یادگیری عمیق

تمرین سوم

استاد درس: دکتر محمدی

مهسا موفق بهروزی

زمستان ۱۴۰۱

سوال یک)

الف) مقاله برای معرفی الگوریتم AdaBelief می‌باشد که می‌توان با ایجاد اندکی تغییر در Adam و بدون اضافه کردن پارامتر جدید به آن، الگوریتم را به دست آورد. ایده بر این اساس است که step size را با استفاده از یک مفهوم به نام باور (belief) تعیین کند. به این‌صورت که اگر گرادیان مشاهده شده و پیش‌بینی شده، تا حد زیادی از یکدیگر فاصله داشته باشند به مشاهدات فعلی بی‌اعتماد شده و گام کوچکی برمی‌داریم و اگر گرادیان مشاهده شده نزدیک به پیش‌بینی شده باشد به آن اعتماد کرده و گام بزرگی برمی‌داریم.

مقاله ۳ مزیت برای AdaBelief ذکر کرده که عبارت‌اند از:

۱. همگرایی سریع مانند روش‌های گرادیان تطبیقی (مانند adam)
۲. تعمیم پذیری خوب مانند خانواده SGD
۳. پایداری مدل (training stability و عدم تغییر زیاد مدل در صورت تغییر دیتاست) در صورت استفاده از AdaBelief بیشتر است.

ب) با در نظر گرفتن g_t به عنوان گرادیان در نقطه t و m_t به عنوان momentum اول یا همان میانگین متحرک نمایی (EMA)، Adam و AdaBelief هر دو از momenteum اول با فرمول زیر برای آپدیت پارامترها استفاده می‌کنند:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

مومنتم اول برای این منظور استفاده می‌شود که تنها از گرادیان فعلی استفاده نکرده و ضریبی از گرادیان‌های قبلی را دخالت دهیم تا تخمینی از گرادیان را به دست آوریم.

تفاوت در ادامه‌ی روش است که Adam برای آپدیتِ adaptive، از توان دوی گرادیان در نقطه t با فرمول زیر استفاده می‌کند:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) {g_t}^2$$

اما AdaBelief از توان دوی اختلاف گرادیان واقعی و مقدار تخمینی آن (m_t) در نقطه t، به صورت زیر استفاده می‌کند:

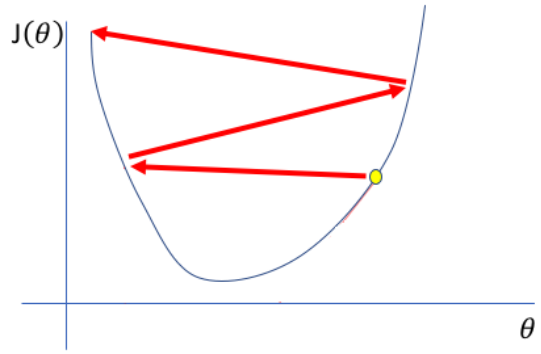
$$s_t = \beta_2 s_{t-1} + (1 - \beta_2)(g_t - m_t)^2$$

سپس در Adam برای آپدیت پارامترها از تقسیم m_t بر $\sqrt{v_t}$ و در AdaBelief از تقسیم m_t بر $\sqrt{s_t}$ استفاده می‌شود.

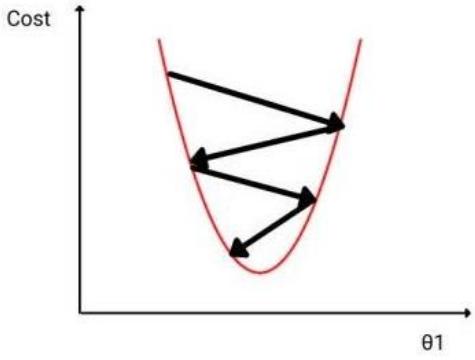
الگوریتم AdaBelief از $\frac{1}{\sqrt{s_t}}$ به عنوان باور (belief) نام می‌برد؛ اگر مقدار گرادیان (g_t) از مقدار پیش‌بینی آن (m_t) فاصله زیادی داشته باشد، مقدار s_t افزایش یافته و در نتیجه $\frac{1}{\sqrt{s_t}}$ کاهش می‌یابد و به این معنی‌ست که باور کمی داریم و به مشاهدات فعلی اعتماد نمی‌کنیم و قدم کوچکی برمی‌داریم. اگر مقدار گرادیان (g_t) و مقدار پیش‌بینی آن (m_t) نزدیک به یک‌دیگر باشند، مقدار s_t کاهش یافته و در نتیجه $\frac{1}{\sqrt{s_t}}$ افزایش می‌یابد و به این معنی‌ست که باور زیادی داریم و قدم بزرگی برمی‌داریم.

