گزارش ۸: مقایسه ی روش های مختلف فراابتکاری برای بهینه سازی ضرایب یک شبکه ی عصبی

نگارش

نیکا شهابی۹۷۱۳۰۲۳

استاد

دکتر مهدی قطعی

خرداد ۱۴۰۰

لینک پروژه در گیتاب:

https://github.com/nikashahabi/deep_learning_with_keras_optimizers

صورت مسئله

روی یک دیتاست لیبل دار (benchmark) یک شبکه ی عصبی train کنید و عملکرد شبکه ی عصبی را با استفاده از optimizer های مختلف بررسی و مقایسه کنید.
در این پروژه شبکه ی عصبی ای با دو لایه ی hidden طراحی شده که در مورد جزیئات آن در ادامه توضیح داده میشود. به علاوه از۴ تا optimizer استفاده شده که درمورد هایپرپارامترهای آنها، نحوه ی عملکرد آن ها و نحوه ی ارزیابی و مقایسه ی آن ها نیز در ادامه توضیح داده میشود.

دیتاست fashion MNIST

توضیح دیتاست: دیتاست استفاده شده از خود tensorflow وبه نام fashion_mnist است. این دیتاست شامل عکس هایی است که هر کدام لیبلی دارند که نشان میدهد عکس مربوطه متعلق به چه دسته ای از ۱۰ دسته ی موجود است. عکس ها ۲۸*۲۸ پیکسل هستند و grayscale میباشند. در این دیتاست، ۶۰۰۰۰ داده ی train set و ۱۰۰۰۰ داده ی test set وجود دارد. Train set را به دو دسته ی train set و validation set تقسیم میکنیم که shape هر کدام از ست ها با استفاده از خروجی کد (تابع loadDatasets) در زیر نشان داده شده است.

X_train_full shape = (60000, 28, 28)
y_train_full shape = (60000,)
X_test shape = (10000, 28, 28)
y_test shape = (10000,)
X_valid shape = (5000, 28, 28)
y_valid shape = (5000,)
X_train shape = (55000, 28, 28)
y_train shape = (55000,)

در شکل زیر ۴۰ داده ی اول train set به علاوه لیبل آن ها plot شده.

۴۰ داده ی اول از train set

به طور کلی هم ۱۰ دسته داریم. یعنی y هر داده از ۰ تا ۹ است که بیانگر کلاس های زیر میباشد:

```
class_names = ["T-shirt/top", "Trouser", "Pullover", "Dress",
"Coat","Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot”]
```

برای توضیحات بیشتر به لینک زیر مراجعه کنید:
/https://keras.io/api/datasets/fashion_mnist

لود کردن دیتاست: برای لود کردن این دیتاست و به دست آوردن train, test and validation set از تکه کد زیر استفاده شده است.
که تابع loadDatasets ، ۳ دسته داده را به ما برمیگرداند، shape آن ها را در فایلی چاپ میکند و چند مثال اولیه از trainset به همراه لیبل های آن ها را plot میکند. (همان شکل بالا)

```
fashion_mnist = keras.datasets.fashion_mnist

sets = loadDatasets(fashion_mnist)
```

دلیل استفاده از train, test and validation set: دلیل استفاده از test set جدا در این است که اطلاعات خود test set وارد مدل ما نشود و ما به وسیله ی آنها مدلمان را train نکنیم و خودمان را گول نزنیم. همچنین دلیل استفاده از validation set این است که هایپرپارامترها و لایه های شبکه عصبی را با توجه به عملکرد مدل روی validation set تغییر دهیم. به عبارتی، از train set برای train کردن مدل و پارامترهای آن استفاده میشود. از validation set برای بررسی اینکه مدلی که با train set، train کردیم با چه کیفیتی کار میکند استفاده میشود که اگر این عملکرد خوب نبود هایپرپارامترها و یا لایه های شبکه عصبی (تعداد و یا activation function آنها) را تغییر دهیم. (Validation set از overfit شدن مدل جلوگیری میکند.) در نهایت از test set فقط برای اینکه مدل نهایی شده را ارزیابی کنیم استفاده میکنیم و با استفاده از آن هایپرپارامترها را تغییر نمیدهیم. (از test and validation set برای train کردن ابدا استفاده نمیشود)
دلیل scale کردن input features: از آنجا که در این مسئله از gradient برای train کردن شبکه عصبی استفاده میشود input features را به بازه ی صفر تا یک scale میکنیم. (با تقسیم آنها بر ۲۵۵.)

```
X_valid, X_train = X_train_full[:5000] / 255., X_train_full[5000:] / 255.

y_valid, y_train = y_train_full[:5000], y_train_full[5000:]

X_test = X_test / 255.
```

