بنام خدا

گزارش تمرین سری سوم درس یارگیری عمیق

سارا بارونی
۹۸۴۴۳۰۳۱

الگوریتم Adagrad

این الگوریتم مقدار learning rate ثابتی که در حالت معمول داشتیم را تقسیم میکند بر جذر مجموع توان دو گرادیان‌ها به طوری که هیستوری را درنظر میگیرد و فلسفه‌ی این کار این است که آداپته شود با تغییرات شیب به گونه‌ای که اگر شیب کم باشد چون گرادیان‌ها در مخرج قرار دارند درنتیجه حاصل learning rate افزایش خواهد یافت و این کمک میکند در قسمت‌هایی که تابع هزینه flat است بتواند سریع تر پیش برود و این مشکل برطرف بشود ضمن اینکه اگر شیب زیاد باشد مقدار learning rate کم میشود و با گام‌های کوچک‌تری پیش میرود و این کمک میکند تا اگر نزدیک نقطه‌ی اپتیمم باشد از آن عبور نکند و محتاط‌تر پیش برود. این الگوریتم در تابع هزینه ای که convex است خوب عمل میکند. اما ایرادی که میتوان از این روش گرفت درواقع آن است که کل هیستوری را درنظر میگیرد مثلا یکی از مشکلاتی که میتواند ایجاد شود آن است که شیب همچنان زیاد شود و این باعث میشود learning rate کم و کمتر شود و گاهی ممکن است آنقدر کم بشود که اثری در آپدیت شدن وزن‌ها نداشته باشد.

در شکل ۱ نیز شمای کلی از این الگوریتم مشاهده میشود.

در پیاده سازی الگوریتم مشاهده میشود که نمودار دقت داده‌ها صعودی است و صعودی بودن این نمودار در test به معنای آن است که overfit نشده است.

لازم به ذکر است درتمامی شکل‌ها نمودار قرمز مربوط به دقت داده‌های test و آبی دقت داده‌های train و سبز دقت داده‌های validation را نشان می‌دهد.

پارامترهای این الگوریتم شامل epsilon و delta می‌باشد که معمولا epsilon را 0.001 درنظر میگیرند و delta را $1e\text{-}8$ درنظر میگیرند. این پارامترها در شکل ۱ که این الگوریتم را نشان میدهد مشاهده میشوند که به چه شکل در این الگوریتم به کاربرده شده اند.

الگوریتم Adadelta

این الگوریتم مشابه الگوریتم Adagrad می‌باشد اما به نوعی اثر هیستوری های دورتر را کم میکند. الگوریتم درقالب فرمول‌های زیر بیان شده است.

$$W(t+1) = W(t) - \frac{\sqrt(D(t-1) + epsilon)}{\sqrt(v(t) + epsilon)}\frac{\partial L}{\partial W(t)} \tag{۱}$$

$$D(t) = beta * D(t-1) + (1-beta)deltaw(t) \tag{۲}$$

$$v(t) = beta * v(t-1) + (1-beta)(\frac{\partial L}{\partial W(t)})^2 \tag{۳}$$

$$deltaw(t) = W(t) - W(t-1) \tag{۴}$$

در این الگوریتم از پارامترهای beta که معمولا 0.95 آن را درنظر میگیرند و epsilon که معمولا آن را $1e\text{-}8$ درنظر میگیرند استفاده می‌شود.

در شکل ۱۲ خروجی مدل با این الگوریتم و همچنین الگوریتم adagrad باهم رسم شده است که همانطور که مشاهده میشود به طور کلی در الگوریتم adadelta افزایش یافته اما مشکلی که دارد از قسمتی به بعد نزولی

شده است. **الگوریتم RMSprob**

این الگوریتم همان الگوریتم Adagrad است با این تفاوت که بجای مجموع توان دو گرادین‌ها میانگین وزن‌دار نمایی آن‌ها را حساب میکند. این الگوریتم زمانی که تابع هزینه nonconvex باشد میتواند بهتر عمل کند و در مواردی که تابع هزینه convex باشد هم سریع تر همگرا میشود باتوجه به اینکه مشکلی که در الگوریتم Adagrad وجود داشت (هیستوری را درنظر میگرفت و ممکن بود قبل از رسیدن به نقطه‌ی اپتیمم وزن‌ها آپدیت نشوند) را حل کرده است به این شکل که اثر هیستوری گرادیان‌ها را هرچه دورتر باشد کم میکند.

پارامترهایی که در این الگوریتم وجود دارد شامل beta و epsilon میباشند که به طور معمول beta را 0.95 و epsilon را 1e-8 درنظر میگیرند.

الگوریتم Adam

روش Adaptive momentom روش RMSprob را با روش momentom ترکیب میکند درواقع momentom کمک میکند نوسانات موجود در جواب را کم کنیم و RMSprob کمک میکند تا نرخ یادگیری را تنظیم کنیم

این الگوریتم به طور کامل در شکل ۲ مشاهده میشود. که پارامترهایی که به کاربرده شده اند در آن شامل beta و delta و epsilon می‌باشند که معمولا beta را 0.999 و 1e-7 و epsilon را 0.001 درنظر میگیرند.