سوال دو)

الف) اگر مقدار نرخ یادگیری زیاد باشد، ممکن است منجر به overshoot یا پرش از نقطه بهینه گردد. در حالت شدید این پرش منجر به واگرایی شده و اصلا به نقطه بهینه همگرا نمی‌شود.



اما ممکن است مانند شکل زیر، پس از چندین پرش به نقطه بهینه همگرا شود اما به علت پرش‌ها، این همگرایی کند می‌شود و مقدار تابع ضرر دیرتر کاهش می‌یابد.



ب) از آن‌جایی که مقدار نرخ یادگیری نباید زیاد کوچک نیز باشد، زیرا سبب می‌شود بسیار کند همگرا شویم، می‌توان در ابتدا مقدار نرخ یادگیری را زیاد قرار داد تا گام‌های بزرگی به سمت نقطه بهینه برداریم و پس از مدتی با نزدیک شدن به نقطه بهینه، مقدار نرخ یادگیری را کاهش دهیم تا از نقطه بهینه پرش نکنیم. در حقیقت می‌توان مقدار نرخ یادگیری را متغیر در نظر گرفت.

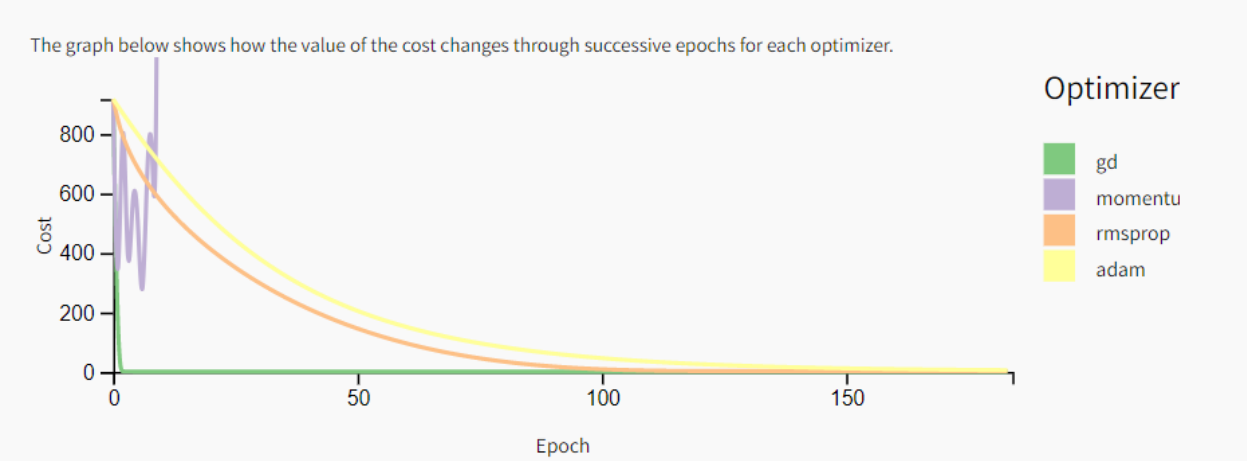
ت) با کوچک در نظر گرفتن نرخ یادگیری، تغییرات کمتری در وزن‌ها ایجاد می‌شود. بنابراین روند یادگیری کند می‌شود. و با در نظر گرفتن مقدار بزرگ برای آن دچار چالش‌هایی می‌شویم که در بخش الف اشاره شد.

ث) یکی از راه‌های حل مشکل sgd، افزودن momentum به آن می‌باشد. به این معنی که گرادیان‌های قبلی را فراموش نکند. برای این کار ضریبی از گرادایان‌های قبلی را به گرادیان فعلی اضافه کرده و سپس از مقدار به دست آمده برای آپدیت پارامترها استفاده می‌کنیم.