معماری شبکه عصبی استفاده شده

با کمک گرفتن از کتاب *Hands-On Machine Learning with Scikit-Learn*, Keras, … شبکه ی عصبی اولیه را با 2hidden layers و با activation function های به ترتیب relu, relu , softmax، و با تعداد نورون های زیرbuild میکنیم. *(این کتاب از همین دیتاست استفاده کرده و شبکه ی عصبی خود را اینطور تعریف کرده. به نظر می‌آید برای شروع استفاده از این شبکه ی عصبی مناسب باشد. جلوتر اگر عملکرد مدل خوب نبود این شبکه را تغییر میدهیم.) (spoiler : عملکردش خوب است.)*

```
# building a classification MLP with two hidden layers
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
```

لازم به ذکر است که flatten کردن برای این است که عکس ها که ۲۸*۲۸ هستند به صورت یک ID array دربیایند. یعنی هیچ پارامتری ندارد و نقش ساده سازی دارد.
لایه ی اول و دوم به ترتیبت۳۰۰ و ۱۰۰ نورون دارند و ازrelu activation function استفاده کرده اند.
لایه ی آخر برای هر کلاس یک نورون دارد. چون ۱۰ لیبل یا کلاس داریم این لایه هم ۱۰ نورون دارد. به علاوه چون کلاس هاexclusive هستند و هر ایتمی دقیقا متعلق به یک کلاس است از softmax activation function برای لایه ی اخر استفاده شده است.
Dense layer پارامترها را به طور دیفالت رندوم و bias ها را صفر initialize میکند.
خروجی model.summary() هم به شکل زیر میباشد:

"Model: "sequential

# Layer (type)	Output Shape	Param
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 300)	235500
dense_1 (Dense)	(None, 100)	30100
dense_2 (Dense)	(None, 10)	1010

Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0

در ادامه میخواهیم این مدل را با استفاده از optimizer های مختلف compile کنیم و سپس مدل را train کنیم و learning curve مربوطه را برای هر optimizer رسم کنیم. در آخر هم عملکرد optimizer ها را مقایسه کنیم.

روال کد، از ساخت شبکه ی عصبی تا classification report

به طور کلی برای۴ optimizer از تابع زیر استفاده میشود.

```
ANN(sets, loss="sparse_categorical_crossentropy", metrics=["accuracy"], optimizer=None, epochs=30)
```

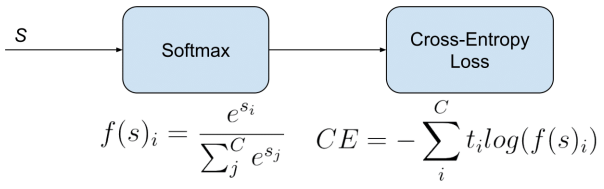
توضیح ورودی های تابع:

- منظور از **sets**، همان test, train and validation set است.
- منظور از **loss** تابعی است که شبکه ی عصبی از آن استفاده میکند تا مدل خود را train کند و مینیمم cost function را به دست بیاورد. (سعی میکند y_hat ها را با معیار loss داده شده تا حد ممکن به y ها نزدیک کند.)
- منظور از **metrics** معیار(ها)یی است که شبکه ی عصبی با استفاده از آن trainخود را میسنجد.(یعنی ببیند با metric داده شده شبکه ی عصبی چطور کار کرده، y_hat ها با این معیار چقدر به yها نزدیک هستند.)