الگوریتم Gradient descent

در این روش وزن‌ها با مقدار یک learning rate ثابت آپدیت میشوند.

الگوریتم L۲ Regularization

در رگولاریزیشن یک بخش جدید یعنی تابعی از بردار وزن‌ها را به تابع هزینه اضافه میکنیم

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} L(y(x,\theta)) + R(\theta) \qquad (۵)$$

که این بخشی که اضافه میکنیم در L2 regularization به این شکل تعریف میشود

$$R_L two(\theta) = \frac{\lambda}{2}\sum_{i=1}^{N} W_i^2 \qquad (۶)$$

به طور معمول از L2 regularization بیشتر از L1 regularization استفاده می‌شود و L2 regularization اثربخشی بیشتری در شبکه‌های عصبی دارد اثر این روش این است که وزن‌ها نسبت به قبل مقادیر کوچکتری داشته باشند و این جلو overfit شدن را میگیرد و مدل ساده‌تری خواهیم داشت. در ادامه روابط ریاضی آپدیت شدن وزن‌ها در این روش را داریم:

$$W = W - \alpha(\frac{\partial J}{\partial W} + \lambda\frac{\partial R}{\partial W}) \qquad (۷)$$

$$W = W - \alpha(\frac{\partial J}{\partial W} + \lambda W) \qquad (۸)$$

$$W = (1-\alpha\lambda)W - \alpha\frac{\partial J}{\partial W} \qquad (۹)$$

در پیاده‌سازی این الگوریتم اینگونه عمل شد که به تابع function lrcost مجموع توان دو وزن‌ها را اضافه کردم و برای مشتق گرفتن از آن اثر آن را در الگوریتم backpropagation اعمال کردم به این شکل که به ازای هرلایه که بخواهم مشتق تابع هزینه را نسبت به وزن‌های مربوط به آن بدست آورم چون دارم مشتق را نسبت به آن وزن‌ها بدست می آورم فقط آن وزن‌ها باقی میمانند زیرا مشتق وزن‌های دیگر نسبت به آن وزن صفر میشود به این شکل مشتق آن را در الگوریتم backpropagation قرار دادم. همانطور که در شکل ۱۴ هم مشاهده میشود overfit نشده است زیرا هردو نمودار تست و ترین و همچنین ولیدیشن هم صعودی هستند.
مقدار لاندا در این الگوریتم 0.001 درنظر گرفته شد که نتایج بهتری نسبت به مقادیر دیگر میداد.

L۱ Regularization

در این روش مشابه روش L2 regularization است با این تفاوت که بجای به توان دو رساندن وزن‌ها قدرمطلق آنها را قرار میدهیم. در این روش هم هدف کاهش دادن وزن‌ها است اما در عمل در این روش بردار وزن‌ها تراکم کمتری دارد و این یک سری از فیچرها را حذف میکند درواقع فیچرهایی که اهمیت کمتری در مدل دارند را حذف میکند به همین دلیل این روش را به عنوان یک مکانیزم برای feature selection هم درنظر میگیرند.
در این الگوریتم درصورتی که مقدار لاندا بزرگ انتخاب میشد حتی در حدود 0.1 مقدار خطا زیاد میشد در حد ۷ زیاد میشد و مقدار آن را خیلی کوچک درنظر گرفتم. در مقدار لاندا 0.000001 خطا به حدود 0.6 رسید خروجی مدل با درنظرگرفتن L1 regularization در شکل ۱۶ و ۱۷ مشاهده میشود نمودار دقت داده ی تست همچنان افزایش پیدا کرده اما به طور کلی دقت کم شده است. همچنین نمودار خطا روندی نزولی دارد و این نکته مثبتی است.

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, perhaps 10^{-7}, for numerical stability
  Initialize gradient accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$.
    Compute update: $\Delta \boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$.   (Division and square root applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$.
  end while

شکل ۱: الگوریتم AdaGrad برگرفته از کتاب deep learning

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers
  Initialize accumulation variables $\boldsymbol{r} = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$.
    Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$.   ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)

