به نام خدا

دانشکده مهندسی کامپیوتر

تمرینات درس یادگیری عمیق

پاسخنامه تمرین سری سوم

دکتر محمدرضا محمدی

دانشجو: مجتبی نافذ

96431335

نیمسال دوم ۰۰-۹۹

سوال یک:

معایب و مزایا دو بهینه ساز RMSProp و AdaGrad را تحقیق کنید و همچنین چه زمانی استفاده از momentum مفید است؟

مزایای AdaGrad:

۱- این بهینه ساز step size ما را در شیب های بالا کاهش می دهد که ما نقطه بهینه را رد نکنیم و step size ما را در شیب های کم افزایش می دهد که اگر منحنی نسبتا یکنواخت بود با گام ها بلند حرکت کنیم تا به نقطه ی بهینه برسیم

۲- هر پارامتر بر اساس گذشته خودش کاهش میابد

معایب AdaGrad:

۱-به دلیل افزایش مداوم مخرج (مجموع توان دو ها گرادیان) نرخ یادگیری همواره در حال کاهش است و رفته رفته step size ما کمتر میشود. که این در کل عیب این بهینه ساز میباشد اما خیلی هم بد نیست چرا که ما هم علاقه داریم که نرخ یادگیری را رفته رفته کاهش دهیم ولی عیب این بهینه ساز این است که این کاهش نرخ دست ما نیست و کنترلی نسبت به آن نداریم ما علاقه داریم هر زمان اراده کردیم نرخ را تغییر دهیم و کم کنیم

۲-نرخ یادگیری در داده های آموزشی هم رفته رفته کم تر میشود.

مزایای RMSProp :

۱- این بهینه ساز step size ما را در شیب های بالا کاهش می دهد که ما نقطه بهینه را رد نکنیم و step size ما را در شیب های کم افزایش می دهد که اگر منحنی نسبتا یکنواخت بود با گام ها بلند حرکت کنیم تا به نقطه ی بهینه برسیم

۲- مشکل AdaGrad که نرخ یادگیری کاهش پیدا می کرد را حل کرد

۳- هر پارامتر بر اساس گذشته خودش کاهش میابد

معایب RMSProp :

-۱

زمان مفید استفاده از momentum : زمانی که تابع خطای ما مینیمم محلی های زیادی داشته باشد.

سوال دو:

همانطور که میدانید callbacks ها به ما امکاناتی میدهند تا روند آموزش داده ها را با جزییات بیشتر دنبال کنیم حال درباره ی modelcheckpoint , earlystopping و tensorboard تحقیق کرده و پارامتر های هر کدام را به طور مختصر توضیح دهید.

Earlystopping: وقتی متریک کنترل شده پیشرفت را متوقف کرد و مقدار ثابتی به خود گرفت آموزش را متوقف کنید.

```
tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0,
    patience=0,
    verbose=0,
    mode="auto",
    baseline=None,
    restore_best_weights=False,
)
```

پارامتر ها:

Monitor: متریک ای که باید چک کنیم تا در صورت عدم پیشرفت آموزش را متوقف کنیم

Min_delta: تعریف عدم پیشرفت متریک از نظر ما =< مثلا اگر **lost** تغییرش کم تر از **0.00001** بود از نظر ما پیشرفت نکرده و آموزش متوقف بشه.

Patience: اگر به تعداد **patience** تا **epochs** متریک مورد نظر ما هیچ پیشرفتی نداشت بعد از آن آموزش متوقف می شود.

Verbose: حالت زبانی **verbosity mode**

Mode: یکی از `{"auto", "min", "max"}` در حالت **min** آموزش زمانی متوقف میشود که کاهش متریک ما به متوقف شود و در حالت **max** آموزش زمانی به پایان میرسد که افزایش متریک ما متوقف شود و در حالت **auto** جهت به صورت خودکار از نام و متریک مانیتور شده استنتاج می شود.

Baseline: این **baseline** مقدار پایه برای متریک مانیتور شده است. اگر مدل نسبت به مقدار پایه بهبود نیابد ، آموزش متوقف می شود.