به علاوه در انتهای هر epoch، loss و metrics عملکرد مدل را به روی validation set هم بررسی میکنند.
(استفاده از هر loss ای به عنوان همان metric هم قابل قبول است هر چند این کار بیهوده است.)
دلیل استفاده از sparse_categorical_crossentropy به عنوان loss: ما دو تا کلاس نداریم. پس از binary_crossentropy استفاده نمیکنیم.به علاوه برای هر داده probability حضور در هر کلاس داده نشده است. به عبارتی one-hot vector برای y نداریم. پس از categorical_crossentropy هم استفاده نمیکنیم. در واقع چون کلاس ها exclusive هستند و sparse label داریم، (برای هر داده y ای از ۰ تا ۹) از sparse_categorical_crossentropy استفاده میکنیم.

$$CE = -log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right)$$

فرمول one-hot vector در زمان استفاده از categorical_crossentropy



$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \quad CE = -\sum_i^C t_i log(f(s)_i)$$

فرمول در حالت کلی

دلیل استفاده از accuracy به عنوان metric: به نظر منطقی می‌آید که بخواهیم بررسی کنیم آیا y_hat همان y است یا خیر. یعنی به عبارتی بخواهیم بدانیم آیا مدل لیبل درستی به داده نسبت داده است یا نه. پس از accuracy استفاده میکنیم. فرمول آن هم ساده و قابل پیش بینی است.

- منظور از **epochs**: در پایان یک epoch همه ی داده ها شانس بررسی شدن داشته اند. یعنی یک epoch یک pass در کل دیتاست است. در پایان epoch، loss و accuracy روی validation set هم بررسی میشوند.
- منظور **optimizer**، optimizer مربوطه برای بهینه سازی شبکه ی عصبی و تاثیرگذار در سرعت train کردن مدل ها برای مینیمم کردن cost function و استعداد آنها در بیرون کشیدن مدل از مینیمم محلی میباشد. به عبارت ساده تر، optimizer ها

در سرعت همگرایی واینکه اصلا همگرایی اتفاق می افتد یا نه تاثیر میگذارند.که از optimizer های زیر برای train کردن شبکه ی عصبی توضیح داده شده استفاده شده و هر یک از آنها به جز یک مورد از default تعریف شده ی keras برای hyper parameter ها استفاده کرده اند. همه ی این optimizer ها varient هایی از gradient descent هستند(که در آن- w = w dW) و همان طور که دیده میشود این optimizer ها از dW که dJ/dW است استفاده میکنند. فقط نحوه ی اپدیت کردن پارامترها (W) در آنها تفاوت دارد.

Optimizer	Update rule	Attribute
(Stochastic) Gradient Descent	$W = W - \alpha dW$	• Gradient descent can use parallelization efficiently, but is very slow when the data set is larger the GPU's memory can handle. The parallelization wouldn't be optimal. • Stochastic gradient descent usually converges faster than gradient descent on large datasets, because updates are more frequent. Plus, the stochastic approximation of the gradient is usually precise without using the whole dataset because the data is often redundant. • Of the optimizers profiled here, stochastic gradient descent uses the least memory for a given batch size.
Momentum	$V_{dW} = \beta V_{dW} + (1 - \beta) dW$ $W = W - \alpha V_{dW}$	• Momentum usually speeds up the learning with a very minor implementation change. • Momentum uses more memory for a given batch size than stochastic gradient descent but less than RMSprop and Adam.
RMSprop	$S_{dW} = \beta S_{dW} + (1 - \beta) dW^2$ $W = W - \alpha \frac{dW}{\sqrt{S_{dW}} + \varepsilon}$	• RMSprop's adaptive learning rate usually prevents the learning rate decay from diminishing too slowly or too fast. • RMSprop maintains per-parameter learning rates. • RMSprop uses more memory for a given batch size than stochastic gradient descent and Momentum, but less than Adam.
Adam	$V_{dW} = \beta_1 V_{dW} + (1 - \beta_1) dW$ $S_{dW} = \beta_2 S_{dW} + (1 - \beta_2) dW$ $V corr_{dW} = \frac{V_{dW}}{(1 - \beta_1)^t}$ $S corr_{dW} = \frac{S_{dW}}{(1 - \beta_2)^t}$ $W = W - \alpha \frac{V_{corr_{dW}}}{\sqrt{S_{corr_{dW}}} + \varepsilon}$	• The hyperparameters of Adam (learning rate, exponential decay rates for the moment estimates, etc.) are usually set to predefined values (given in the paper), and do not need to be tuned. • Adam performs a form of learning rate annealing with adaptive step-sizes. • Of the optimizers profiled here, Adam uses the most memory for a given batch size. • Adam is often the default optimizer in machine learning.