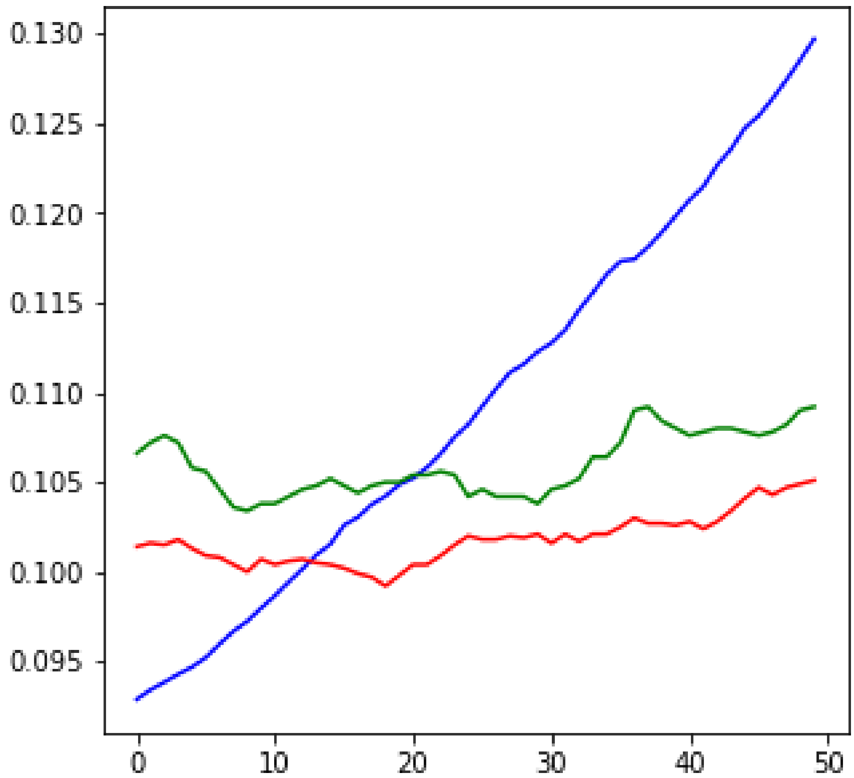
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$.
  end while

شکل ۲: الگوریتم RMSprob برگرفته از کتاب deep learning

Algorithm 8.7 The Adam algorithm

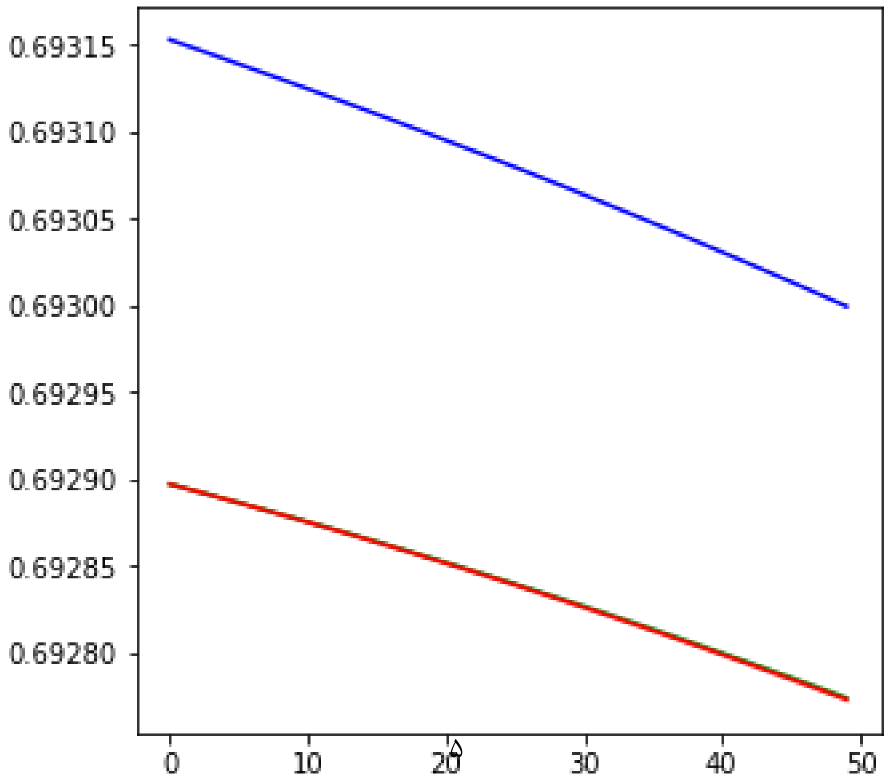
Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1) \boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2) \boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
    Compute update: $\Delta \boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

