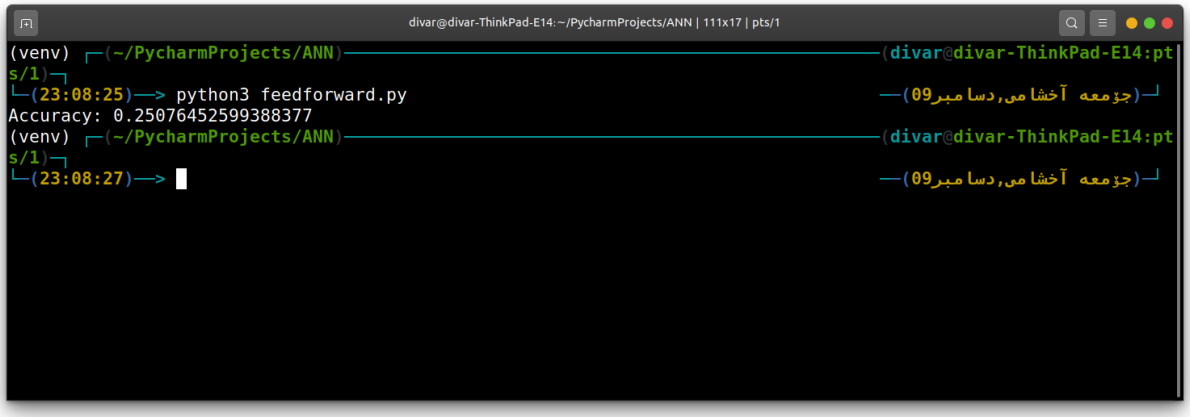
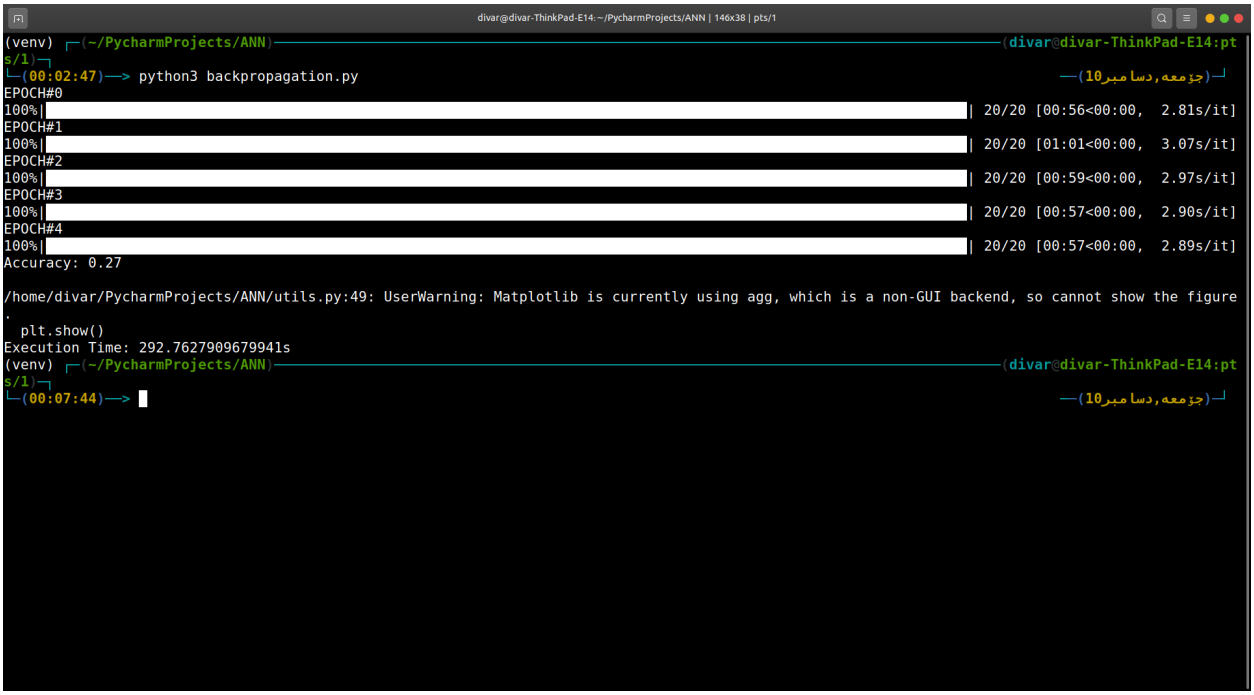
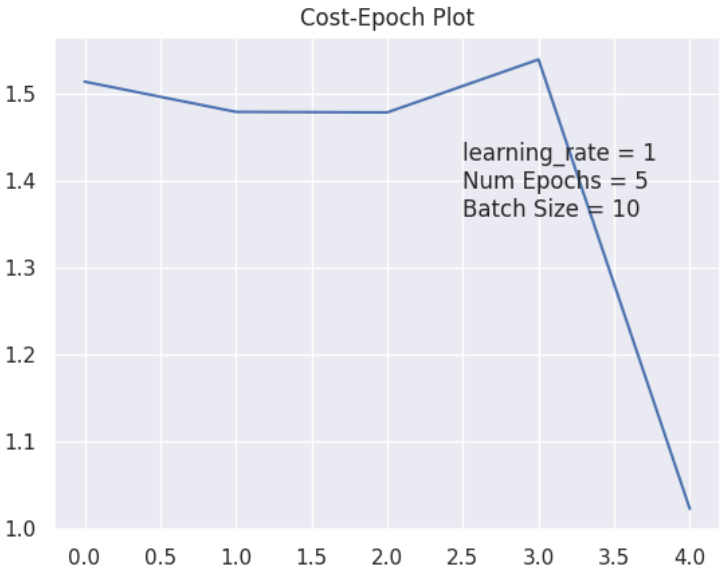
۱. گام دوم، Feedforward