مقایسه ی ۴ Optimizer استفاده شده، Notation : dW = dJ/dW

از تکرار مواردی که در جدول آمده به صرف ترجمه ی آنها به فارسی خودداری میکنیم و فرمول های ریاضی را در توضیحات زیر نمی‌اوریم. صرفا به دادن شهودی از این مدل ها بسنده میکنیم.
۱- استفاده از stochastic gradient descent: stochastic یعنی رندوم. در این روش به طور رندوم یک data point استفاده میشود تا به وسیله ی آن مشتق ها گرفته شوند. این ایده از حجم محاسبات در gradient descent میکاهد.

```
tf.keras.optimizers.SGD(
    learning_rate=0.01, momentum=0.0, nesterov=False, name="SGD",
**kwargs
)
```

```
w = w - learning_rate * g
```

۲- استفاده از momentum: یعنی همان sgd + momentum. در sgd ما به گرادیان های قبلی توجه نمیکنیم. اگر slope در لحظه ی مربوطه کم است، حرکت ما هم کند است. ولی در momentum به step های قبلی توجه میشود.

از همان SGD مرحله ی قبل استفاده میکنیم که اپدیت کردن پارامتر چون momentum بزرگ تر از صفر است به حالت زیر درمیاید. Velocity ای که در هر مرحله اپدیت میشود در واقع اطلاعاتی از گرادیان های قبلی در اختیار ما میگذارد.

```
velocity = momentum * velocity - learning_rate * g
w = w + velocity
```

هایپرپارامتر momentum بین صفر و یک است. صفر یعنی اصطکاک ماکسیمم و یک یعنی اصطکاک مینیمم. به طور معمول این هایپرپارامتر ۰.۹ گذشته میشود. در این پروژه ما هم همین کار را کردیم.

۳- استفاده از حالت دیفالت کتابخانه ی keras برای train کردن. که یعنی استفاده از rmsp. rmsp، میانگینی از مربعات گرادیان های قبلی را نگه میدارد و هنگام اپدیت کردن پارامترها، dW را بر ریشه ی این مقدار تقسیم میکند. (اپسیلونی هم در فرمول وجود دارد. برای فهمیدن عملیات ریاضی دقیق به جدول مراجعه کنید.)

```
tf.keras.optimizers.RMSprop(
    learning_rate=0.001,
    rho=0.9,
    momentum=0.0,
    epsilon=1e-07,
    centered=False,
    name="RMSprop",
    **kwargs
)
```

۴- استفاده از Adam: Adam ایده ی rmsp را با momentum ترکیب میکند.مانند momentum میانگینی از گرادیان های قبلی را نگه میدارد و مانندrmsp، past squared gradients را نگه میدارد.

```
tf.keras.optimizers.Adam(
    learning_rate=0.001,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-07,
    amsgrad=False,
    name="Adam",
    **kwargs
)
```

پس در دو حالت اول learning_rate= 0.01 و در دوحالت بعدی برابر 0.001 است. که دیفالت های خود keras هستند.

