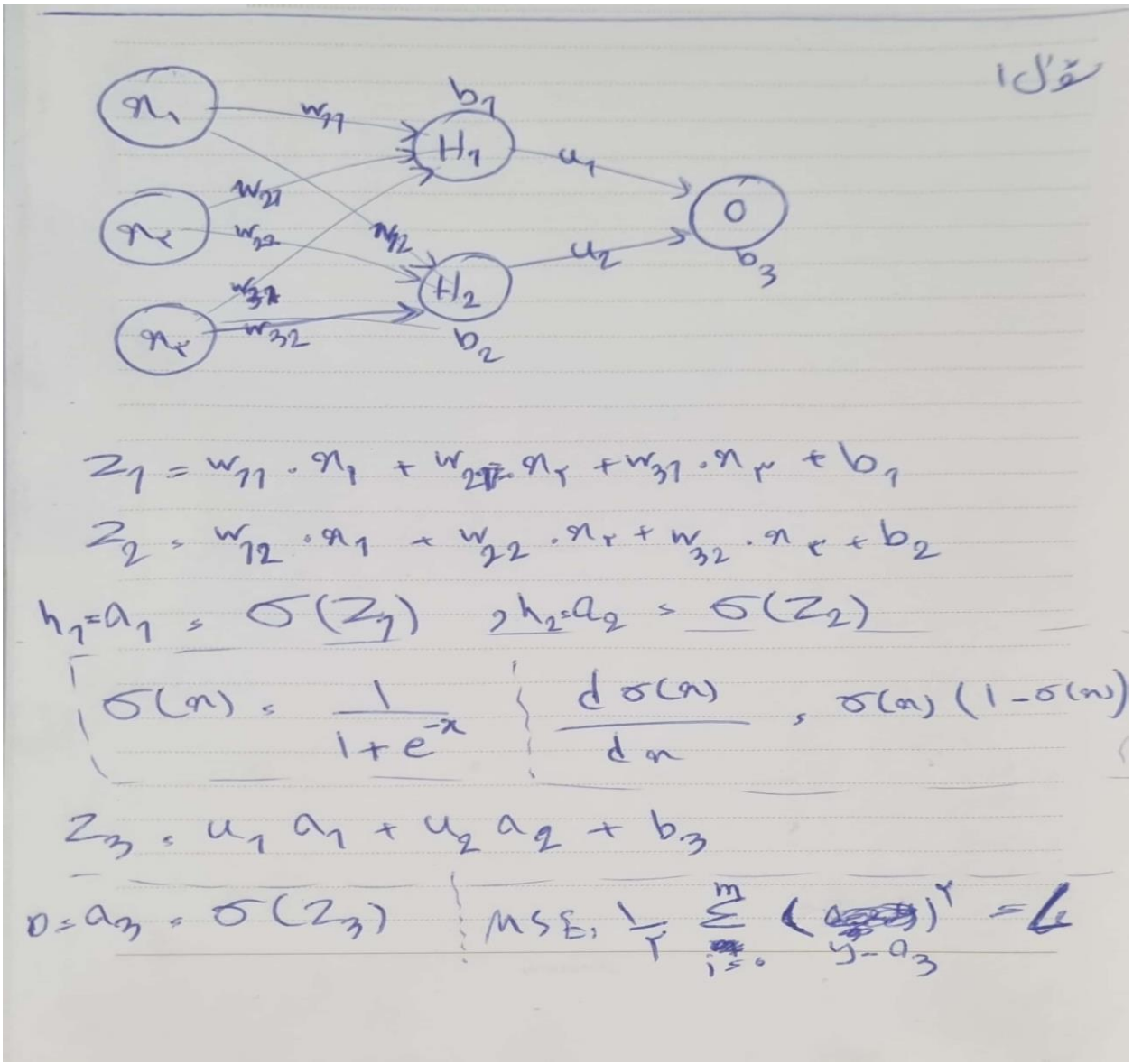
Mahan Veisi – CN paper HW3 – 400243081

-1 با توجه به موارد گفته شده، سعی میکنیم ابتدا همه مراحل forward و backward را روی کاغذ نوشته و پس از بدست اوردن مشتق ها، کد آنها را مینویسم که بتوانیم برای هر تعداد epochای نتیجه را بدست آوریم. البته با توجه به اینکه در صورت سوال مقدار beta برای momentum ذکر نشده است، فرض میکنیم که Beta برابر 0.5 میباشد.



$$\frac{\partial L}{\partial a_3} = \left(\frac{1}{2}\right)(2)\left(\frac{y - a_3}{a_3(1-a_3)}\right)(-1) = a_3 - y$$

$$\frac{\partial L}{\partial z_3} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} = (a_3 - y)\left((a_3)(1-a_3)\right)$$

$$\frac{\partial L}{\partial b_3} = \frac{\partial L}{\partial z_3} \cdot \frac{\partial z_3}{\partial b} = \frac{\partial L}{\partial z_3} \cdot (1)$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_1} = \frac{\partial L}{\partial z_3} \cdot (a_1)$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial z_3} \cdot (a_2)$$

$$\frac{\partial L}{\partial z_1} = \left(w_1 \times \frac{\partial L}{\partial z_3}\right)\left(a_1 \times (1 - a_1)\right)$$

