک Autoencoder 6 لایه برای فرآیند حذف نویز داده طراحی کرد. پس از اینکه مجموعه داده‌های آموزشی Denoise شده را از Denoised Autoencoder دریافت کردیم، مجموعه داده‌های آموزشی را به این طبقه‌بندی‌کننده شبکه عصبی کاملاً متصل عمیق وارد می‌کنیم. در پایان، ما از SoftMax با Cross-entropy به عنوان تابع ضرر برای طبقه بندی نهایی استفاده می کنیم.

Denoised Dataset (29)
Fully-Connected-Layer (22)
Fully-Connected-Layer (15)
Fully-Connected-Layer (10)
Fully-Connected-Layer (5)
Fully-Connected-Layer (2)
SoftMax Cross Entropy Loss Function

شکل 2 معماری شبکه کلاسیفیکیشن

3- الگوریتم های رفع Imbalancement مورد استفاده در مقاله:

- Oversampling: تکنیکی است که برای مقابله با مجموعه داده های نامتعادل استفاده می شود، موضوع آن برای ایجاد نمونه کلاس خاصی است تا توزیع کلاس مجموعه داده اصلی متعادل شود. مزایای استفاده از نمونه برداری بیش از حد نشان داده شده است
- Synthetic Minority Oversampling Technique: SMOTE (تکنیک نمونه برداری بیش از حداقلیت مصنوعی) یکی از محبوب ترین تکنیک های نمونه برداری بیش از حد است. برای ایجاد یک نقطه داده مصنوعی، ابتدا باید یک k خوشه نزدیکترین همسایه را در فضای ویژگی پیدا کنیم، سپس به طور تصادفی یک نقطه را در این خوشه پیدا کنیم، در نهایت از میانگین وزنی برای "جعل" نقطه داده جدید استفاده کنیم.



شکل 3 تاثیر **Sampling**

4- پیاده سازی و آموزش:

به ترتیب ابتدا داده را لود کردیم و sampling مدنظر را پیاده سازی کردیم. سپس دو مدل مورد استفاده را تعریف کرده و در ادامه اقدام به آموزش آنها کردیم.

برای بخش denoising معماری، نحوه آموزش و نتایج به شرح زیر هستند:

```
input_dim = X1_train.shape[1] #29
input_layer = Input(shape=(input_dim, ))
encoder1 = Dense(22, activation="tanh", activity_regularizer=
                        regularizers.l1(learning_rate))(input_layer)
encoder2 = Dense(15, activation="relu")(encoder1)
encoder3 = Dense(10, activation="tanh")(encoder2)
decoder1 = Dense(15, activation='relu')(encoder3)
decoder2 = Dense(22, activation='tanh')(decoder1)
decoder3 = Dense(input_dim, activation='relu')(decoder2)

autoencoder = Model(inputs=input_layer, outputs=decoder3)
```

شکل 4 معماری شبکه **Denoising**

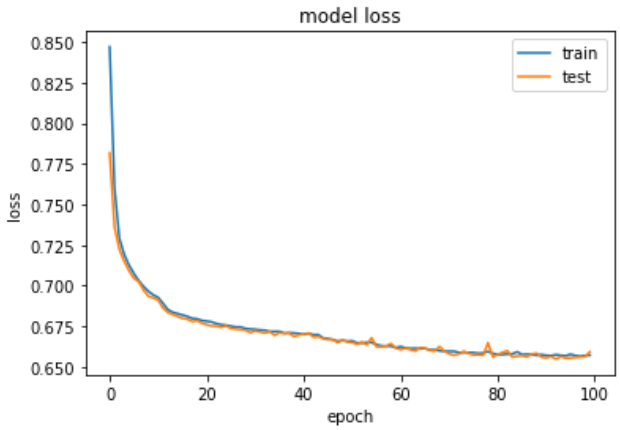
```
autoencoder.compile(metrics=['accuracy'],
                    loss='mean_squared_error',
                    optimizer='adam')

checkpointer = ModelCheckpoint(filepath='autoencoder_fraud.h5',
                               save_best_only=True,
                               verbose=0)

tensorboard = TensorBoard(log_dir='./logs',
                          histogram_freq=0,
                          write_graph=True,
                          write_images=True)

history = autoencoder.fit(X1_train, X1_train,
                          epochs=nb_epoch,
                          batch_size=batch_size,
                          shuffle=True,
                          validation_data=(X1_test, X1_test),
                          verbose=1,
                          callbacks=[checkpointer, tensorboard]).history
```