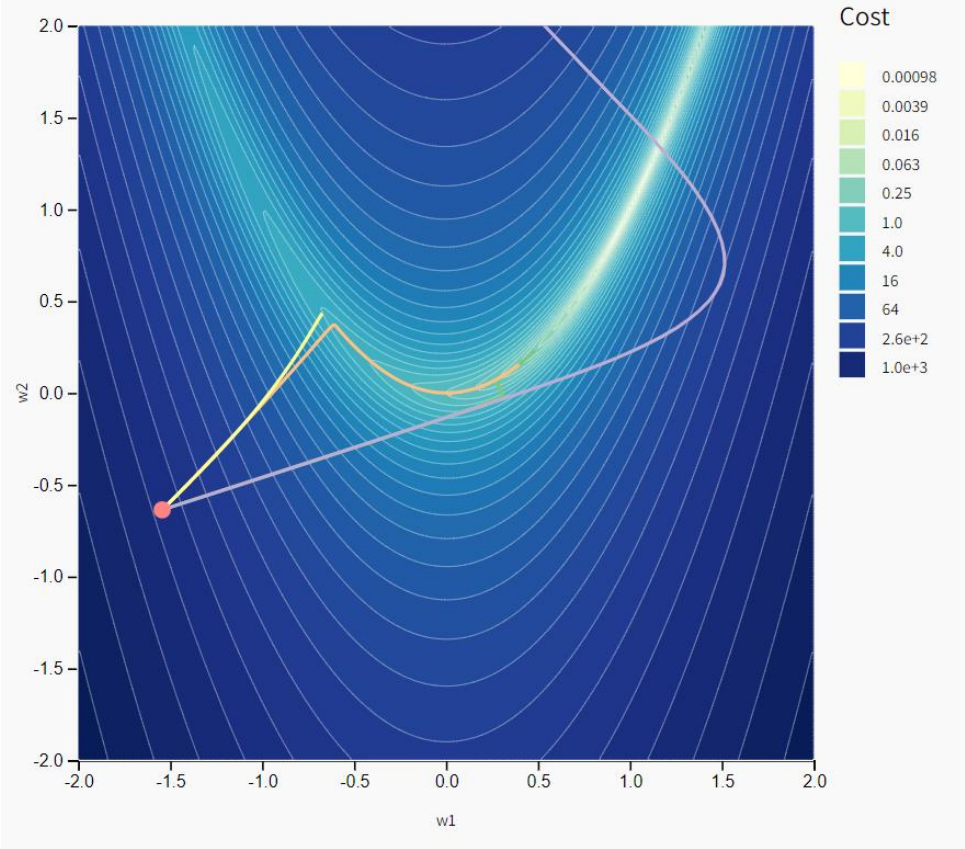
راه دیگر استفاده از Nesterov momentum است که به جای استفاده از گرادیان در نقطه فعلی، به جلو نگاه می‌کند و گرادیان را در نقطه‌ای محاسبه می‌کند که اگر با همین سرعت حرکت کند به آن‌جا می‌رسد. این مساله برای حل مشکل overshoot مومنتم مطرح شده است.

سوال سه)

الف) با در نظر گرفتن مقادیر پیش فرض(lr = 0.001 , lr decay =0)، gd بسیار سریع همگرا می‌شود. و momentum نیز از همان ابتدا واگرا می‌شود. Adam و RmsProb نیز با سرعت کمی به سمت مقادیر بهینه همگرا می‌شوند.



در این مثال با توجه به نمودار زیر که نشان می‌دهد نقاط کمرنگ‌تر، لاس کم‌تری دارند، gd و RMSProb به نقاط بهتری از Adam همگرا شده‌اند.