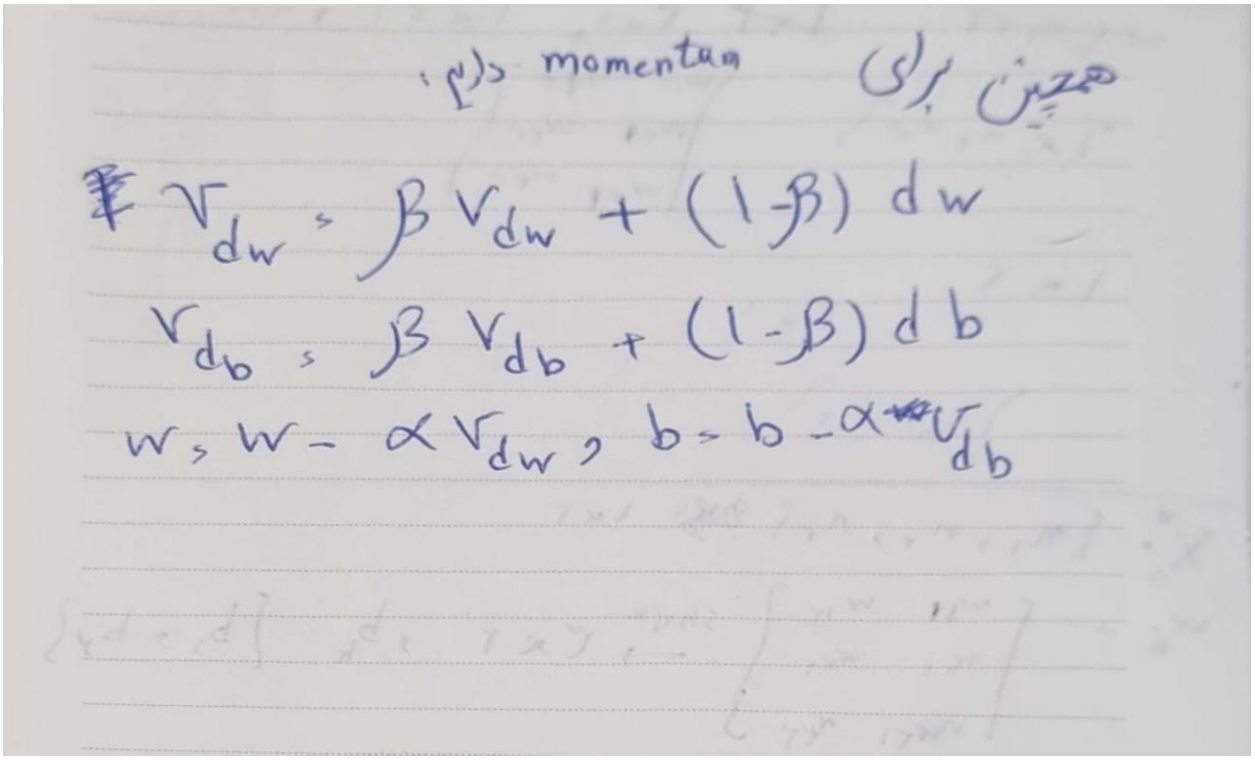
$$\frac{\partial L}{\partial z_2} = \left(w_2 \times \frac{\partial L}{\partial z_3}\right)\left(a_2 \times (1-a_2)\right)$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial z_1} \cdot (1) \;,\; \frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial z_2} \cdot (1)$$

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial z_1} \cdot \underbrace{\frac{\partial z_1}{\partial w_{11}}}_{x_1} = \frac{\partial L}{\partial z_1} \cdot (x_1)$$

$$\vdots$$

$$\frac{\partial L}{\partial w_{32}} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_{32}} = \frac{\partial L}{\partial z_2} \cdot (x_3)$$




```
Initial Values:
w11: 0.2 w12: -0.3 w21: 0.4 w22: 0.1 w31: -0.5 w32: -0.2
b1: -0.4 b2: 0.2
u1: -0.3 u2: -0.2
b3: 0.1
-------------------------
Epoch 0 - Loss: 0.135799
w11: 0.1984429214279178 | w12: -0.301144545775947 | w21: 0.4 | w22: 0.1 | w31: -0.5015570785720822 | w32: -0.201144545775947
b1: -0.4015570785720822 | b2: 0.198855454224053
u1: -0.2922323303072442 | u2: -0.19003776929413585
b3: 0.12340983556727902
momentum_w11: 0.0017300873023135504 | momentum_w12: 0.0012717175288300007
momentum_w21: 0.0 | momentum_w22: 0.0
momentum_w31: 0.0017300873023135504 | momentum_w32: 0.0012717175288300007
momentum_b1: 0.0017300873023135504 | momentum_b2: 0.0012717175288300007
momentum_u1: -0.00863074410306195 | momentum_u2: -0.011069145228737967
momentum_b3: -0.026010928408087802
```

```
-------------------------
Epoch 1 - Loss: 0.131835
w11: 0.19570356464439803 | w12: -0.30313231476414637 | w21: 0.4 | w22: 0.1 | w31: -0.504296435355602 | w32: -0.2031323147641464
b1: -0.404296435355602 | b2: 0.19686768523585363
u1: -0.27838049210251936 | u2: -0.17226117074927427
b3: 0.16522786420514388
momentum_w11: 0.003043729759466403 | momentum_w12: 0.0022086322091104116
momentum_w21: 0.0 | momentum_w22: 0.0
momentum_w31: 0.003043729759466403 | momentum_w32: 0.0022086322091104116
momentum_b1: 0.003043729759466403 | momentum_b2: 0.0022086322091104116
momentum_u1: -0.015390931338583175 | momentum_u2: -0.01975177616095729
momentum_b3: -0.04646447626429427
-------------------------
Final Values:
w11: 0.19570356464439803 | w12: -0.30313231476414637 | w21: 0.4 | w22: 0.1 | w31: -0.504296435355602 | w32: -0.2031323147641464
b1: -0.404296435355602 |b2: 0.19686768523585363
u1: -0.27838049210251936 |u2: -0.17226117074927427
b3: 0.16522786420514388
momentum_w11: 0.003043729759466403 | momentum_w12: 0.0022086322091104116
momentum_w21: 0.0 | momentum_w22: 0.0
momentum_w31: 0.003043729759466403 | momentum_w32: 0.0022086322091104116
momentum_b1: 0.003043729759466403 | momentum_b2: 0.0022086322091104116
momentum_u1: -0.015390931338583175 | momentum_u2: -0.01975177616095729
momentum_b3: -0.04646447626429427
-------------------------
-------------------------
Prediction: 0.5001850044665491
Final Loss: 0.1318351471295045
```

link:

[full colab link for epochs with 1000 and acc of 0.98]

```
b1: -0.32916322996591035 | b2: 0.45075469678407065
u1: 0.6277185120534774 | u2: 1.1141282010835714
b3: 2.9719277540455233
momentum_w11: -5.830332334211167e-05 | momentum_w12: -0.00010438959351724897
momentum_w21: 0.0 | momentum_w22: 0.0
momentum_w31: -5.830332334211167e-05 | momentum_w32: -0.00010438959351724897
momentum_b1: -5.830332334211167e-05 | momentum_b2: -0.00010438959351724897
momentum_u1: -0.00015004355195756438 | momentum_u2: -0.00024100179849360508
momentum_b3: -0.0003945421189974547
-------------------------
Final Values:
w11: 0.27579122597486133 | w12: -0.04042573980244602 | w21: 0.4 | w22: 0.1 | w31: -0.4242087740251393 | w32: 0.05957426019755333
b1: -0.3242087740251386 |b2: 0.4595742601975535
u1: 0.6403757937271692 |u2: 1.1344695310088457
b3: 3.0050558534893477
momentum_w11: -5.312596063432039e-05 | momentum_w12: -9.403469798946821e-05
momentum_w21: 0.0 | momentum_w22: 0.0
momentum_w31: -5.312596063432039e-05 | momentum_w32: -9.403469798946821e-05
momentum_b1: -5.312596063432039e-05 | momentum_b2: -9.403469798946821e-05
momentum_u1: -0.00013476652557375558 | momentum_u2: -0.0002166887815389207
momentum_b3: -0.0003511155952939901
-------------------------
Prediction: 0.9811355379766513
Final Loss: 0.00017813405068907892
```