شکل 5 نحوه آموزش شبکه **Denoising**



شکل 6 نمودار خطا برای داده آموزش و تست

5- نمودار Confusion Matrix و مقادیر F1, Recall, Precision, Accuracy:

بیشترین F1 به ازای Threshold=1 بوده که مقادیر آن به شرح زیر است:

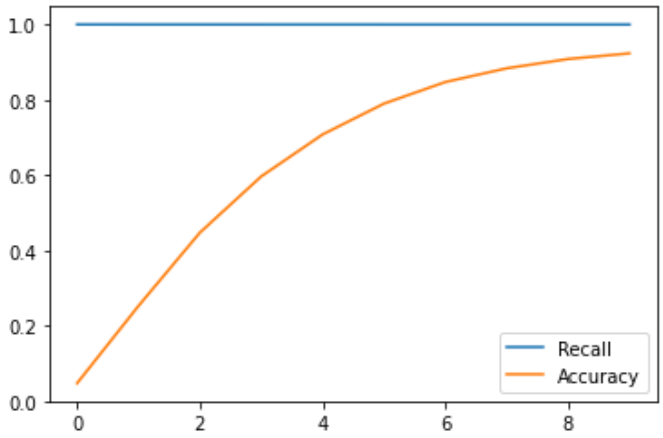
```
Confusion_matrix:
 [[78756  6540]
 [   17   130]]
Accuracy:  92.325878
Recall:    99.978419
Precision: 92.332583
F1:        96.003510
```

در حالت Imbalancement دیتاست، علت عدم اعتکا و استفاده از Accuracy از آنجاییست که در شرایطی که بعنوان مثال ما 95 درصد دیتایی با تارگت 1 و 5 درصد دیتا با تارگت صفر داریم، نوشتن یک خط کد print("1") به 95 درصد accuracy می رسد. در نتیجه در این حالات باید از معیارهای دیگری همانند F1 استفاده گردد.

6- ارزیابی با Thresholdهای متفاوت برای Oversampling و نمودار شکل 7 مقاله:

حال Threshold خروجی را بین صفر تا یک تغییر می‌دهیم. نمودار به صورت زیر خواهد بود:

برای حالت با Sampling و autoencoder نمودار معیارهای Accuracy و Recall به شرح زیر می شود:



شکل 7 نمودار **Accuracy** و **Recall** به ازای **Threshold** های مختلف

به ازای Threshold=1 ماتریس Confusion به شرح زیر است:

```
Confusion_matrix:
 [[85269    27]
 [   27   120]]
Accuracy: 99.936799
Recall:   99.968345
```

7- آموزش داده بدون Resampling و حذف نویز به جهت مقایسه:

نتایج به شرح زیر می شوند.

```
Confusion_matrix:
 [[78756  6540]
 [   17   130]]
Accuracy: 92.32587
Recall:   99.97841
```

مشاهده می شود که با اعمال دو متود مدنظر، شبکه به دقت بالاتری رسیده و جداسازی دو داده بهتر انجام می گیرد.

پاسخ ۲ – Liveness Detection

۲-۱. پیاده سازی

الف)

پاسخ ۳ – تشخیص کاراکتر نوری

۳-۱. پیاده سازی

الف) در شبکه‌های عصبی کانولوشن، ما در لایه‌های اولیه یادگیری ویژگی داریم که در آن ویژگی‌های بسیار ابتدایی آموخته می‌شوند. "Deep" در "DCNN" به تعداد لایه های شبکه اشاره دارد. داشتن 5 تا 10 لایه یا حتی بیشتر از لایه های یادگیری ویژگی در یک CNN عادی معمول است. معماری های مدرن مورد استفاده در کاربردهای پیشرفته دارای شبکه هایی با عمق بیش از 50 تا 100 لایه هستند. عملکرد سی ان ان تقریباً شبیه به کار بیش از حد ساده شده مغز انسان در تشخیص اجزای بینایی از طریق قشر بینایی است. از طرفی با افزایش لایه ها به جهت ساخت DCNN ها، مشکلاتی مانند Overfitting به علت افزایش تعداد پارامترها و یا Gradient Descent به وجود می آید که نیازمند روش هایی مانند بلوک های Residual و یا Batch Normalization و یا Dropout است.