نحوه ی عملکرد تابع ANN:

۱. در ابتدا شبکه ی عصبی ای با معماری توضیح داده شده را build میکند.

```
# building a classification MLP with two hidden layers

model = keras.models.Sequential([

    keras.layers.Flatten(input_shape=[28, 28]),

    keras.layers.Dense(300, activation="relu"),

    keras.layers.Dense(100, activation="relu"),

    keras.layers.Dense(10, activation="softmax")
```

```
])
```

۲. آن را کامپایل میکند.

```
# compiling the model (if optimizers are used, hyper parameters are their default value)
model.compile(loss=loss, metrics=metrics, optimizer=optimizer if (optimizer) else "rmsprop")
```

۳، train میکند.

```
# training the model
history = model.fit(X_train, y_train, epochs=epochs, validation_data=(X_valid, y_valid))
```

۴. با history به دست آمده از آن learning curve رسم میکند.

```
# plot learning curve
plotLearningCurve(history, optimizerName)
```

۵. با استفاده از مدل train شده پیشگویی برای test set را انجام میدهد.(لازم به ذکر است که در این مرحله احتمال حضور هر یک از example ها در هر کلاس به دست می اید. نه پیش گویی مستقیم یک عکس)

```
# make prediction on test set
y_prediction = model.predict(X_test)
y_prediction_bool = np.argmax(y_prediction, axis=1)
```

۶. با استفاده از مرحله ی قبل classification report را آماده میکند.

```
report.write(classification_report(y_test, y_prediction_bool) + "\n")
```

۷. برای ارتباط بهتر با کاربر هم برای ۴۰ داده ی اول test set پیشگویی مربوطه و لیبل اصلی را نشان میدهد.

```
plotItemsWithLabels(X_test, y_test, model=model, optimizer=optimizerName)
```

که بر خلاف مرحله ی ۶ مستقیم کلاس ها را پیش بینی میکند.

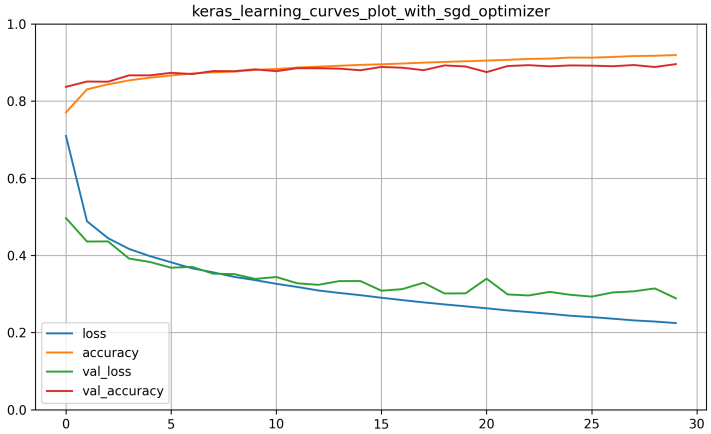
```
y_prediction_classes = model.predict_classes(X)
```

در قسمت تحلیل خروجی درمورد learning curve و classification report توضیح داده میشود.

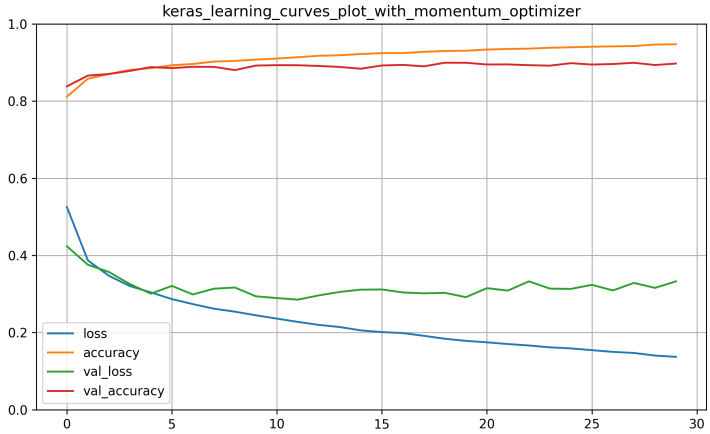
خروجی مسئله و تحلیل آن(مقایسه ی optimizer ها با learning curve و classification report)