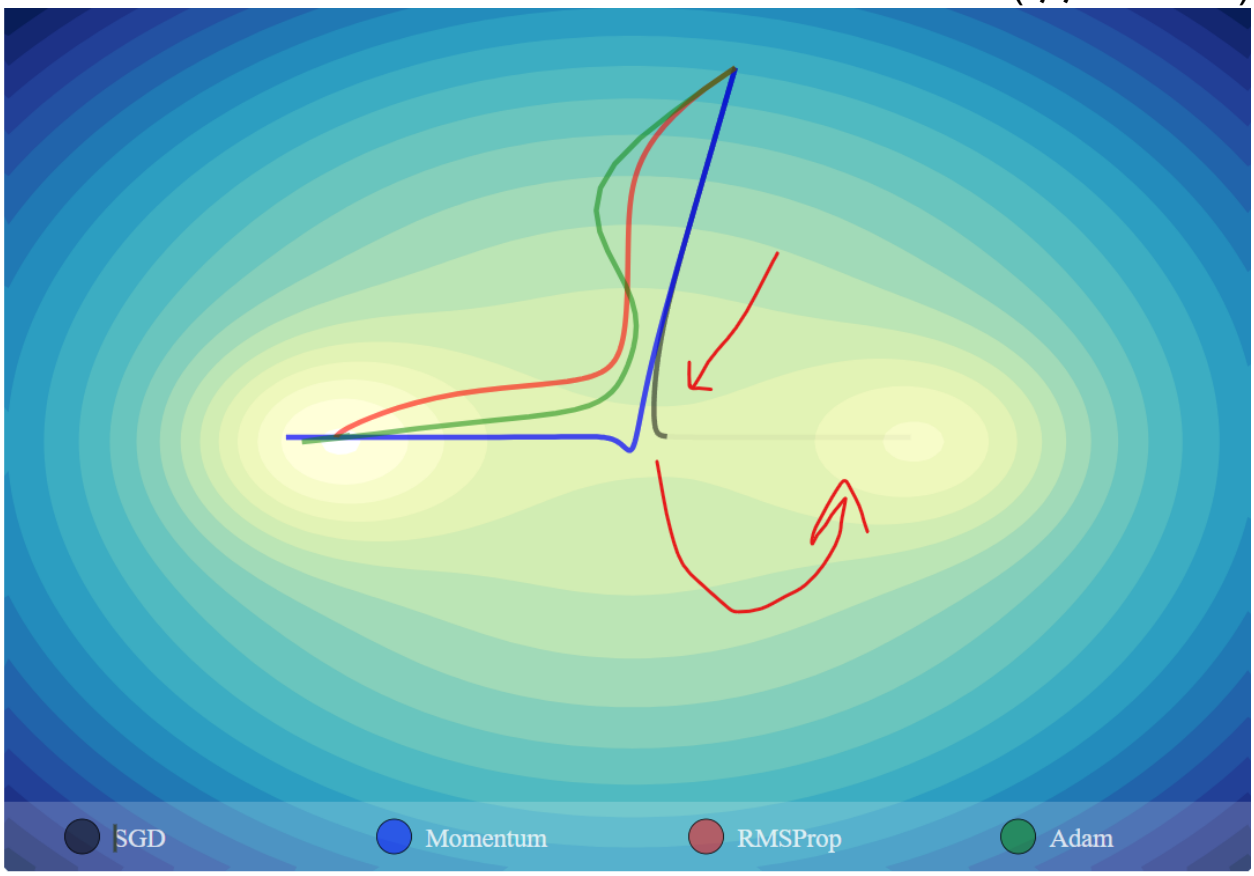
2- برای این سوال ابتدا بهتر است نگاهی به هریک از الگوریتم ها بیاندازیم:

SGD: در این الگوریتم که از پایه ای ترین نوع optimization است، هر بار مدل یکی از دیتا ها (در برخی اوقات قسمتی از آن) را به صورت random انتخاب کرده و پس از پیشبینی، به محاسبه loss پرداخته و در نهایت روی همین یک بررسی، پارامتر ها آپدیت میشوند. که همانطور که از توضیحات پیداست، این الگوریتم میتواند دچار نویز شود چون هربار روی یک داده کار میکند، پس با وجود سادگی آن و داشتن کمترین پارامتر های اضافی و بهینه بودن در بحث حافظه(فقط learning rate)، ممکن است بسیار کند بوده،

دیر به همگرایی برسد و پایداری زیادی ندارد.

$$\theta_{t+1} = \theta_t - \alpha \nabla J(\theta_t, x^{(i)}, y^{(i)})$$

در شکل ها هم مشاهده میکنیم که معمولا SGD علاوه بر اینکه دیر به نقطه optimal میرسد، در برخی موارد چون به آموخته های قبلی توجه نداشته و صرفا همان داده انتخاب شده حال حاضرش را بررسی میکند (به نوعی گریدی است) به بهترین نقطه optimal (دایره سمت چپ) نمیرسد:



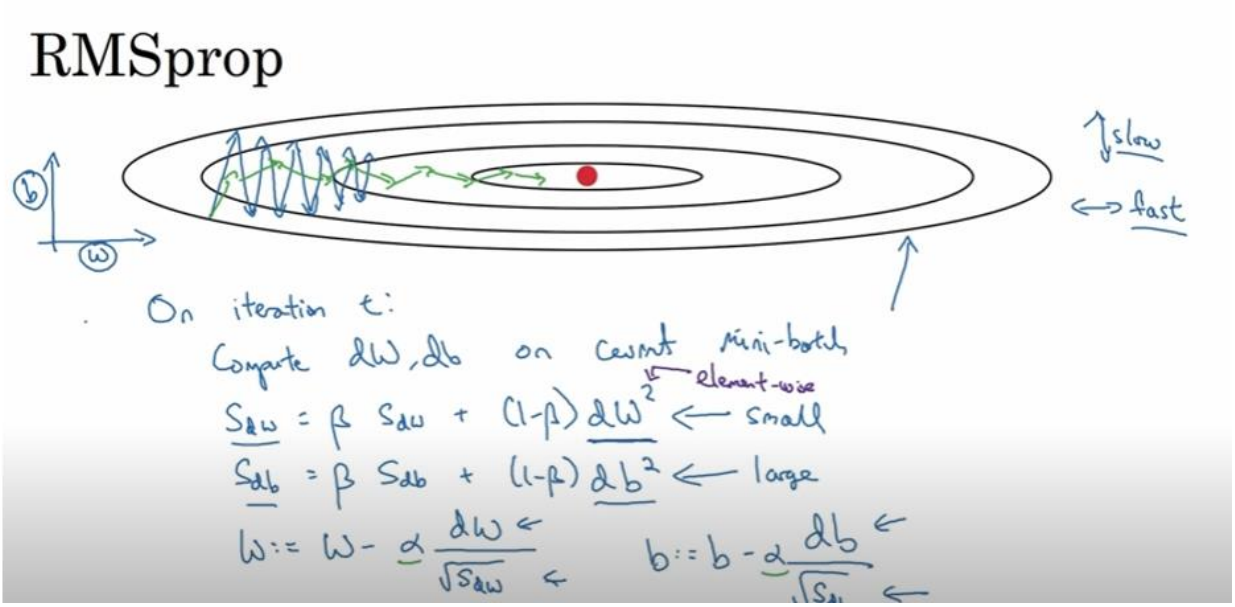
در تصویر مشاهده میکنیم که سایر الگوریتم ها به بهترین جواب رسیده اند در حالی که SGD علاوه بر اینکه به جواب نرسیده است، در آخر نیز به بهترین جواب نمیرسد!