ب) SGD با Momentum همیشه بهتر از الگوریتم SGD معمولی کار می کند. مشکل SGD این است که در حالی که به دلیل نوسان زیاد سعی می کند به حداقل برسد، نمی توانیم نرخ یادگیری را افزایش دهیم. بنابراین برای همگرایی زمان می برد. در این الگوریتم، ما از میانگین وزنی نمایی برای محاسبه گرادیان استفاده می کنیم و از این گرادیان برای به روز رسانی پارامتر استفاده می کنیم.

معادله برای به روز رسانی وزن و سوگیری در SGD:

$$w_t = w_{t-1} - \eta \frac{\partial L}{\partial w_{t-1}}$$

$$b_t = b_{t-1} - \eta \frac{\partial L}{\partial b_{t-1}}$$

شکل 8 معادلات به روز رسانی برای **SGD**

معادله ای برای به روز رسانی وزن ها و بایاس در SGD با Momentum:

$$w_t = w_{t-1} - \eta V_{dw_t}$$

$$\text{where } V_{dw_t} = \beta V_{dw_{t-1}} + (1-\beta)\frac{\partial L}{\partial w_{t-1}}$$

$$b_t = b_{t-1} - \eta V_{db_t}$$

$$\text{where } V_{db_t} = \beta V_{db_{t-1}} + (1-\beta)\frac{\partial L}{\partial b_{t-1}}$$

شکل 9 معادلات به روز رسانی برای **SGD** به همراه **Momentum**

در SGD با Momentum، ما Momentum را در تابع گرادیان اضافه کرده ایم. منظور من از این است که گرادیان فعلی به گرادیان قبلی خود و غیره وابسته است. این باعث تسریع SGD برای همگرایی سریعتر و کاهش نوسان می شود.

Adadelta توسعه ای از Adagrad است که تلاش می کند تا نرخ یادگیری را به شدت کاهش دهد. ایده پشت Adadelta این است که به جای خلاصه کردن تمام گرادیان های مربع گذشته از گام های زمانی 1 تا "t"، چه می شود اگر بتوانیم اندازه پنجره را محدود کنیم. به عنوان مثال، محاسبه گرادیان مجذور 10 گرادیان گذشته و میانگین. این را می توان با استفاده از میانگین وزنی نمایی روی گرادیان به دست آورد.

$$\text{Adagrad} \Rightarrow \quad \eta_t^{'} = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}} \quad \text{where} \quad \alpha_t = \sum_{i=1}^{t} \left(\frac{\partial \mathrm{L}}{\partial \mathrm{w}_{t-1}}\right)^2$$

$$\text{so, as } \text{"t"} \uparrow\uparrow \ , \ \alpha_t \uparrow\uparrow, \ \textit{and} \ \eta_t^{'} \downarrow\downarrow$$

شکل 10 نحوه یادگیری در **Adagrad**

معادله بالا نشان می دهد که با افزایش گام های زمانی "t" مجموع گرادیان های مجذور "α" افزایش می یابد که منجر به کاهش نرخ یادگیری "η" می شود. به منظور حل افزایش نمایی در مجموع گرادیان های مجذور ««α، α»» را با میانگین وزنی نمایی گرادیان های مجذور جایگزین کردیم.

$$\eta_t^{'} = \frac{\eta}{\sqrt{\mathrm{S}_{\mathrm{dw}_t} + \varepsilon}} \quad \text{where} \quad \mathrm{S}_{\mathrm{dw}_t} = \beta S_{\mathrm{dw}_{t-1}} + (1-\beta)\left(\frac{\partial \mathrm{L}}{\partial \mathrm{w}_{t-1}}\right)^2$$

$$\varepsilon \text{ is a small } + \text{ve number to avoid divisibilty by } 0$$

شکل 11 پیاده سازی **Adadelta**

بنابراین، در اینجا برخلاف آلفا "α" در آداگراد، که در آن پس از هر مرحله زمانی به طور تصاعدی افزایش می یابد. در Adadelta، با استفاده از میانگین های وزنی نمایی نسبت به گرادیان گذشته، افزایش Sdw تحت کنترل است. محاسبه "Sdw" مشابه مثالی است که در بخش میانگین وزنی نمایی انجام دادم. مقدار "β" معمولا 0.9 یا 0.95 است.