خروجی پروژه به صورت فایل های تکست و عکس میباشد و در ترمینال چیز خاصی چاپ نمیکند. در واقع، مراحل ۴، ۶ و ۷ (در توضیح عملکرد تابع ANN)خروجی مشخص از کد به صورت یک فایل تکست یا عکس داشته اند. این فایل ها در figures directory در گیتاب موجود هستند. هر کدام از آنها را می آوریم و درمورد آنها توضیح میدهیم و مقایسه ی optimizer ها را به وسیله ی آنها انجام میدهیم.
خروجی های زیر مربوط به هایپرپارامترهایی است که درمورد آنها توضیح داده شده. به علاوه epochs = 30.

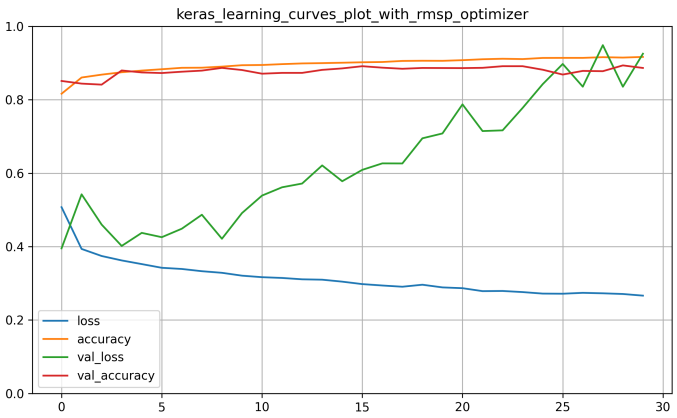
۴. Learning curves : این نمودار ابزار خوبی برای دیدن نحوه ی یادگیری مدل و under fit و یا overfit شدن آن است. اگر loss به train set با گذشت زمان کم نشود یعنی مدل عملکرد خوبی روی یادگیری نداشته و underfitting رخ داده. Underfit یعنی در نهایت هم مدل از پس train set هم برنیاید و پیش بینیش روی آن خوب نباشد.
در حالت overfit واریانس زیاد است. دقت مدل روی داده ی train زیادی خوب است و روی داده ی validation با اختلاف بدتر است. این موضوع در نمودار به این شکل نمایان میشود که loss برای train set کم شود و برای validation set زیاد شود.
حال learning curve به ۴ optimizer توضیح داده شده را می‌آوریم.