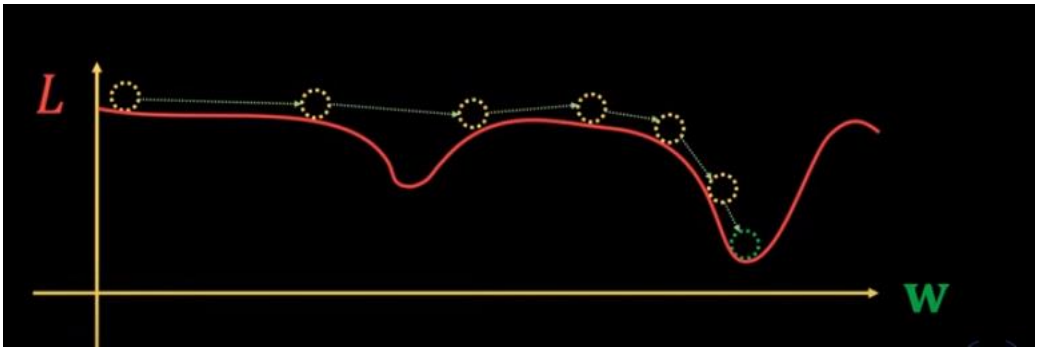
Momentum: در این الگوریتم همانطور که در سوال قبلی نیز کمی توضیح داده شد، به نوعی از تکانه داده های قبلی استفاده میشود، پس علاوه بر آنچه در SGD وجود دارد، یک پارامتر برای ذخیره سازی نیزداریم که کمک کند راحت تر بدانیم الگوریتم به کدام سمت در حال همگرا شدن است

$$v_{t+1} = \beta v_t + (1-\beta) \nabla J(\theta_t, x(i), y(i))$$
$$\theta_{t+1} = \theta_t - \alpha v_{t+1}$$

پس به طور کلی با داشتن یک پارامتر اضافی میتواند مقداری بیشتر بار حافظه ای داشته باشد اما سرعت همگرایی و رسیدن به جواب بهینه در آن بسیار بیشتر است چون مسیر را به طریقی حفظ میکند و در نتیجه میتواند نسبت به SGD برای داده های نویزی مقاوم تر باشد. و میتوان گفت نسخه ای اپگرید شده از SGD است، که البته محبوبیت بیشتری نیز دارد.

RMSprop: در این حالت به نوعی adaptive learning rate داریم که برخلاف momentum که لرنینگ ریت آن ثابت است، بر اساس کسری از squareهریک از پارامتر ها (به صورت مجزا) تغییر میکند به گونه ای که در سمت نویز ها آپدیت کمتر و در سمت تغییر های flat، قدم های بزرگتری برمیدارد.