بهینه ساز Adam یکی از محبوب ترین بهینه سازها است. ایده پشت بهینه ساز Adam استفاده از مفهوم حرکت از "SGD با Momentum" و نرخ یادگیری تطبیقی از "Adadelta" است.

میانگین های وزنی نمایی برای شیب های گذشته:

$$V_{dw_t} = \beta V_{dw_{t-1}} + (1-\beta)\frac{\partial L}{\partial w_{t-1}}$$

$$V_{db_t} = \beta V_{db_{t-1}} + (1-\beta)\frac{\partial L}{\partial b_{t-1}}$$

شکل 12 میانگین وزنی نمایی برای شیب ها

میانگین های وزنی نمایی برای گرادیان های مجذور گذشته:

$$S_{dw_t} = \gamma S_{dw_{t-1}} + (1-\gamma)(\frac{\partial L}{\partial w_{t-1}})^2$$

$$S_{db_t} = \gamma S_{db_{t-1}} + (1-\gamma)(\frac{\partial L}{\partial b_{t-1}})^2$$

شکل 13 میانگین وزنی نمایی برای گرادیان ها

با استفاده از معادله بالا، اکنون فرمول به روز رسانی وزن و سوگیری به نظر می رسد:

$$w_t = w_{t-1} - \frac{\eta}{\sqrt{S_{dw_t} - \varepsilon}} * V_{dw_t}$$

$$b_t = b_{t-1} - \frac{\eta}{\sqrt{S_{db_t} - \varepsilon}} * V_{db_t}$$

شکل 14 معادلات به روز رسانی وزن های و بایاس در **Adam**

مزیت استفاده از بهینه ساز Adam:

- ساده برای پیاده سازی
- کارآمد محاسباتی
- نیازهای کمی حافظه
- مناسب برای مشکلات با گرادیان های بسیار پر سر و صدا/یا کم
- فراپارامترها تفسیر بصری دارند و معمولاً نیاز به تنظیم کمی دارند

ج) در پیاده سازی DCNN ارائه شده در مقاله از چهار لایه کانولوشنی با تعداد Feature Map های مختلف استفاده شده است. همچنین به جهت کاهش Overfitting در هنگام آموزش از لایه Dropout استفاده شده است تا شبکه قادر به حفظ کردن دیتا نشود. از طرفی از Batch Normalization نیز برای پیشرفت یادگیری شبکه استفاده شده است. از لایه های Pooling نیز به جهت کاهش ابعاد فیچرهای خروجی استفاده شده است.

د) نمودارهای Loss و Accuracy و Confusion Matrix و مقادیر Precision، Recall و F1 برای هرکدام از سه روش بهینه سازی:

برای بهینه ساز Adam:

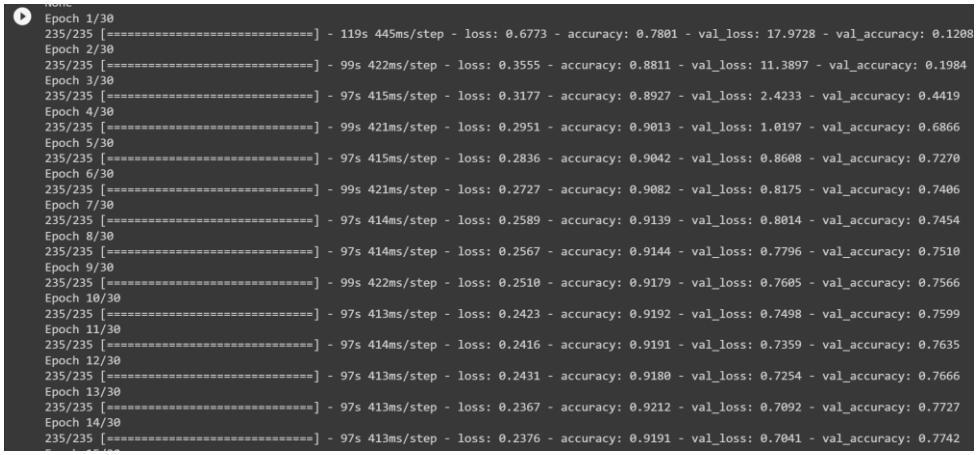
```
Epoch 1/30
235/235 [==============================] - 115s 447ms/step - loss: 0.3140 - accuracy: 0.8992 - val_loss: 16.9577 - val_accuracy: 0.1989
Epoch 2/30
235/235 [==============================] - 97s 411ms/step - loss: 0.1143 - accuracy: 0.9645 - val_loss: 8.1587 - val_accuracy: 0.2237
Epoch 3/30
235/235 [==============================] - 97s 411ms/step - loss: 0.0902 - accuracy: 0.9722 - val_loss: 0.7629 - val_accuracy: 0.7858
Epoch 4/30
235/235 [==============================] - 97s 411ms/step - loss: 0.0832 - accuracy: 0.9750 - val_loss: 0.2286 - val_accuracy: 0.9294
Epoch 5/30
235/235 [==============================] - 97s 411ms/step - loss: 0.0726 - accuracy: 0.9773 - val_loss: 0.2051 - val_accuracy: 0.9367
Epoch 6/30
235/235 [==============================] - 97s 411ms/step - loss: 0.0690 - accuracy: 0.9782 - val_loss: 0.1648 - val_accuracy: 0.9489
Epoch 7/30
235/235 [==============================] - 97s 411ms/step - loss: 0.0668 - accuracy: 0.9799 - val_loss: 0.1629 - val_accuracy: 0.9503
Epoch 8/30
235/235 [==============================] - 97s 411ms/step - loss: 0.0636 - accuracy: 0.9810 - val_loss: 0.1479 - val_accuracy: 0.9543
Epoch 9/30
235/235 [==============================] - 98s 419ms/step - loss: 0.0617 - accuracy: 0.9815 - val_loss: 0.1479 - val_accuracy: 0.9542
Epoch 10/30
235/235 [==============================] - 97s 411ms/step - loss: 0.0609 - accuracy: 0.9814 - val_loss: 0.1441 - val_accuracy: 0.9557
Epoch 11/30
235/235 [==============================] - 97s 411ms/step - loss: 0.0585 - accuracy: 0.9823 - val_loss: 0.1406 - val_accuracy: 0.9571
Epoch 12/30
235/235 [==============================] - 97s 411ms/step - loss: 0.0560 - accuracy: 0.9827 - val_loss: 0.1398 - val_accuracy: 0.9570
Epoch 13/30
235/235 [==============================] - 97s 412ms/step - loss: 0.0541 - accuracy: 0.9841 - val_loss: 0.1335 - val_accuracy: 0.9592
Epoch 14/30
235/235 [==============================] - 97s 412ms/step - loss: 0.0544 - accuracy: 0.9833 - val_loss: 0.1316 - val_accuracy: 0.9601
```

شکل 15 مقادیر خطا در هنگام آموزش برای 30 ایپاک

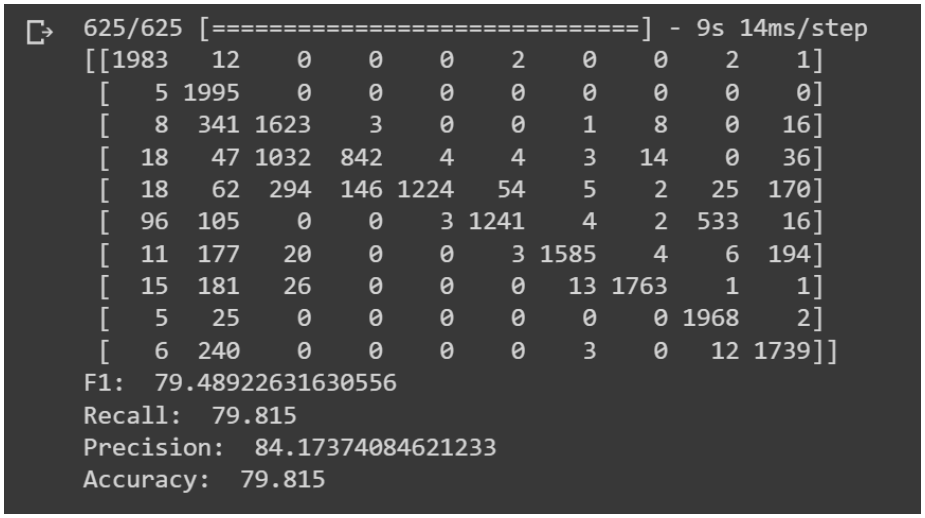
```
625/625 [==============================] - 9s 14ms/step
[[1995    1    0    0    1    1    1    0    0    1]
 [   2 1994    0    0    0    0    1    0    2    1]
 [   0   31 1924    0   26    0   12    0    0    7]
 [   3    1  127 1793   64    1    4    1    0    6]
 [   1    5    2    1 1981    0    7    0    0    3]
 [  25    0    0    0   18 1834    5    0  115    3]
 [   2    4    1    0    3    1 1960    0    1   28]
 [   7    5   16    0    1    0   54 1915    1    1]
 [   0    1    0    0    0    0    0    0 1997    2]
 [   2   10    0    0    0    0    5    0    8 1975]]
F1:  96.82442399199051
Recall:  96.84
Precision:  96.95026456157855
Accuracy:  96.84
```