Restore_best_weights : بهترین وزن ها رو دخیره کن و خروجی بده نه آخرین **epoch** رو.

Modelcheckpoint: ذخیره ی مدل کراس یا وزن های مدل کراس با یک فرکانس مشخص. که بعد ها می توان دوباره این وزن ها را **load** کرد و ادامه آموزش را از همین جا ادامه داد.

```
tf.keras.callbacks.ModelCheckpoint(
    filepath,
    monitor="val_loss",
    verbose=0,
    save_best_only=False,
    save_weights_only=False,
    mode="auto",
    save_freq="epoch",
    options=None,
    **kwargs
)
```

Filepath: مسیر مدل کراس ذخیره شده.

Monitor: متریک ای که باید مانیتور شود.

(.Prefix the name with "val_" to monitor validation metrics .Use "loss" or "val_loss" to monitor the model's total loss)

verbose:حالت زبانی **verbosity mode** که ۰ یا ۱ است.

Save_best_only: اگر مقدار **save_best_only** مثبت بوده و **True** باشد بهترین مدل را ذخیره خواهیم کرد. و وقتی یکی را ذخیره کردیم تا زمانی که از لحاظ متریک مانیتور شده یک مدل بهتر از قبلی را پیدا نکردیم آن را **overwrite** نکردیم.

Mode : یکی از `{"auto", "min", "max"}` اگر **save_best_only** مقدار **True** بگیرد. تصمیم اینکه فایل ذخیره شده ی کنونی را بر اساس مینیمم گیری یا ماکزیمم گیری بهینه کنیم و **overwrite** کنیم را مشخص میکند.

Save_weights_only: وزن های مدل را هم ذخیره کنیم یا خیر؟

Save_freq: دو حالت دارد : اول اینکه مقدار آن **'epoch'** باشد که در این صورت هر اپیک ذخیره سازی صورت میگیرد. مقدار بعدی و دومی که میتواند بگیرد یک **integer** است که مفهوم آن این است که هر چند **batch** یکبار ذخیره سازی صورت بگیرد.

Options: Optional tf.train.CheckpointOptions object if save_weights_only is true or optional tf.saved_model.SaveOptions object if save_weights_only is false.

****kwargs**: آرگومان های اضافی برای سازگاری

TensorBoard: فعالسازی visualizations و تجسم

- **Metrics summary plots**
- **Training graph visualization**
- **Activation histograms**
- **Sampled profiling**

```
tf.keras.callbacks.TensorBoard(
    log_dir="logs",
    histogram_freq=0,
    write_graph=True,
    write_images=False,
    update_freq="epoch",
    profile_batch=2,
    embeddings_freq=0,
    embeddings_metadata=None,
    **kwargs
)
```

Log_dir: مسیر دایرکتوری که در آن می توان فایلهای log را ذخیره کرد که توسط TensorBoard تجزیه می شود.

Histogram_freq: فرکانسی (in epochs) که با آن هیستوگرام های activation و weight را برای لایه های مدل محاسبه میکند. اگر روی 0 تنظیم شود ، هیستوگرام محاسبه نمی شود.

Write_graph: اینکه آیا نمودار را در TensorBoard تجسم کنید یا نه. وقتی log_graph روی True تنظیم می شود ، پرونده log خیلی حجیم و بزرگ میشود.

write _images: اینکه آیا وزن های مدل را برای visualize به عنوان تصویر در TensorBoard باید نوشت یا خیر.