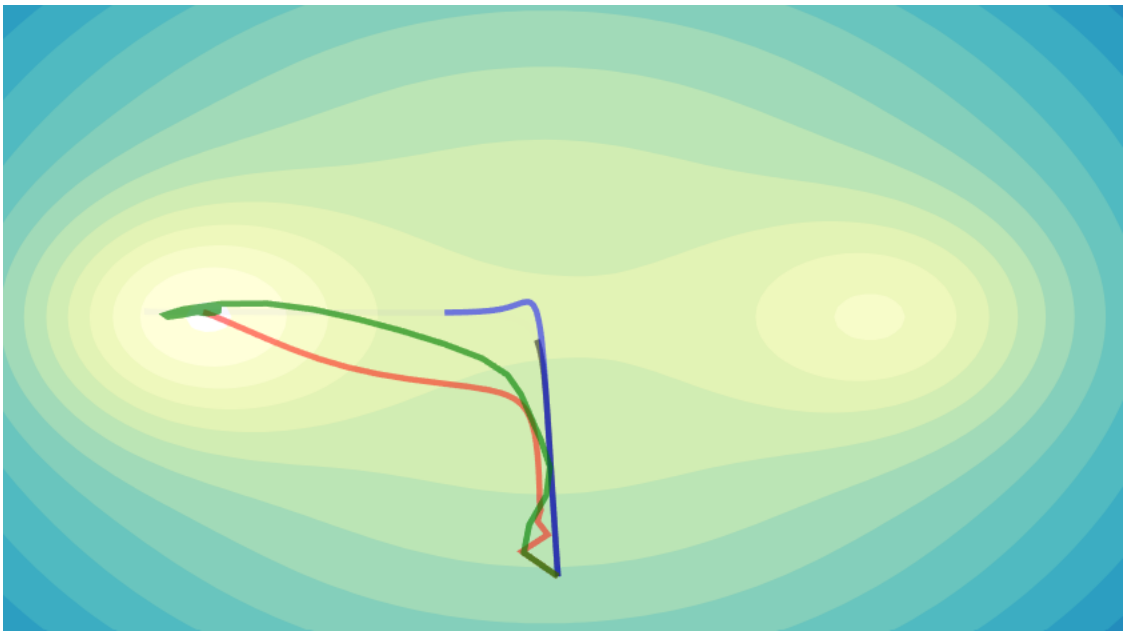


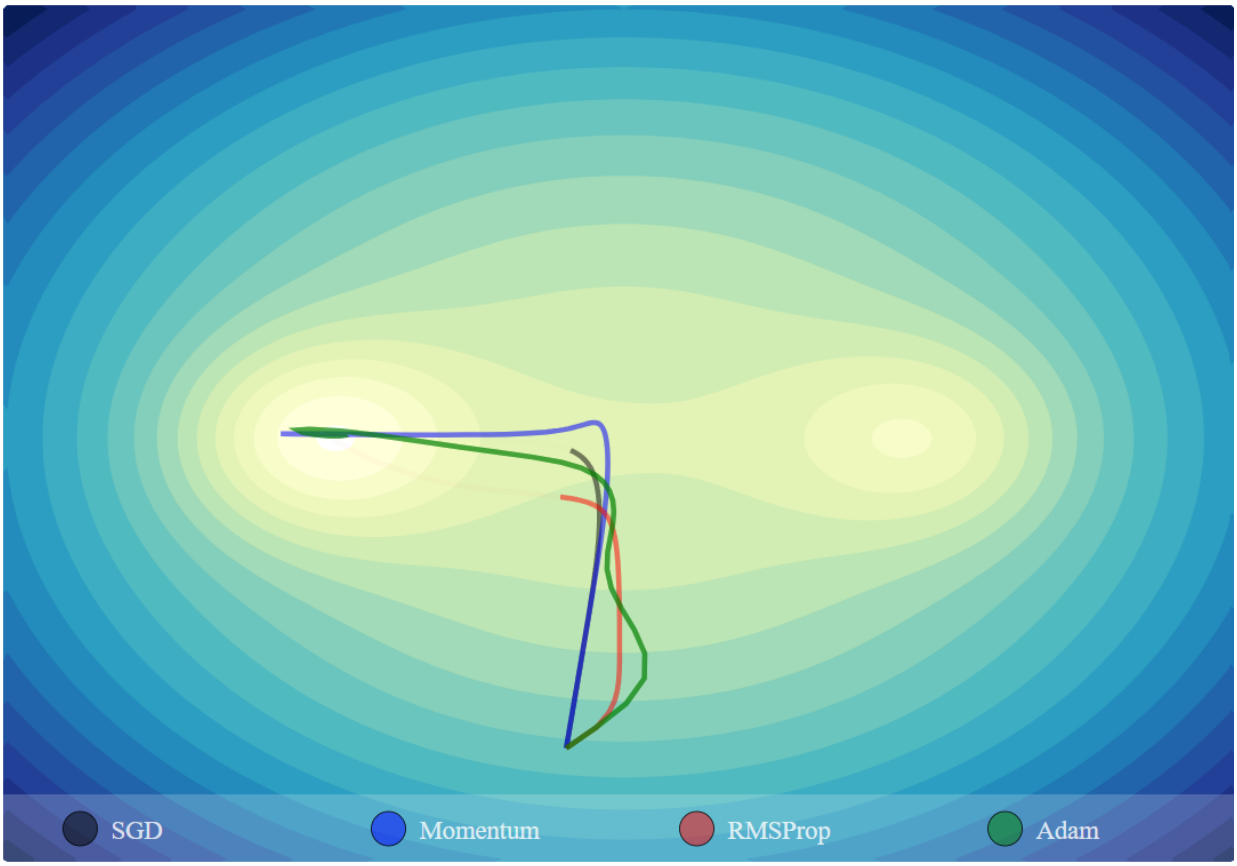
همچنین این الگوریتم نسبت به momentum در مقیاس های متفاوت باز میتواند عمل کند و چون به طور مستقیم با پارامتر هایی که learning rate قرار است روی آن ها تاثیر بگذارد، در ارتباط است، حتی اگر learning rate ما خیلی دقیق نباشد، بازهم میتواند خود را با شرایط سازگار کند در حالی که این یک نقص برای momentum و نیز SGD است.

$$vt+1=\beta vt+(1-\beta)(\nabla J(\theta t))2$$

$$\theta t+1=\theta t-vt+1+\epsilon\alpha\nabla J(\theta t)$$

به طور کلی اینکه از میان این دو الگوریتم، یعنی momentum و RMSprop کدامیک سریعتر هستند، بحث اشتباهی است زیرا کاملا به شرایط بستگی دارد:





چون هرکدام از آنها از یک ویژگی خاص و متمایز از دیگری استفاده میکنند اما با در نظر گرفتن برخی ویژگی های برتری که RMSprop داشته و در بالا اشاره شد، در شرایطی که learning rate و نیز سایر بحث های مسئله کاملا درست ست شده باشند و به طوری کلی مسئله ما خیلی پیچیده نباشد، momentum میتواند در برخی موارد به دلیل داشتن حافظه ای برای سرعت قبلی، زودتر به نتیجه برسد. اما همانطور که گفته شد مقایسه سرعت آنها به طور کلی درست نیست. بحثی که مهم است این است که در Adam از ویژگی های منحصر به فرد هرکدام استفاده میکنیم تا به یک الگوریتم بسیار قوی برسیم.

Adam: این الگوریتم از دو ایده جالب momentum و RMSprop استفاده میکند و با ترکیب آن دو به یکی از بهترین الگوریتم های optimization تبدیل میشود. پس به طور کلی علاوه بر داشتن exponentially decaying average پارامتر ها (از momentum) و exponentially decaying average of past squared gradients (از RMSprop) یک بایاس هم اضافه میکند تا در برخی موارد به مشکل نخوریم:

Adam optimization algorithm

$V_{dW} = 0, S_{dW} = 0, V_{db} = 0, S_{db} = 0$

On iteration t:

Compute dW, db using current mini-batch

$V_{dW} = \beta_1 V_{dW} + (1-\beta_1) dW$, $V_{db} = \beta_1 V_{db} + (1-\beta_1) db$ ← "momentum" β_1

$S_{dW} = \beta_2 S_{dW} + (1-\beta_2) dW^2$, $S_{db} = \beta_2 S_{db} + (1-\beta_2) db$ ← "RMSprop" β_2

$V_{dW}^{corrected} = V_{dW} / (1-\beta_1^t)$, $V_{db}^{corrected} = V_{db} / (1-\beta_1^t)$

$S_{dW}^{corrected} = S_{dW} / (1-\beta_2^t)$, $S_{db}^{corrected} = S_{db} / (1-\beta_2^t)$

$$W := W - \alpha \frac{V_{dW}^{corrected}}{\sqrt{S_{dW}^{corrected}} + \varepsilon} \qquad b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected}} + \varepsilon}$$