شکل 16 تمامی نتایج خواسته شده برای بهینه ساز **Adam**

برای بهینه ساز SGD:

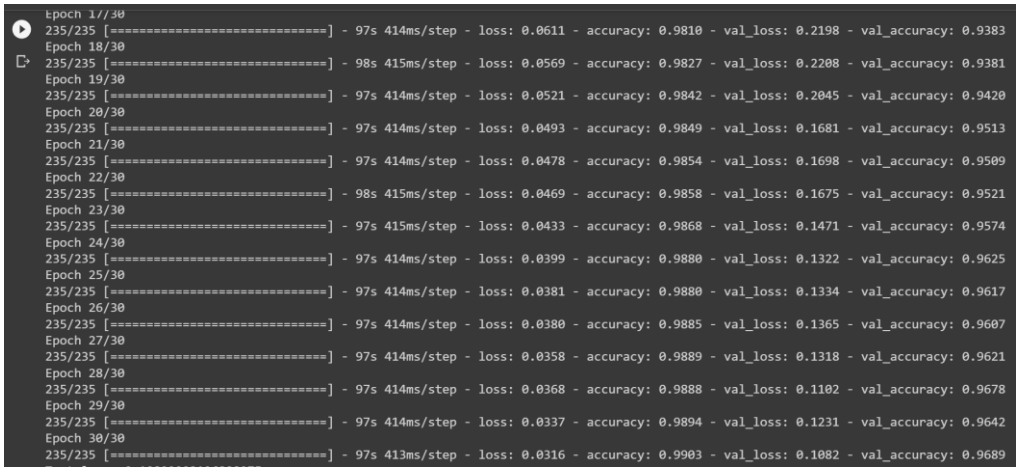


شکل 17 مقادیر خطا در هنگام آموزش برای 30 ایپاک



شکل 18 تمامی نتایج خواسته شده برای بهینه ساز **SGD**

برای بهینه ساز Adadelta:



شکل 19 مقادیر خطا در هنگام آموزش برای 30 ایپاک

```
625/625 [==============================] - 9s 14ms/step
[[1999    0    0    0    0    0    0    1    0    0]
 [   0 1999    0    0    0    0    0    0    1    0]
 [   2   39 1957    0    0    0    0    0    0    2]
 [  13    0  109 1875    3    0    0    0    0    0]
 [   6   10   53   14 1907    0    3    0    0    7]
 [  86   30    2    0    4 1815    2    1   59    1]
 [   8   36    8    0    0    2 1914    4    2   26]
 [   2   16    6    0    0    0    0 1976    0    0]
 [   0   10    0    0    0    0    0    0 1990    0]
 [   8   42    0    0    0    0    1    0    4 1945]]
F1:  96.89120818878446
Recall:  96.885
Precision:  97.05313040464702
Accuracy:  96.885
```

شکل 20 تمامی نتایج خواسته شده برای بهینه ساز **Adadelta**

۵) معماری و پارامترهای بهترین شبکه:

از آنجایی که آموزش شبکه ها باید سه بار برای بهینه ساز های مختلف اجرا می شد، امر آموزش برای صد ایپاک بسیار زمانبر بوده و در نتیجه همگی برای 30 ایپاک آموزش دیده اند. همچنین بهترین لرنینگ ریت و هایپرپارامترهای دیگر در فایل های این بخش قابل مشاهده است.

همانطور که مشاهده می شود، دو بهینه سازی Adam و Adadelta به خوبی به مقدار قابل قبولی می رسند اما از آنجایی که بهینه ساز Adam با سرعت بیشتری به مقادیر بالا میل می کند، بهینه سازی نسبتا بهتر می باشد.