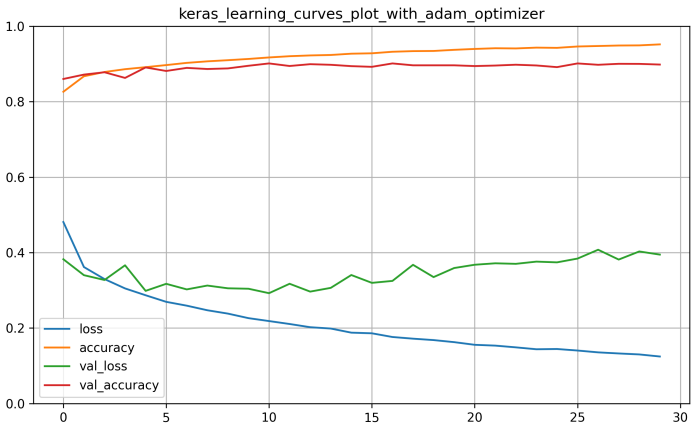
Sgd



Momentum



Rmsp



Adam

نکته ی اول این است که همه ی نمودار ها در طی ۳۰ تا epoch به بهتر یا ثابت ماندن روی داده ی train ادامه داده اند و underfit نشده اند و loss کمی روی train set داشته اند.
نکته ی دوم اینکه rmsp و Adam از جایی به بعد روی validation set عملکردشان رو به نزول رفته و overfit شده اند. این overfit در rmsp به شدت فجیع است و در adam هم اصلا مناسب نیست. به نظر مناسب می آید که epoch را به 10 برای Adam تغییر دهیم تا از overfit جلوگیری کنیم برای rmsp هم عملکرد روی validation set اصلا مناسب نیست و این optimizer در dataset ما خوب کار نکرده است.
نکته ی سوم اینکه به نظر می آید validation set loss بتواند در sgd باز هم کمتر شود. پس train را در این حالت بد نیست ادامه دهیم.
نکته ی چهارم اینکه قوس loss برای validation set از ۱۵ به بعد با train set در momentumفرق میکند. با اینکه validation loss رو به نزول نرفته ولی این هم باز نشانه ی overfitting است. بهتر است train این مدل را هم از ۳۰ به ۱۵ بکاهیم.

۶.Classification report: metric های استفاده شده برای ارزیابی عملکرد مدل ها روی test set به ترتیب precision, recall, f1 score, accuracy هستند.
TP و TN و FP و FN در مسئله ی ما دارای اهمیت نیستند. فقط مهم است که یک عکس درست پیش بینی و دسته بندی شود. پس با توجه به فرمول های زیر منطقی است که از F1score و یا accuracy استفاده کنیم.

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

Different metrics

ردیف macro average و weighted average برای هر metric یکسان است چون تعداد داده های کلاس ها برابر است. ما از accuracy و میانگین f1-score ها برای مقایسه استفاده میکنیم.
عکس اورده شده screenshot از فایل classification_report.txt است.

```
   ------------------------------------------------
classification report with sgd
              precision    recall  f1-score   support

           0       0.81      0.86      0.83      1000
           1       0.99      0.97      0.98      1000
           2       0.80      0.82      0.81      1000
           3       0.88      0.90      0.89      1000
           4       0.83      0.78      0.81      1000
           5       0.97      0.96      0.96      1000
           6       0.71      0.69      0.70      1000
           7       0.92      0.96      0.94      1000
           8       0.96      0.97      0.96      1000
           9       0.97      0.94      0.96      1000

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.88     10000
weighted avg       0.89      0.89      0.88     10000

   ------------------------------------------------
classification report with momentum
              precision    recall  f1-score   support

           0       0.83      0.87      0.85      1000
           1       0.99      0.98      0.99      1000
           2       0.83      0.80      0.82      1000
           3       0.90      0.91      0.90      1000
           4       0.84      0.79      0.81      1000
           5       0.99      0.95      0.97      1000
           6       0.70      0.74      0.72      1000
           7       0.93      0.99      0.96      1000
           8       0.98      0.97      0.97      1000
           9       0.97      0.96      0.96      1000

    accuracy                           0.89     10000
   macro avg       0.90      0.89      0.90     10000
weighted avg       0.90      0.89      0.90     10000

   ------------------------------------------------
classification report with rmsp optimizer
              precision    recall  f1-score   support

           0       0.86      0.76      0.81      1000
           1       0.99      0.96      0.98      1000
           2       0.90      0.64      0.75      1000
           3       0.86      0.91      0.88      1000
           4       0.72      0.90      0.80      1000
           5       0.98      0.95      0.97      1000
           6       0.64      0.73      0.68      1000
           7       0.95      0.95      0.95      1000
           8       0.98      0.95      0.97      1000
           9       0.94      0.97      0.96      1000

    accuracy                           0.87     10000
   macro avg       0.88      0.87      0.87     10000
weighted avg       0.88      0.87      0.87     10000

   ------------------------------------------------
classification report with adam
              precision    recall  f1-score   support

           0       0.87      0.81      0.84      1000
           1       0.98      0.98      0.98      1000
           2       0.77      0.85      0.81      1000
           3       0.87      0.92      0.89      1000
           4       0.82      0.78      0.80      1000
           5       0.98      0.97      0.97      1000
           6       0.72      0.69      0.71      1000
           7       0.93      0.98      0.96      1000
           8       0.98      0.97      0.97      1000
           9       0.97      0.94      0.96      1000

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.89      0.89      0.89     10000
```