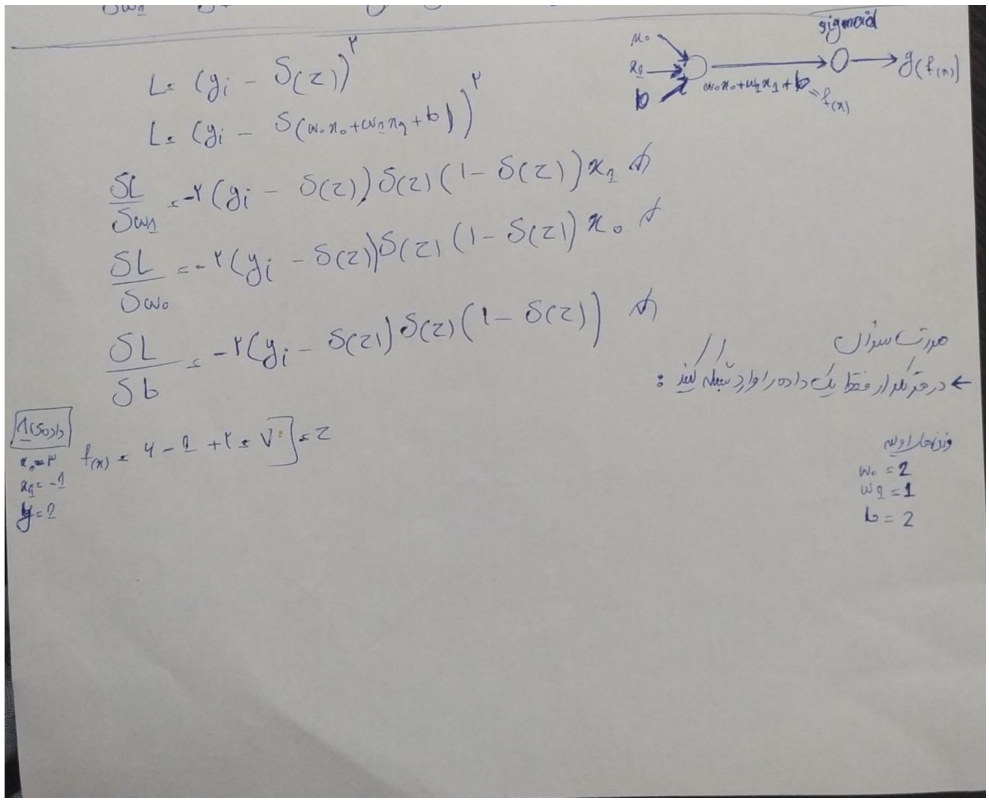
Update_freq: سه مقدار میتواند بگیرد: 'batch' یا 'epoch' یا integer . که اگر batch باشد بعد هر batch ما خطا و متریک ها را در tensorboard مینویسیم.اگر epoch باشد بعد هر epoch و اگر integer باشد مثلا برابر ۵۰، هر ۵۰ ، batch یکبار.

Profile_batch: برای profile کردن batch ها استفاده مشود میتوان یک عدد باشد که یعنی batch فلان را با ویژگی هایش به من نشان بده و میتواند یک رنج باشد که یعنی از batch مثلا n ام تا n+m ام را تحلیلش را نمایش بده.

Embeddings_freq:فرکانسی (در epoch ها) که در آن لایه های embed شده تجسم می یابد. اگر روی 0 تنظیم شود ، لایه های Embed شده قابل مشاهده نیستند.

Embeddings_metadata: یک dictionary که در آن نام لایه را به نام فایلی که در آن لایه های embed شده را ذخیره می کند map میکند

سوال ۳:



حل عددی را با کد بدون کراس در فولدر های اصلی قرار داده ام.

سوال ۴:

```
adam accuracy = 93.84 %
Adagrad accuracy = 52.769999999999996 %
accuracy RMSprop= 93.49 %
```

همانطور که میدانیم بهینه ساز adam که ترکیبی از momentum و RMSProp است بیشترین دقت را پیدا کرده.

و همچنین adagrad که کم کم نرخ یادگیری اش کاهش میابد از دقت پایینی برخوردار است. و rmsprop هم دقت خوبی به نسبت دارد.

دقت کنید چون سرعت همگرایی RMSProp بیشتر از adam است و سریع تر همگرا میشود و همچنین در تابع های خطای ساده تر که شیب کم تری دارند بهتر و سریع تر از Adam همگرا میشود گاها دقت بیشتری از adam دارد.