شکل ۳: الگوریتم Adam برگرفته از کتاب deep learning



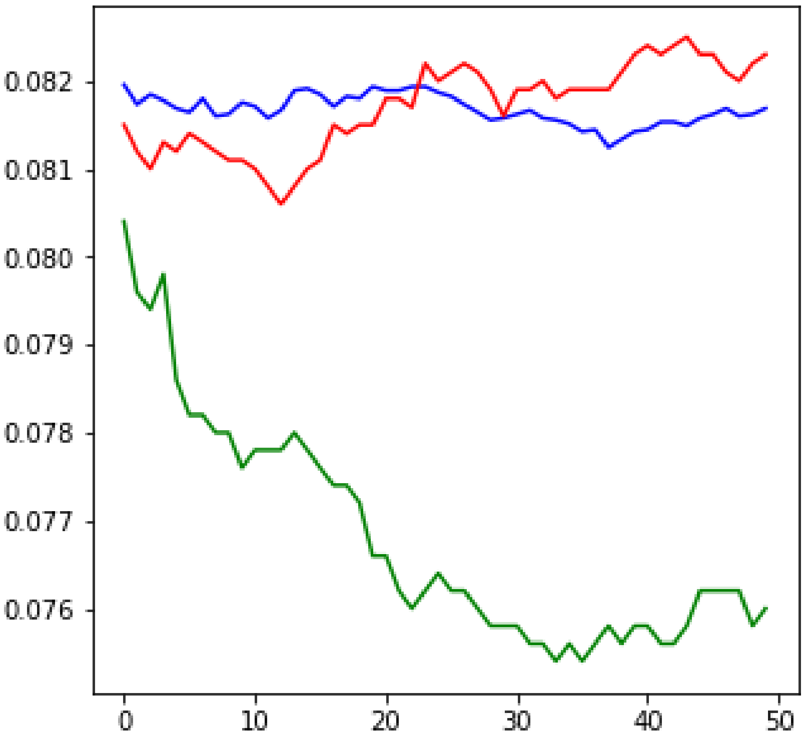
۵.

شکل ۴: نمودار دقت Gradient descent الگوریتم



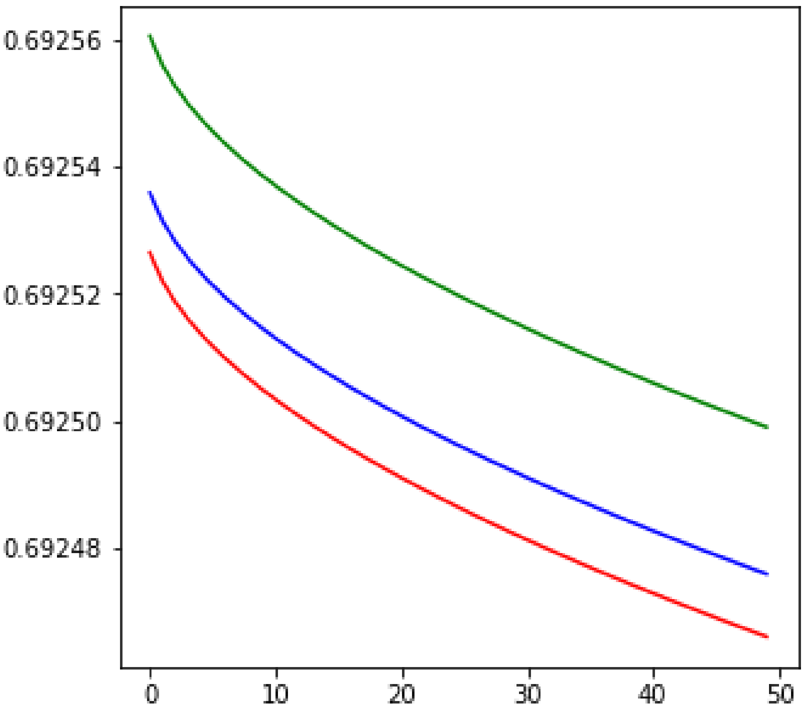
۵.

شکل ۵: نمودار خطا Gradient descent الگوریتم



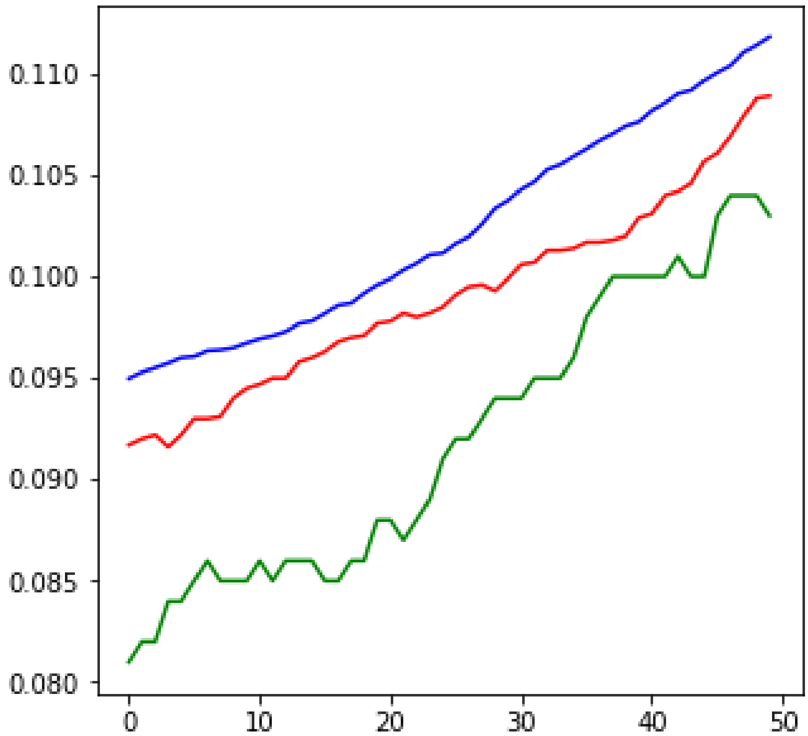
۵.