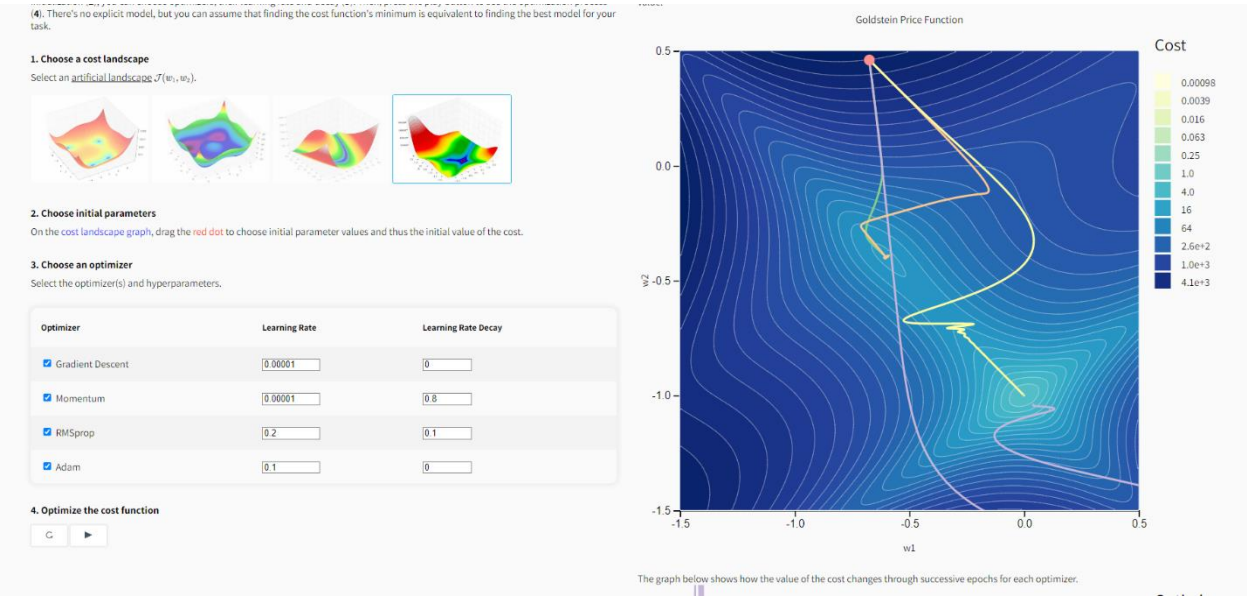
در مورد gd به نظر می‌رسد learning rate عدد مناسبی‌ست که با سرعت خوبی به نقطه‌ای بهینه همگرا شده. در momentum به نظر می‌رسد learning rate عدد بزرگی‌ست که باعث پرش از نقاط بهینه شده است. در Adam با توجه به سرعت بسیار آهسته نمودار در همگرا شدن به نقطه بهینه به نظر می‌رسد افزایش learning rate، اقدام مناسبی باشد.

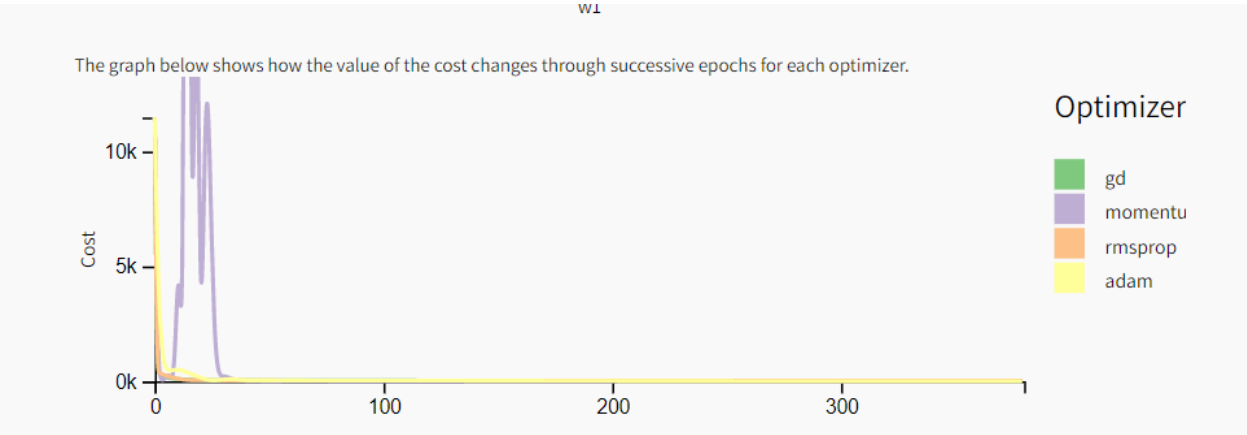
ب) ابتدا یک‌بار بهینه‌سازها با مقادیر پیش فرض تست شدند. باتوجه به پرش شدید در gd، باید مقدار نرخ یادگیری را بسیار کم کرد. با آزمایش مقادیر گوناگون، lr = 0.00001 در تعداد ایپاک‌های بالا به جواب بهینه همگرا می‌شود.

مساله واگرایی در momentum نیز با مقادیر پیش‌فرض اتفاق می‌افتد. با کاهش مقدار نرخ یادگیری به ۰.۰۰۰۰۱ و تنظیم lr decay به ۰.۸، هرچند که کاهش مقدار loss با نویز همراه است اما در ایپاک‌های بالاتر از ۵۰ به نقاط خوبی همگرا می‌شود.

در RMSProb مساله واگرایی رخ نمیدهد و از ابتدا به سمت نقاط بهینه حرکت می‌کند. در Adam با افزایش نرخ یادگیری به ۰.۱ با سرعت خوبی به نقطه بهینه همگرا می‌شود.