Andrew Ng

$$m_{t+1} = \beta_1 m_t + (1-\beta_1) \nabla J(\theta_t, x^{(i)}, y^{(i)})$$

$$v_{t+1} = \beta_2 v_t + (1-\beta_2) \nabla J(\theta_t, x^{(i)}, y^{(i)}))^2$$

$$\grave{m}_{t+1} = 1 - \beta_1^{t+1} m_{t+1}$$

$$\grave{v}_{t+1} = 1 - \beta_2^{t+1} v_{t+1}$$

$$\theta_{t+1} = \theta_t - \grave{v}_{t+1} + \epsilon \alpha \grave{m}_{t+1}$$

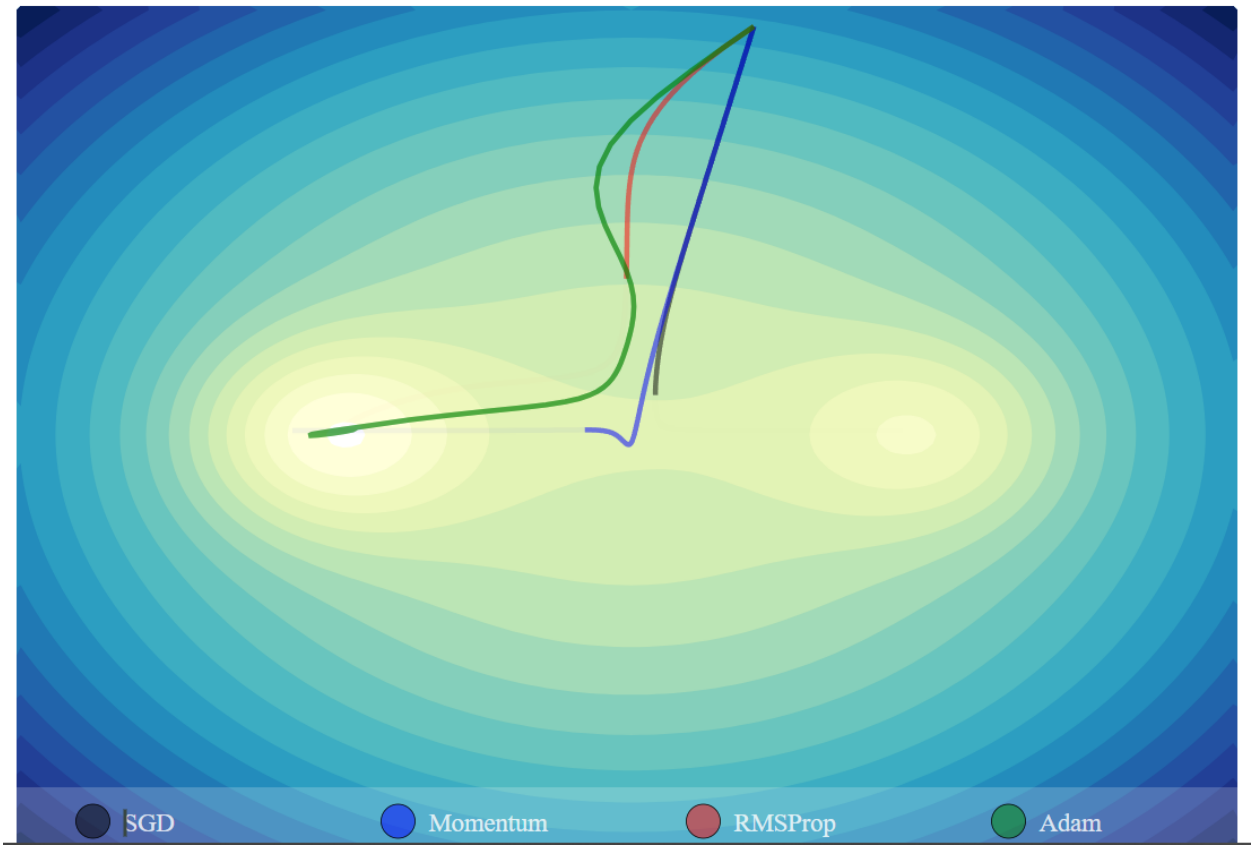
Adam

Algorithm 8.7 The Adam algorithm

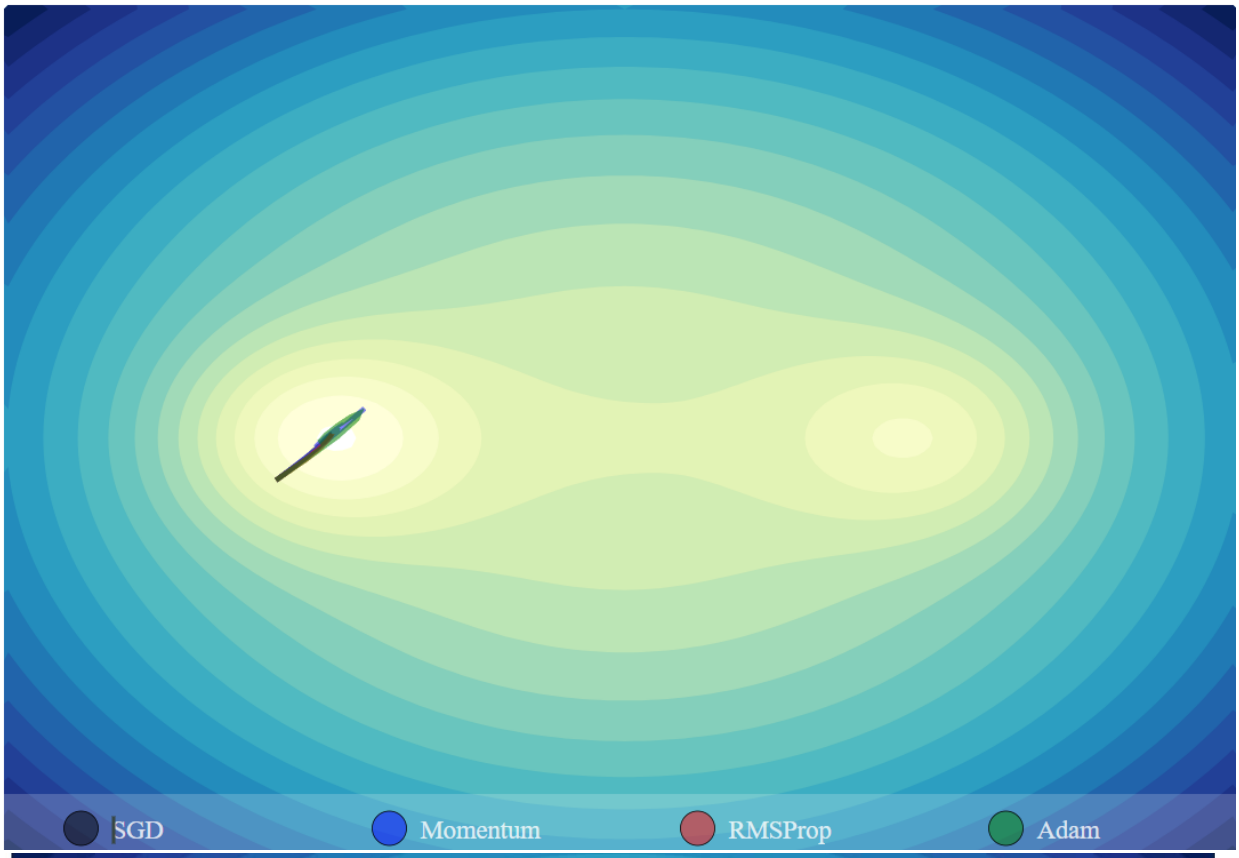
Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  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