شکل ۶: نمودار دقت با learning rate مربوط به الگوریتم adograd



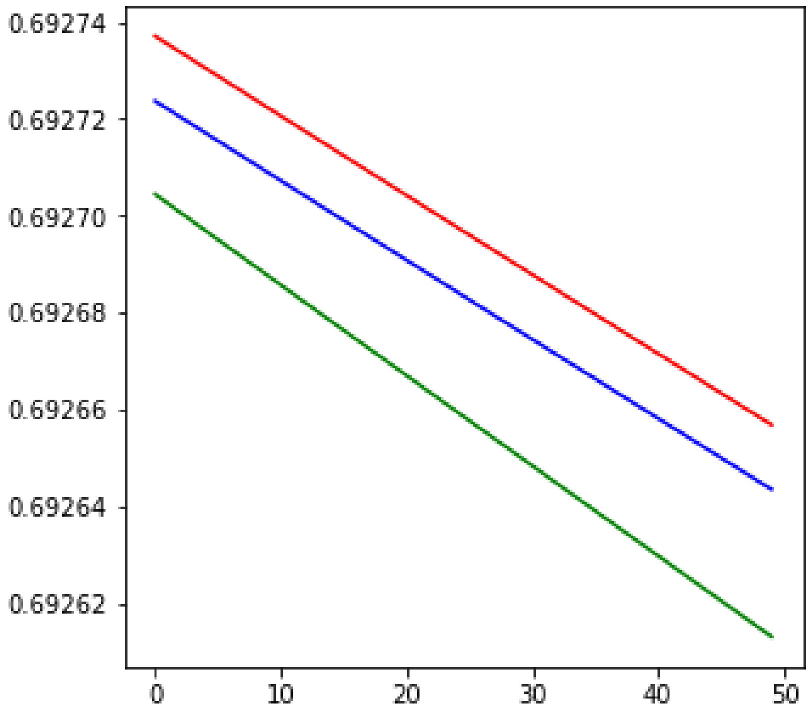
۵.

شکل ۷: نمودار خطا با learning rate مربوط به الگوریتم adograd



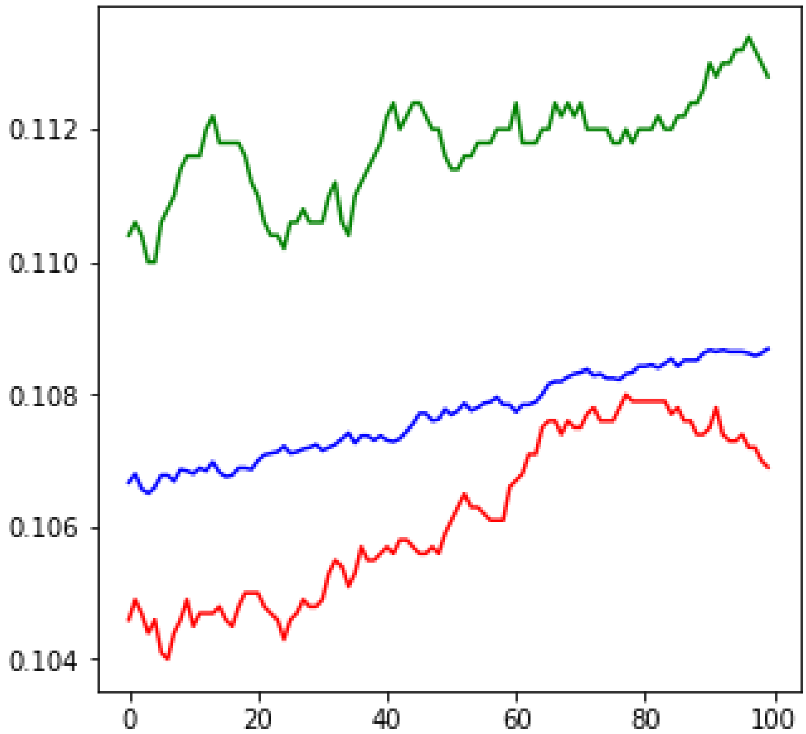
۵.