با این تنظیمات، adam و momentum از باقی بهینه‌سازها نتیجه بهتری گرفته و در بین این دو نیز momentum در تعداد ایپاک کمتر و با سرعت بیشتری بهینه می‌شود.





	lr	Lr decay
GD	0.00001	0
Momentum	0.00001	0.8
RMSProb	0.2	0.1
Adam	0.1	0

سوال چهار)

می‌توان به عنوان یک راه مساله را به این شکل حل کرد:

ابتدا مدل را با استفاده از داده‌های task اول آموزش می‌دهیم. سپس طبق فرض مساله به این داده‌ها دسترسی نداریم اما اگر مدل به خوبی آموزش دیده باشد، می‌توان چند داده‌ی رندوم که خارج از مجموعه داده‌ی task اول می‌باشد (مجموعه x) را به مدل داده، سپس خروجی مدل را به عنوان برچسب‌های مجموعه x در نظر گرفت. و این مجموعه را به همراه داده‌های task دوم برای آموزش مدل در نظر می‌گیریم و به همین ترتیب ادامه می‌دهیم.

سوال پنج)

①

$$\hat{y} = w_1 x_1^2 + w_2 x_2^2 + w_3 x_1 x_2 + b$$

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$\frac{\partial L}{\partial w_1} = \frac{1}{n} \sum_{i=1}^{n} -2 (y_i - \hat{y}_i) x_1^2$$

$$\frac{\partial L}{\partial w_2} = \frac{1}{n} \sum_{i=1}^{n} -2 (y_i - \hat{y}_i) x_2^2$$

$$\frac{\partial L}{\partial w_3} = \frac{1}{n} \sum_{i=1}^{n} -2 (y_i - \hat{y}_i) x_1 x_2$$

$$\frac{\partial L}{\partial b} = \frac{1}{n} \sum -2 (y_i - \hat{y}_i)$$

x_1	x_2	y	$\hat{y}$
1	-1	10	2
2	0	13	5

محاسبه $\hat{y}$ با numpy انجام شده است. $L = \frac{1}{2}(64+64) = 64$

$$\frac{\partial L}{\partial w_1} = -(8 \times 1 + 8 \times 4) = -40$$

$$\frac{\partial L}{\partial w_2} = -(8 \times (-1)^2 + 8 \times 0) = -8$$

$$\frac{\partial L}{\partial w_3} = -(8 \times (-1) + 8 \times 0) = 8$$

$$\frac{\partial L}{\partial b} = -(8 + 8) = -16$$

$vx = 0$

while True:

dx = compute

$vx = \rho vx + dx$

$x = x - \alpha vx$

$\Rightarrow$

$vx = 0$

$$dx = \begin{bmatrix} -40 \\ -8 \\ 8 \\ -16 \end{bmatrix}$$

$$vx = \rho vx + dx = \begin{bmatrix} -40 \\ -8 \\ 8 \\ -16 \end{bmatrix}$$

$w_1 = w_1 - \alpha vx = 1 - (0.1 \times -40) = 5$

$w_2 = w_2 - \alpha vx = -1 - (0.1 \times -8) = -0.2$

$w_3 = w_3 - \alpha vx = -1 - (0.1 \times 8) = -1.8$

$b = 1 - \alpha vx = 1 - (0.1 \times (-16)) = 2.6$

دو داده دیگر را با وزن های بالا آموزش میدهیم:

x_1	x_2	y	$\hat{y}$
0	2	11	1.8
-1	1	4	9.2

$$\frac{\partial L}{\partial w_1} = -((11 - 1.8) \times 0 + (4 - 9.2) \times 1) = 5.2$$

$$\frac{\partial L}{\partial w_2} = -(9.2 \times 4 + (-5.2) \times 1) = -31.59$$

$$\frac{\partial L}{\partial w_3} = -(9.2 \times 0 + (-5.2) \times -1) = -5.2$$

$$\frac{\partial L}{\partial b} = -(9.2 + (-5.2)) = -4$$

$$r x = 0.9 \times \begin{bmatrix} -40 \\ -8 \\ 8 \\ -16 \end{bmatrix} + \begin{bmatrix} 5.2 \\ -31.59 \\ -5.2 \\ -4 \end{bmatrix} = \begin{bmatrix} -30.8 \\ -38.79 \\ 2 \\ -18.4 \end{bmatrix}$$

