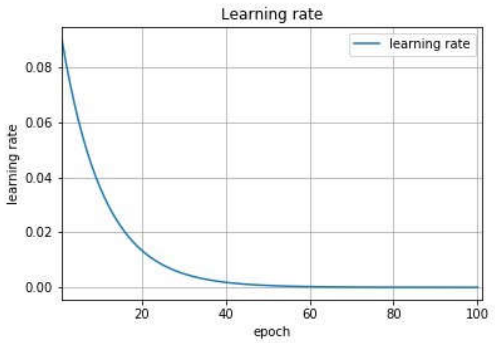
پاسخ سوال سوم:

قسمت اول:

پیاده سازی آن به صورت زیر است

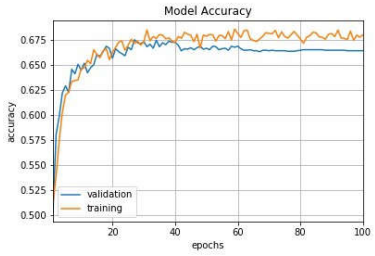
```
def exp_decay(t):
   initial_lrate = 0.1
   k = 0.1
   lrate = initial_lrate * exp(-k*t)
   return lrate

lrate = LearningRateScheduler(exp_decay)
```

که متغیر t هر ایتریشن را نشان میدهد.



قسمت دوم:

میتوانیم در قسمت بهینه سازی کدمان از یک نرخ یادگیری ثابت استفاده کنیم ولی برای همگرایی بهتر، می‌توانیم آن را کاهش دهیم تا مطمین باشیم همگرایی گلوبال خوبی در عوض خراب کردن همگرایی لوکال دارد. روش های مختلفی برای این کاهش وجود دارد که میتوان به Exponential Decay و Step Decay و Time-based Decay اشاره کرد که تفاوتشان در نحوه کاهش این نرخ می باشد.



قسمت سوم:

در کد بالا Decay steps همان k که نرخ کاهش را کنترل میکند. مشخص است هرچه کمتر باشد این کاهش ملایم تر خواهد بود. و initial_lrate همان Decay rate گفته شده در سوال است که مقدار اولیه را نشان میدهد. بهتر است در ابتدا مقادیر نسبتا بزرگتری نسبت به حالتی که قرار بود ثابت داده شود گذاشته شود تا به مرور خودش کاهش پیدا کند و سرعت همگرایی در ابتدا سریع تر شود.