Classification report

Accuracy	Average f1 score	
0.89	0.88	**Sgd**
0.89	0.90	**Momentum**
0.87	0.87	**Rmsp**
0.89	0.89	**Adam**

Accuracy : Adam = sgd = momentum > rmsp
Average f1score : momentum > Adam > sgd > rmsp
سه مدل momentum, Adam, sgd به هم بسیار نزدیک هستند. ولی به هر حال نتیجه گیری کلی میکنیم که برای مدل و دیتاست ما: momentum > Adam > sgd > rmsp
هر چند باید در نظر بگیریم که عملکرد بدتر rmsp به این خاطر است که طبق learning curves، rmsp واریانس داشت و overfit شده بود)

۷. پیشگویی ۴۰ داده ی اول test set: با اینکه از ۴۰ داده ی اول هیچ نتیجه گیری کلی و مقایسه ای نمیشود کرد ولی این شکل برای کاربر جذاب و قابل توجه است.

Sgd

Momentum

Adam


Ankle boot p:Ankle boot
Pullover p:Pullover
Trouser p:Trouser
Trouser p:Trouser
Shirt p:Shirt
Trouser p:Trouser
Coat p:Coat
Shirt p:Shirt
Sandal p:Sandal
Sneaker p:Sneaker
Coat p:Coat
Sandal p:Sandal
Sneaker p:Sandal
Dress p:Dress
Coat p:Coat
Trouser p:Trouser
Pullover p:Pullover
Coat p:Coat
Bag p:Bag
T-shirt/top p:T-shirt/top
Pullover p:Pullover
Sandal p:Sandal
Sneaker p:Sneaker
Ankle boot p:Sandal
Trouser p:Trouser
Coat p:Shirt
Shirt p:Shirt
T-shirt/top p:T-shirt/top
Ankle boot p:Ankle boot
Dress p:Coat
Bag p:Bag
Bag p:Bag
Dress p:Dress
Dress p:Dress
Bag p:Bag
T-shirt/top p:T-shirt/top
Sneaker p:Sneaker
Sandal p:Sandal
Sneaker p:Sneaker
Ankle boot p:Ankle boot

Rmsp

پیشنهاد

هایپرپارامترها را برای optimizerها عوض کنیم،
زمان ران شدن را هم به عنوان معیاری برای مقایسه در نظر بگیریم،
معماری شبکه عصبی را عوض کنیم،
همه ی optimizer ها از یک learning_rate یکسان استفاده کنند،
با توجه به overfitting، epoch =10 در نظر بگیریم و دوباره optimizer ها را مقایسه کنیم.

توضیح توابع مسئله

به صورت کامنت در بین کد

منابع

منابعی که از کد آنها استفاده شده:
Hands-On Machine Learning with Scikit-Learn, Keras, … فصل ۱۰ ، ۱۱ گیتاب این کتاب:

https://github.com/ageron/handson-ml2/blob/master/10_neural_nets_with_keras.ipynb

https://github.com/ageron/handson-ml2/blob/master/10_neural_nets_with_keras.ipynb

منابعی که از آنها برای تحلیل، مشخصات دیتاست،… استفاده شده:
https://keras.io/api/optimizers/

https://keras.io/api/metrics/

https://keras.io/api/losses/

https://www.tensorflow.org/datasets/catalog/fashion_mnist

http://en.wikipedia.org/wiki/Cross-validation_%28statistics%29.

https://www.deeplearning.ai/ai-notes/optimization/ جدول اورده شده برای مقایسه ی optimizer ها

https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d