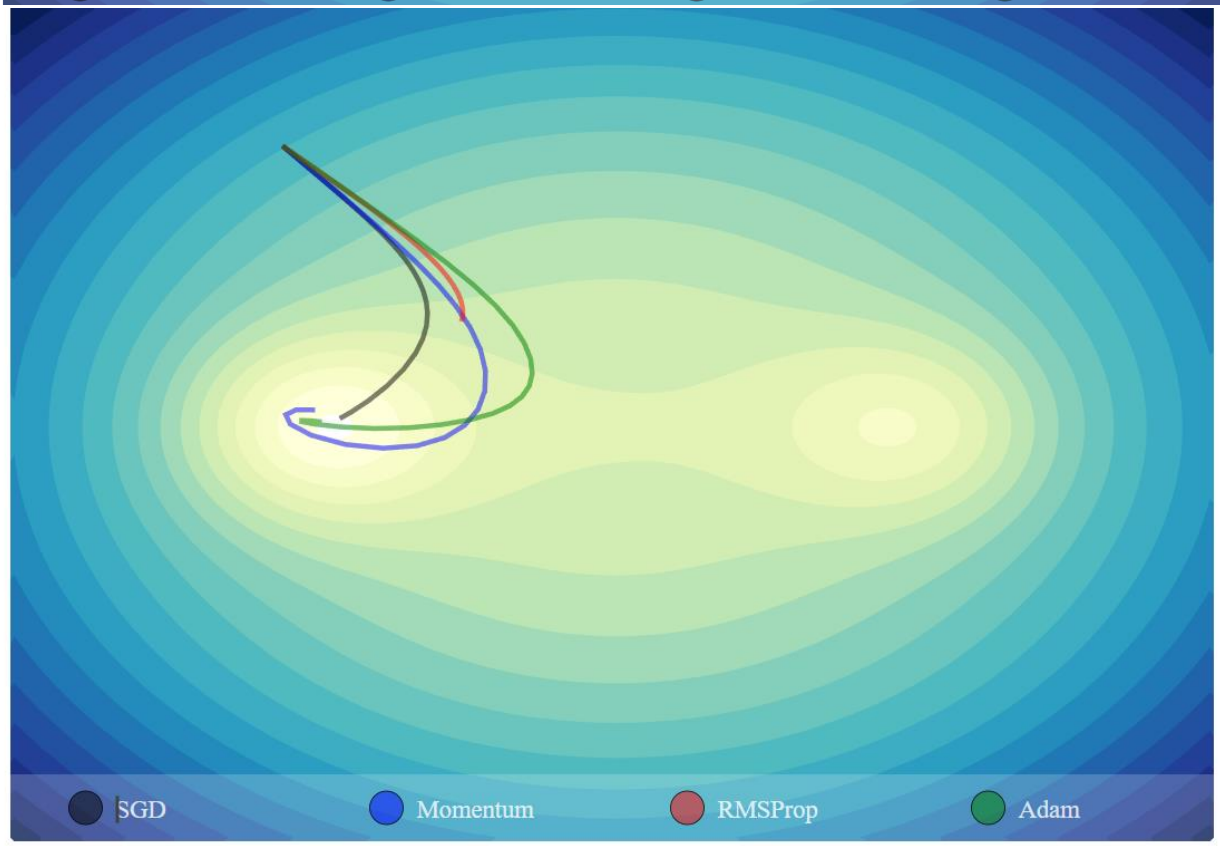
مشاهده میکنیم که در فرمول ها از توان t هم استفاده میشود. در پیمایش های اول حاصل (1-B^t) عددی کوچکتر از یک میباشد و باعث میشود بتوانیم قدم های بزرگی در تغییر پارامتر ها جهت رسیدن به گلوبال اپتیمم برداریم اما کم کم با افزایش t، کل عبارت به سمت یک میرود و کمتر از این نوع boost ها داریم و قدم های تغییرات کوچکتر میباشد.

در تصاویر هم میبنیم زمانی که مسائل پیچیده میشوند، برای مثال در این تصویر نقطه شروع را نزدیک لوکل اپتیمم میگذاریم (دایره سمت راست)، در این حالت Adam علاوه بر اینکه به بهترین جواب میرسد، خیلی سریعتر از سایرین این کار را انجام میدهد:



البته اگر خیلی نزدیک جواب نقطه را انتخاب کنیم، از آنجایی که SGD یکی از گریدی ترین الگوریتم های میان این 4 تا میباشد، مسیر کوتاه تری را طی میکندو به جواب میرسد:


SGD
Momentum
RMSProp
Adam


SGD
Momentum
RMSProp
Adam

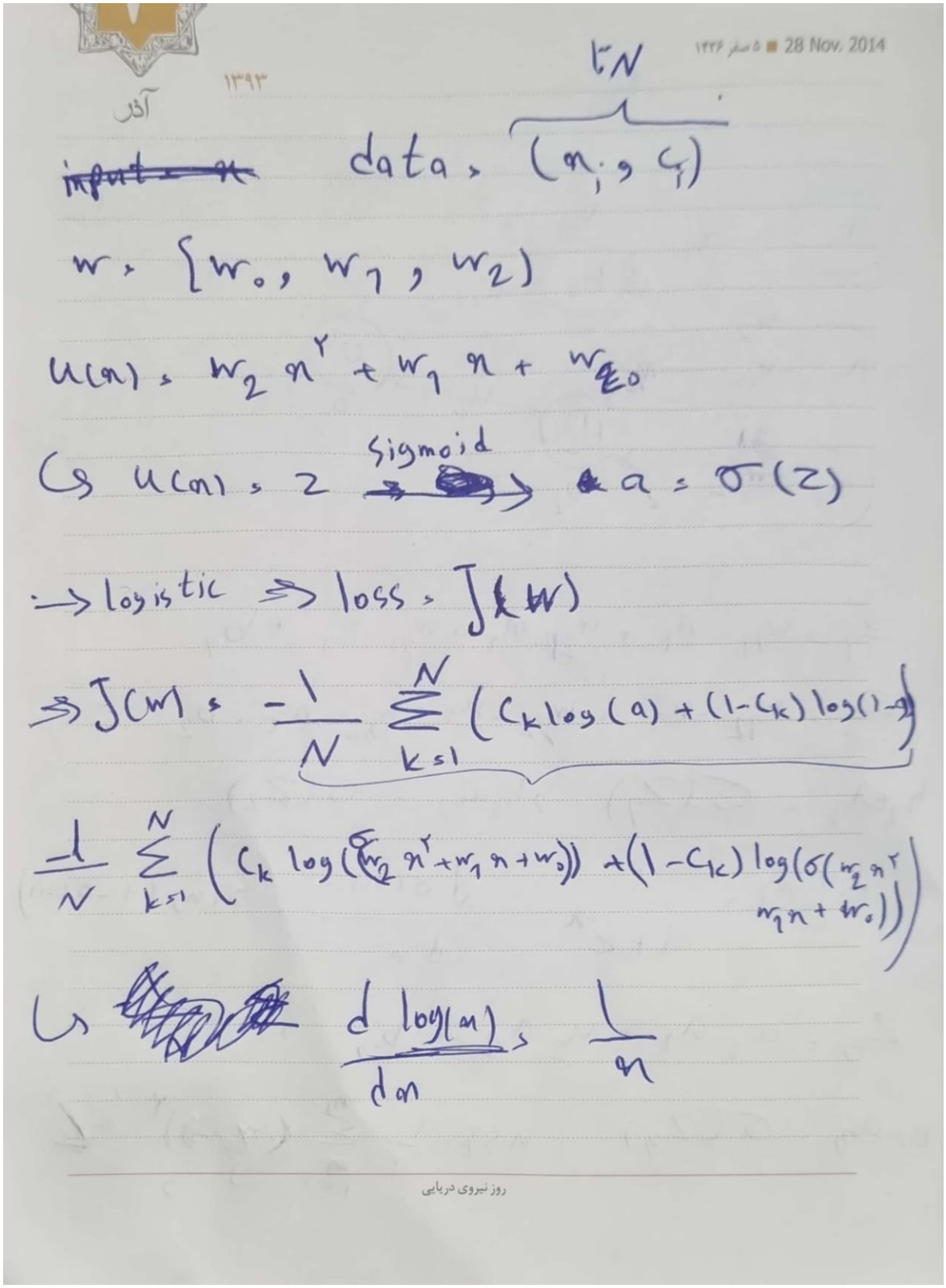
اما واضح است در مسائل پیچیده به SGD نمیتوان به اندازه Adam اعتماد کرد.
پس Adam از پارامتر های بیشتری برخوردار است و مموری بیشتری میخواهد، محبوبیت بسیار بالایی دارد، قدرت جنرالیتی بالایی دارد و در اکثر مواقع زودتر به نتیجه میرسد.

3- یکی از مشکلات رایج در train یک شبکه عصبی vanishing gradients میباشد. مثلا در یکی از حالات، اگر متشق تابع loss نسبت به لایه های انتهایی مقداری بسیار کم باشد، مثلا 0.001، اگر بخواهیم مشتق را نسبت به لایه های اولیه و ابتدایی بدست بیاوریم باید از همین عدد استفاده کنیم (بخاطر استفاده از قائده زنجیره ای)، پس اگر مشتق همان هم کم باشد (یا حتی مشتق قابل توجه ای هم باشد) به دلیل کوچک بودن مشتق لایه قبلی، حاصل بسیار کم و رو به صفر خواهد بود و در نتیجه پارامتر ها، خصوصا پارامتر های لایه های اول دچار تغییر خاصی نمیشوند و هرچه ما iteration بیشتری هم داشته باشیم قرار نیست شاهد تغییر زیاد و مهمی شویم. البته توجه داریم که برخلاف این ایده هم میتواند رخ دهد، یعنی مشتق لایه های جلوتر بسیار زیاد بوده و روی لایه های بعدی هم اثر بگذارد (exploding gradient که آن هم راه حل خود را دارد)

یکی از دلایل هرکدام از این اتفاقات، داشتن تابع اکتیوشن از نوع tanh یا sigmoid است، زیرا این دو روی مقادیر و تاثیرات بزرگ را بزرگتر و کوچک را کوچک تر میکنند، پس یکی از راه حل ها استفاده از توابع فعال ساز relu در لایه های ابتدایی میباشد.
یکی دیگر از راه حل های بسیار ارزشمند، مقدار دهی اولیه درست پارامتر ها است، در بسیاری از موارد شاهد هستیم که ()np.zeros برای مقدار دهی اولیه پارامتر استفاده میشود که این عامل دردسر سازی میتواند باشد و یا اینکه به گونه ای رندوم انتخاب شود که از رنج های امن خارج شود. پس برای اطمینان از اینکه وزن‌ها در محدوده مناسبی برای یادگیری قرار دارند میتوانیم از تکنینک هایی مانند Xavier/Glorotیا He initialization استفاده کنیم.
علاوه بر پارامتر ها، ورودی ها میتوانند سبب مشکل شوند، پس اگر از تکنیک هایی مانند batch normalization و scalarها استفاده کنیم، شانس بهبود افزایش پیدا

میکند.
همچنین اگر شبکه را خیلی deep کرده باشیم، شانس رخ دادن vanishing را بسیار افزایش داده ایم،پس باید مواظب نحوه انتخاب معماری شبکه باشیم و در صورتی که قرار است با دیتا های کم و یا مهم تر از آن، epochهای کمی داشته باشیم، شبکه های بسیار عمیق میتوانند این مشکل را ایجاد کنند.
یکی دیگر از دلایل داشتن دقت پایین، میتواند به دلیل overfitبودن باشد، پس باید مواظب هندل کردن این قضیه نیز باشیم

4- با توجه به اطلاعات داده شده داریم:



پنجشنبه

۶

آذر ۱۳۹۳

27 Nov. 2014 ■ ۴ صفر ۱۴۳۶

$$\rightarrow \frac{d J(w)}{d a} = -\frac{1}{N} \sum_{k=1}^{N} \left(\frac{c_k}{a} + \frac{1-c_k}{1-a} \right)$$

$a = \sigma(z)$

$$\frac{\partial J(w)}{\partial u} = \frac{\partial J}{\partial a} \cdot \frac{\partial a}{\partial u} \rightarrow a(1-a)$$

$$\Rightarrow \quad \hookrightarrow = \left(-\frac{1}{N} \sum_{k=1}^{N} \left(\frac{c_k}{a} + \frac{1-c_k}{1-a} \right) \right) a \cdot (1-a)$$

$$\frac{\partial J(w)}{\partial w_2} = \frac{\partial J}{\partial u} \cdot \frac{\partial u}{\partial w_2} = \frac{\partial J}{\partial u} \cdot x_k^{(2)}$$

مربوط به ۳ ← $\frac{\partial J}{\partial u} \cdots x_k$

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial u} \cdot \frac{\partial u}{\partial w_1} = \frac{\partial J}{\partial u} \cdots$$

$$\frac{\partial J}{\partial w_0} = \frac{\partial J}{\partial u} \cdot \frac{\partial u}{\partial w_0} = \frac{\partial J}{\partial u}$$