$$w_1 = w_1 - \alpha r x = 5 - (0.1) \times (-30.8) = 8.08$$

$$w_2 = w_2 - \alpha r x = -0.2 - (0.1) \times (-38.79) =$$

$$w_3 = w_3 - \alpha r x = -1.8 - (0.1) \times (2) = -2$$

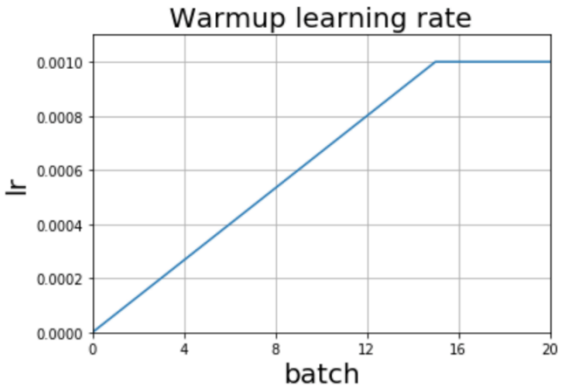
$$b = b - \alpha r x = 2.6 - (0.1) \times (-18.4) = 4.44$$

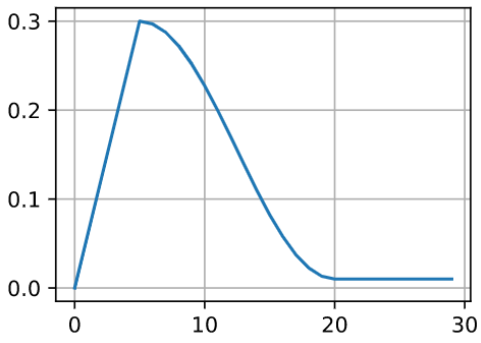
مقدار w2 بر روی کاغذ فراموش شده بود.

W2 = 3.679

سوال ششم)

الف) تکنیک warmup به این شیوه است که طی یک دوره‌ی warmup، نرخ یادگیری تا یک مقدار بیشینه افزایش یافته وسپس تا پایان فرآیند بهینه‌سازی، کاهش می‌یابد. برای سادگی می‌توان warmup خطی را مثال زد که یک scheduler نرخ یادگیری‌ست در آن نرخ یادگیری را به صورت خطی تا رسیدن به یک مقدار ثابت افزایش می‌دهیم.





ایده warmup نرخ یادگیری ساده است .در مراحل اولیه آموزش - وزن ها از حالت ایده آل خود فاصله دارند .این به معنای به روز رسانی های فراوان پارامترهای زیاد است که می تواند به عنوان "overcorrections" برای هر وزن دیده شود - که در آن به روز رسانی شدید یک وزن ممکن است به روز رسانی وزن دیگری را نفی کند و مراحل اولیه آموزش را ناپایدارتر کند .این تغییرات برطرف می‌شوند، اما می‌توان با داشتن یک نرخ یادگیری کوچک برای شروع و سپس اعمال نرخ یادگیری بزرگ‌تر از آن‌ها جلوگیری کرد.

همچنین اگر مجموعه داده‌ها بسیار متمایز باشند، ممکن است مدل دچار «early over-fitting»شود. اگر داده‌های shuffle شده شامل مجموعه‌ای از مشاهدات مرتبط با ویژگی‌های قوی باشد، آموزش اولیه مدل می‌تواند به شدت به سمت آن ویژگی‌ها منحرف شود. Warmup روشی برای کاهش اثر برتری نمونه‌های آموزشی اولیه است .بدون آن، ممکن است لازم باشد چند دوره اضافی را اجرا کنیم تا به همگرایی مورد نظر دست یابیم.

همچنین این تکنیک، واگرایی پارامترها در شبکه‌های عمیق را محدود می‌کند.

مرجع:

12.11. Learning Rate Scheduling — Dive into Deep Learning 1.0.0-beta0 documentation (d2l.ai)

https://stackoverflow.com/questions/55933867/what-does-learning-rate-warm-up-mean

https://stackabuse.com/learning-rate-warmup-with-cosine-decay-in-keras-and-tensorflow/

ب) تفاوت واضح و عمده‌ی این دو، در این است که learning rate decay با نرخ یادگیری زیاد شروع شده و سپس چندین بار آن را کاهش می‌دهد. اما warmup با مقدار کم شروع شده و افزایش می‌یابد.

ت) همان‌طور که در نوت بوک مشاهده می‌شود، شبکه در ابتدا به خوبی همگرا می‌شود. (زیر ۵ ایپاک)