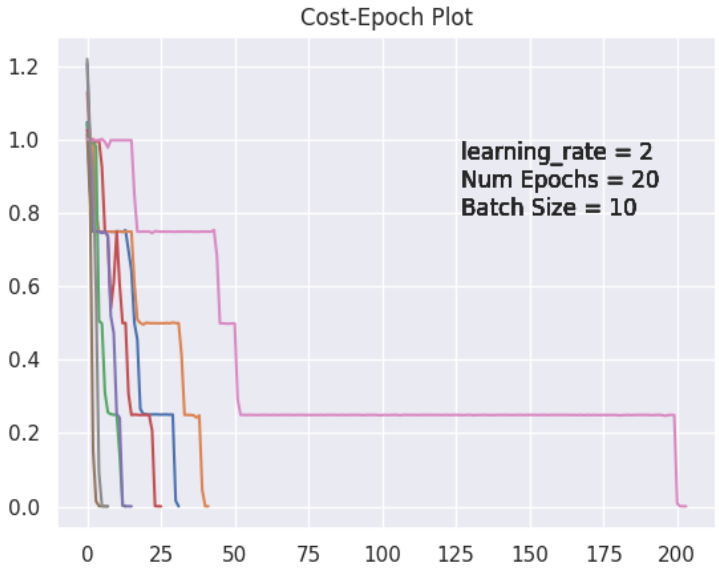
۲. گام سوم، پیاده‌سازی Backpropagation



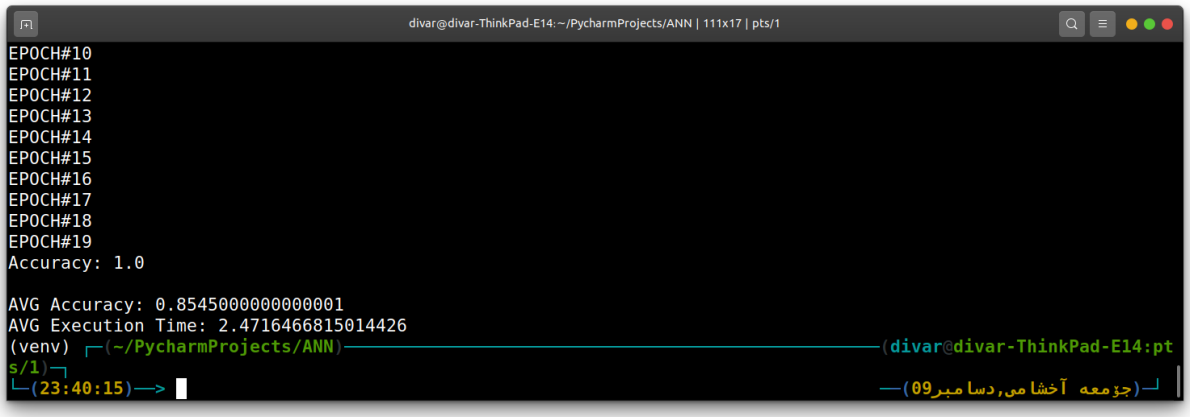


3. گام چهارم، پیاده‌سازی Vectorization