شکل ۸: نمودار دقت با learning rate مربوط به الگوریتم adam



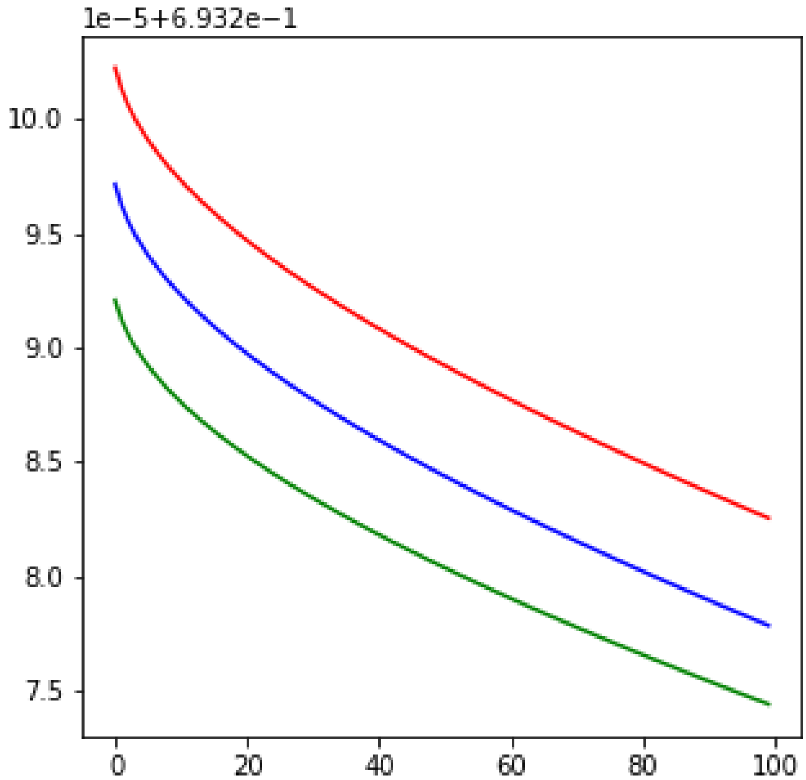
۵.

شکل ۹: نمودار خطا با learning rate مربوط به الگوریتم adam



۵.

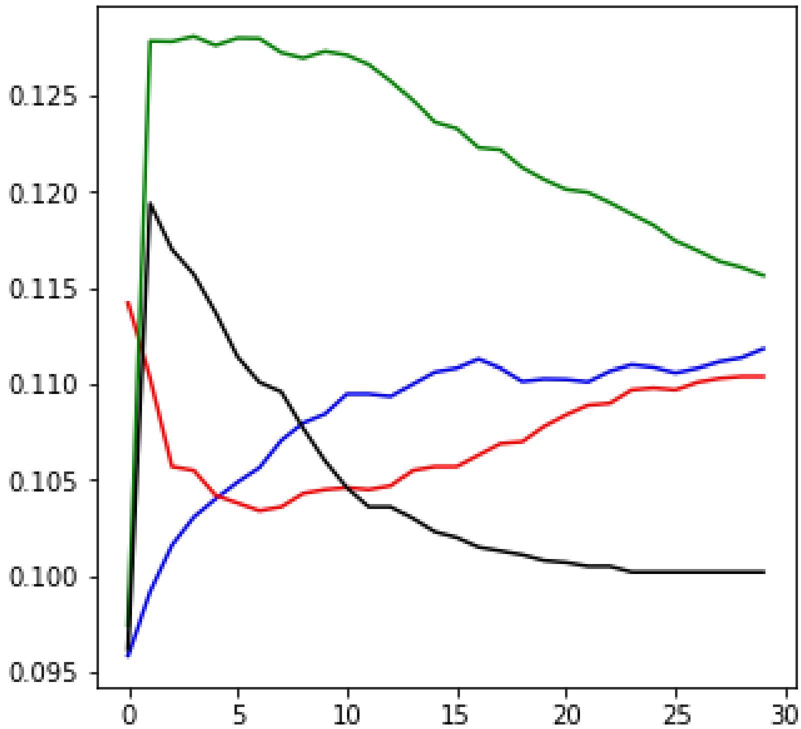
شکل ۱۰: نمودار دقت با learning rate مربوط به الگوریتم rmsprob



۸

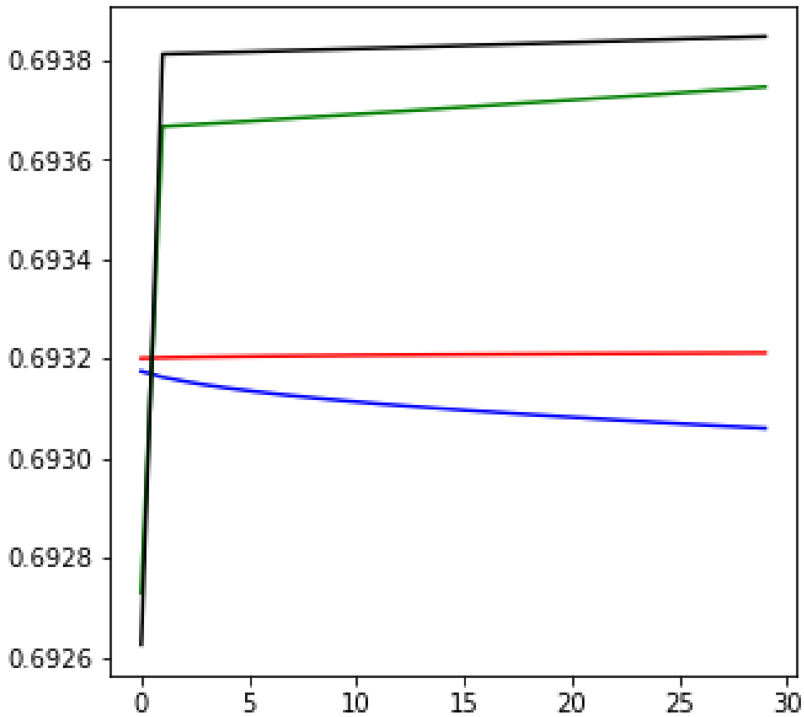
۵.