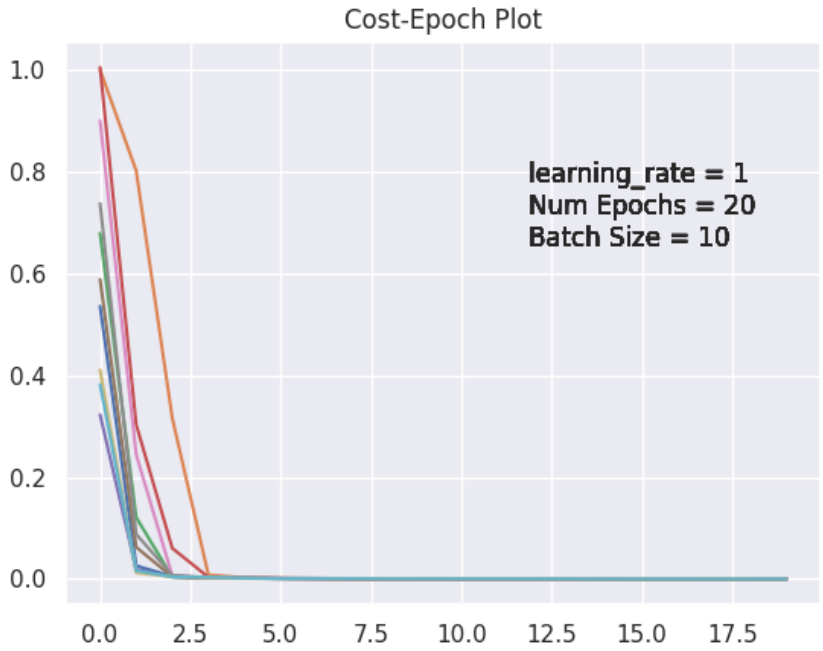
نمودار هزینه‌ها:

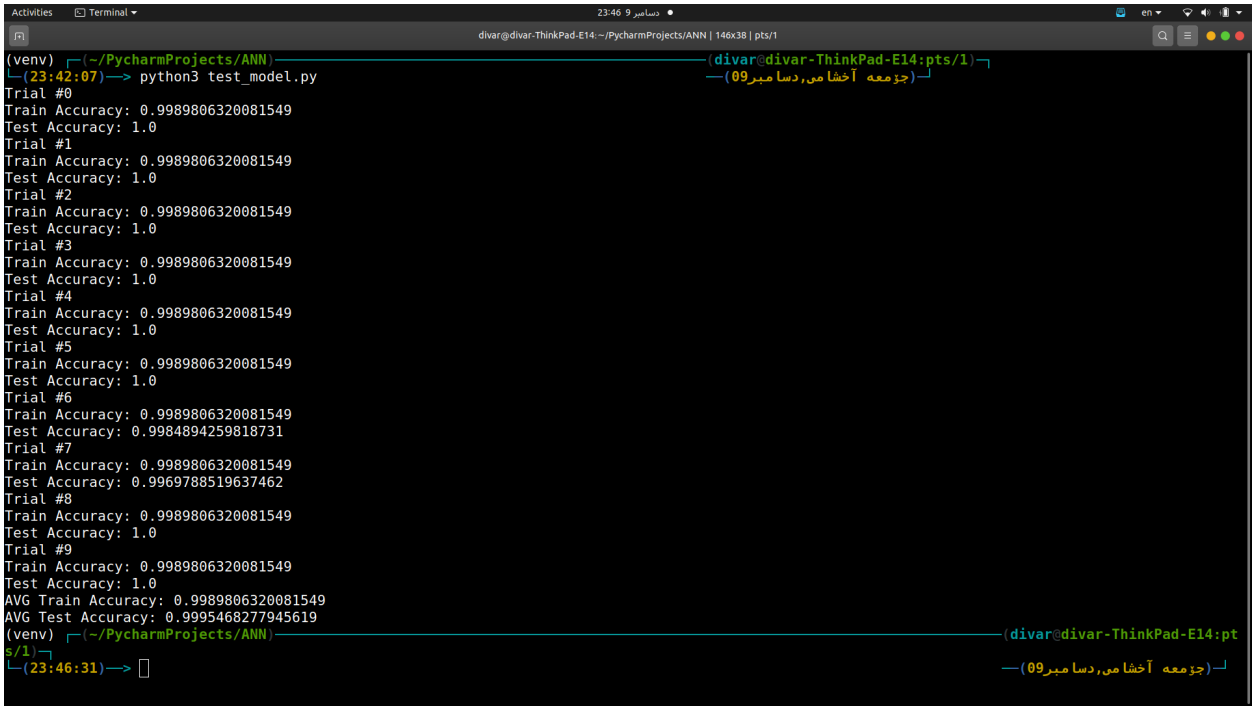


میانگین دقت و میانگین زمان اجرای الگوریتم، طی ۱۰ بار اجرا:



۴. گام پنجم، تست مدل





امتیازی‌ها:

۱. انتخاب یک ضریب یادگیری مناسب

برای این کار، مقادیر مختلف ضریب یادگیری را بر حسب تعداد Epochهای مورد نیاز برای همگرایی مقایسه کرده، و بهترین را انتخاب می‌کنیم. تعریفمان از همگرایی، زمانی است که میزان cost کم‌تر از یک میزان مشخص (در این جا 0.01) شود.

روند بالا در کد ماژول find_lr پیاده شده است. در این ماژول، به ازای هر ضریب یادگیری، میانگین تعداد ایپاک‌های مورد نیاز برای رسیدن به همگرایی (تعریف شده به عنوان convergence_cost) گزارش می‌شود.

تغییر تعداد Epochها:

Epochs	AVG Accuracy on Test Set
1	0.61
5	0.84
10	0.97

Learning Rate = 1, Batch Size = 10

تغییر اندازه‌ی Batch:

Batch Size	AVG Accuracy on Test Set
1	0.9734138972809667
5	0.924773413897281
10	0.8749244712990937

Epochs = 10, Learning Rate = 1

تغییر Learning Rate:

Learning Rate	AVG Accuracy on Test Set
0.5	0.9728096676737159
1	0.9235649546827794
2	0.35861027190332323

Epochs = 10, Batch Size = 10

۲. روش‌های پیشرفته‌تر در SGD

فارغ از این که برای ضریب یادگیری چه مقداری تعیین کنیم، در مرحله‌ای از اجرای الگوریتم آن مقدار ممکن است بیش از حد بزرگ یا کوچک باشد. در چنین شرایطی نیاز به یک ضریب یادگیری متغیر خواهیم داشت، که برای آن چندین الگوریتم مانند Adam, RMSProp, Momentum پیشنهاد شده‌اند.

روش Momentum در پروژه پیاده شده است. این روش به این صورت عمل می‌کند که در هر مرحله، علاوه بر حاصل‌ضرب گرادیان در ضریب یادگیری، یک جمله‌ی جدید هم روی تغییرات وزن متأثر است. این جمله، حاصل‌ضرب یک ضریب جدید (ضریب Momentum) در گرادیان میزان تغییر وزن در پیمایش قبلی‌ست.

Backpropagation Training

$$w_{i,j}(t+1) = w_{i,j}(t) + \eta \Delta w_{i,j} + \alpha \Delta w_{i,j}(t)$$

- where:
  - delta is the weight change
  - eta is the learning rate
  - alpha is the momentum term

به این ترتیب، نحوه‌ی به‌روزرسانی وزن‌ها به شکل زیر می‌شود.