شکل ۱۱: نمودار خطا با learning rate مربوط به الگوریتم rmsprob



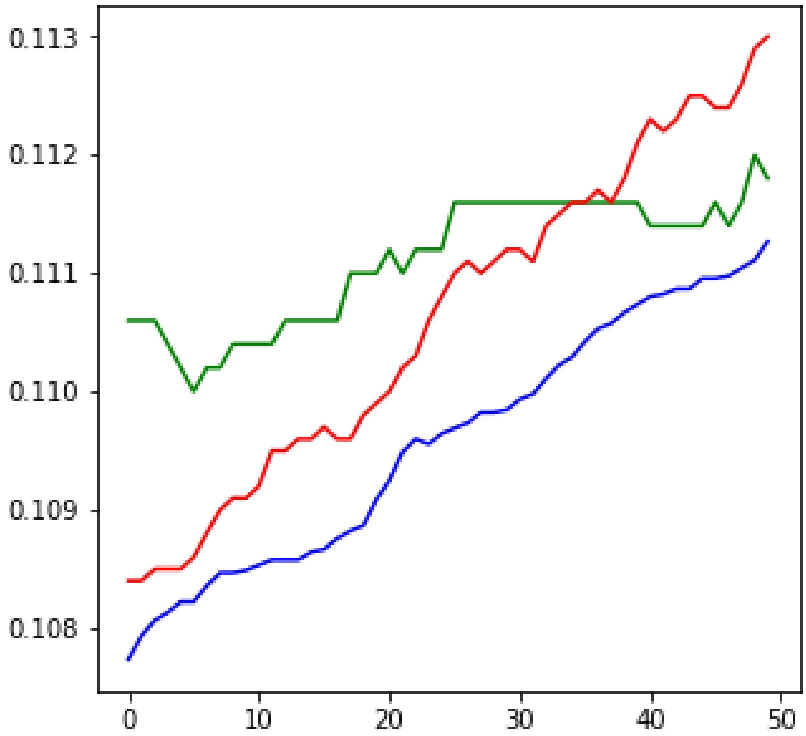
۵.

شکل ۱۲: نمودار دقت با learning rate مربوط به الگوریتم adograd و adadelta میباشد که نمودار سبز مربوط به داده های آموزش و مشکی داده‌های تست adagrad و آبی آموزش و قرمز تست adadelta میباشد



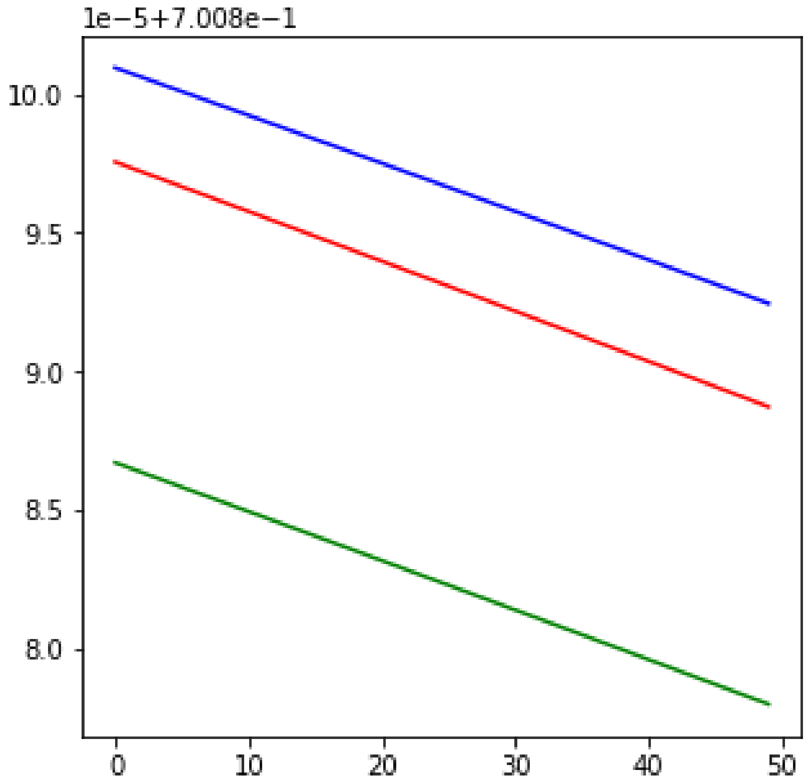
۵.