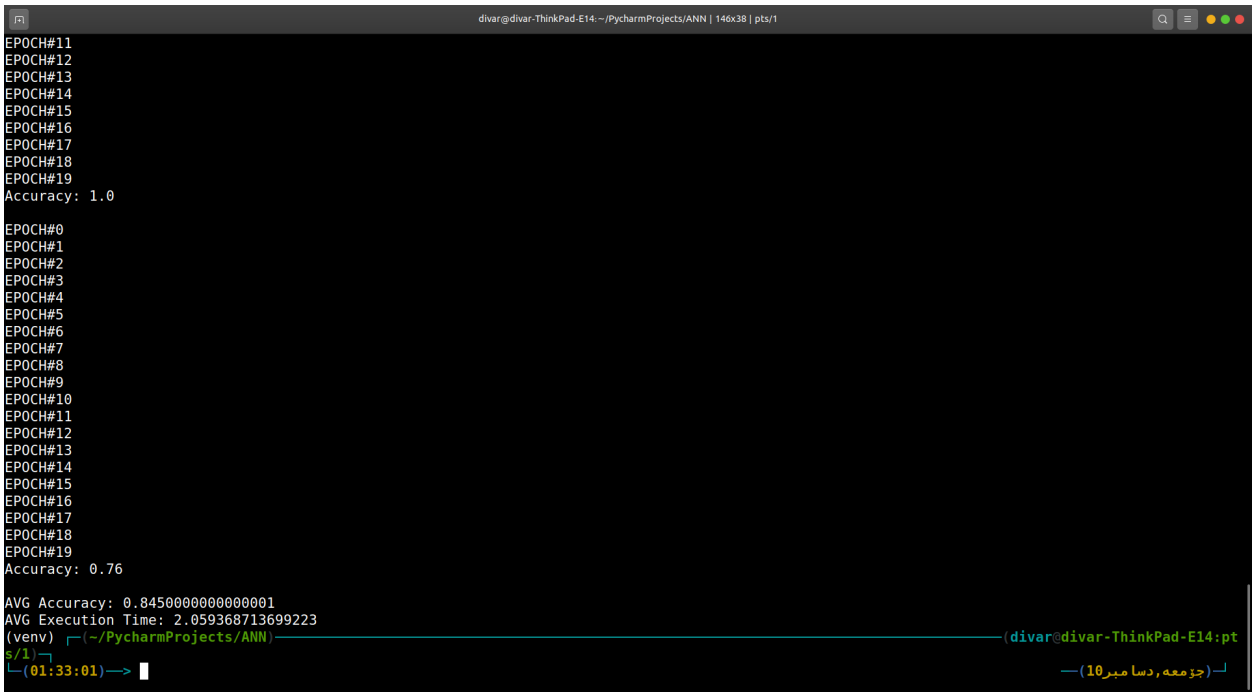
```
w1_velocity = learning_rate * (d_w1 / batch_size) + momentum * w1_velocity
w1 -= w1_velocity

w2_velocity = learning_rate * (d_w2 / batch_size) + momentum * w1_velocity
w2 -= w2_velocity

w3_velocity = learning_rate * (d_w3 / batch_size) + momentum * w3_velocity
w3 -= w3_velocity

b1 -= learning_rate * (d_b1 / batch_size)
b2 -= learning_rate * (d_b2 / batch_size)
b3 -= learning_rate * (d_b3 / batch_size)
```

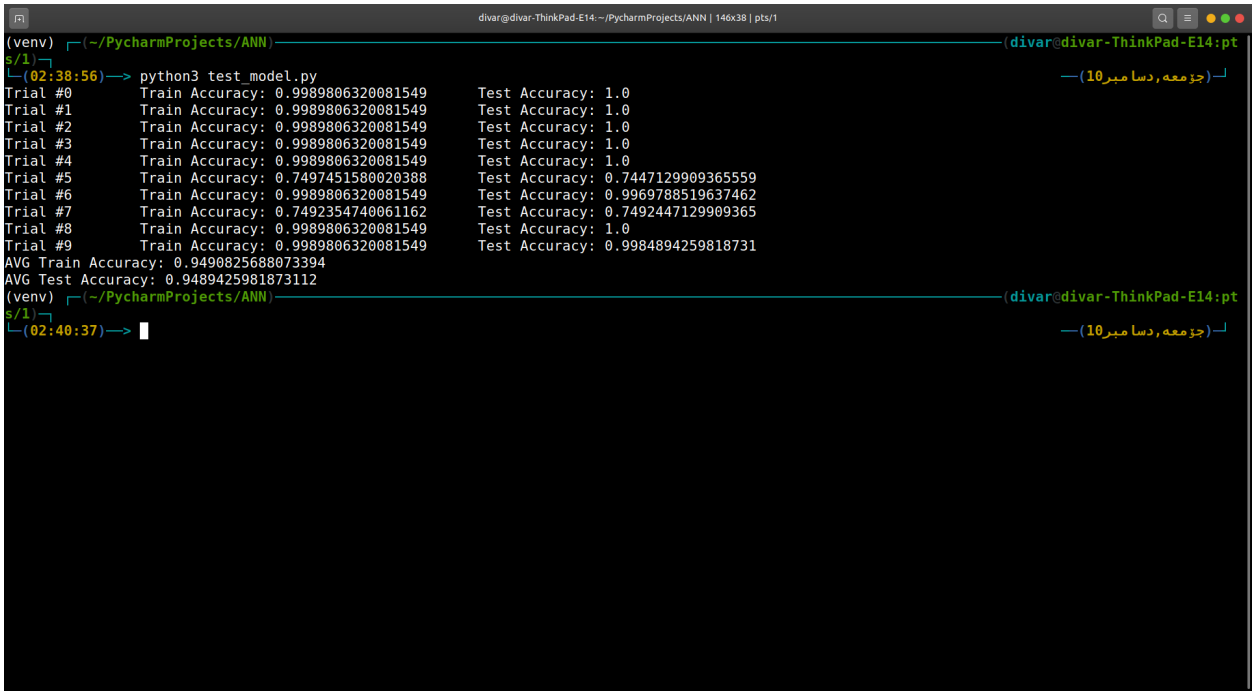
با اجرای کد فایل mometum به نتایج زیر می‌رسیم.