شکل ۱۳: نمودار خطا با learning rate مربوط به الگوریتم adograd و adadelta میباشد که نمودار سبز مربوط به داده های آموزش و مشکی داده‌های تست adagrad و آبی آموزش و قرمز تست adadelta میباشد



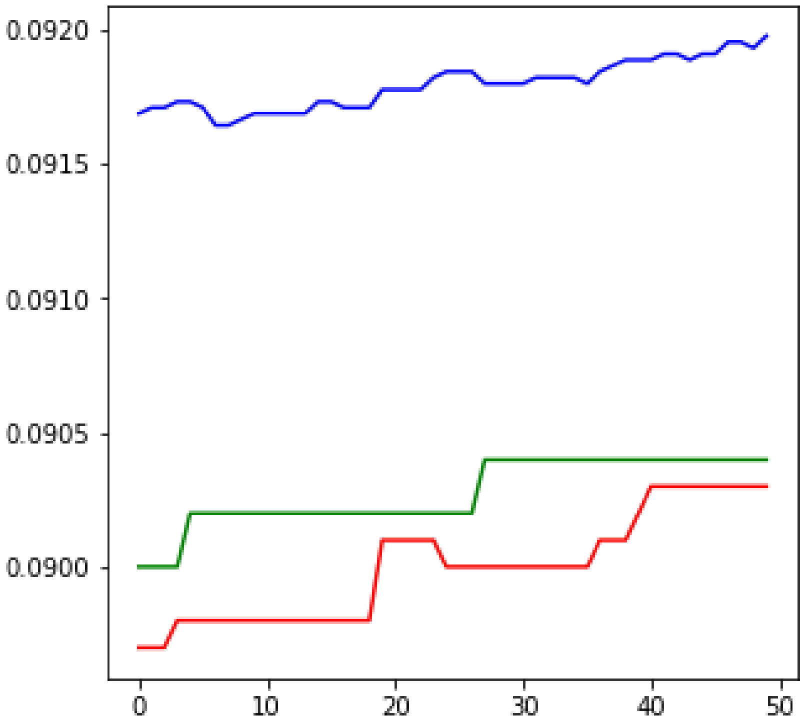
۵.

شکل ۱۴: نمودار دقت مدلی که L۲ Regularization در آن استفاده شده است.



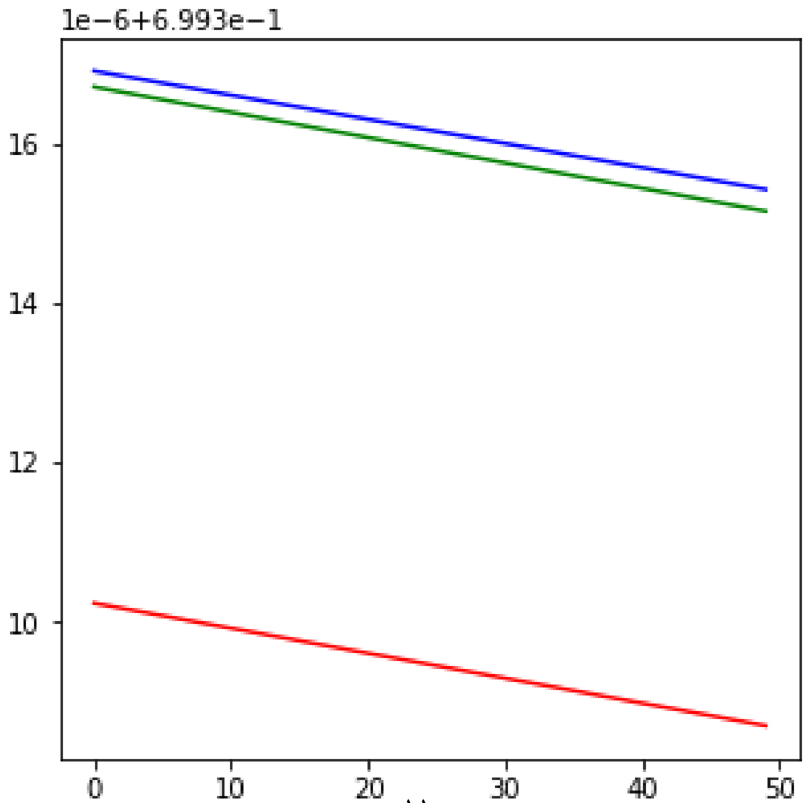
۵.

شکل ۱۵: نمودار خطا مدلی که L۲ Regularization در آن استفاده شده است.



۵.

شکل ۱۶: نمودار دقت مدلی که L۱ Regularization در آن استفاده شده است.



۵.

شکل ۱۷: نمودار خطا مدلی که L۱ Regularization در آن استفاده شده است.