۳. تست برنامه برای کلاس‌های بیش‌تر

دو کلاس Cherry 1 و Pineapple به کلاس‌های دیتاست اضافه شده‌اند. برای این کار، نیاز به تغییر فایل‌های Load_Dataset.py و Feature_Extracion_*.py بود. فایل‌های جدید با ابعاد به‌روز شده ضمیمه شده‌اند.

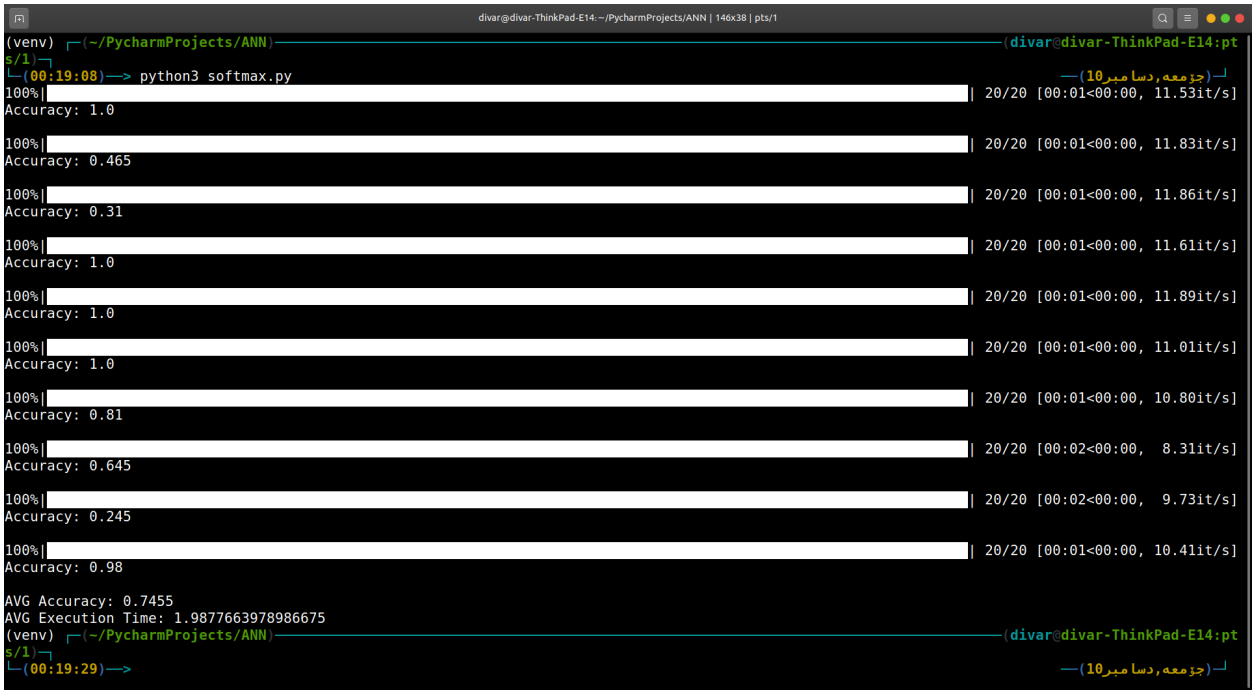
نتیجه‌ی خروجی برنامه برای این حالت جدید به شکل زیر است.



۴. استفاده از Softmax

تابع Softmax در فایل utils پیاده شده، و از آن به عنوان تابع فعال‌سازی لایه‌ی آخر (محاسبه‌ی a3) در ماژول softmax استفاده شده است.

با اجرای برنامه در چنین حالتی خواهیم داشت:



همان‌طور که مشاهده می‌شود میزان دقت به 0.74 رسیده و میانگین زمان اجرای برنامه نیز به 1.98 ثانیه رسیده است.