

1a (1)

(i) momentum باعث می‌شود که گرادینت فقط به گرادینت mini-batch مرتبط با خروجی وابسته نباشد.

دقیقاً مانند کاری که SGD انجام می‌دهد. در بعضی از موارد اگر ضریب یادگیری خیلی بزرگ باشد ما هیچ وقت به راه حل بهینه نخواهیم رسید و converge نمی‌شویم. در طرف مقابل اگر خیلی هم کوچک باشد

converge خیلی آرام صورت خواهد گرفت. استفاده از momentum باعث می‌شود که ضریب یادگیری

stable باشد و به سرعت تغییر نکند و می‌توانیم ضریب یادگیری را با مقدار بزرگ‌تری مقدار دهی

اولی کنیم تا خروجی خود را تنظیم کند تا converge زودتر اتفاق بیفتد.

(ii) پارامترهای مدل که آپدیت های کوچک‌تری دریافت می‌کنند، آپدیت‌های بزرگ‌تری

دریافت خواهند کرد. شرایط متغیر را در نظر می‌گیریم. پارامترهای مدل که آپدیت‌های بزرگ‌تری

دریافت می‌کنند، ضریب یادگیری بهینه خود را خواهند داشت. پس می‌توانیم ضریب‌های

یادگیری adaptive را به عنوان مثال سازی آپدیت پارامتر در نظر بگیریم.

(b)

(i) هدف مقیاس این است مقدار expected تغییر  $h_{drop}$  را همچون  $h$  قرار دهیم.

پس چون مقدار expected تابع صورت مقابل است:  $d = 1 - P_{drop} \Rightarrow$

$$\gamma = \frac{1}{1 - P_{drop}}$$

(ii) این عمل regularization مورد استفاده است تا از overfitting جلوگیری کند.

در هنگام overfitting مدل در سوابق training اتقار می‌کند.

(۲)

(a)

stack	Buffer	new dependency	Transition
[ROOT]	[I, parseed, this, sentence, correctly]		initial config
[ROOT, I]	[parseed, this, sentence, correctly]		shift
[ROOT, I, parseed]	[this, sentence, correctly]		shift
[ROOT, parseed]	[this, sentence, correctly]	I ← parseed	left-arc
[ROOT, parseed, this]	[sentence, correctly]		shift
[ROOT, parseed, this, sentence]	[correctly]		shift
[ROOT, parseed, sentence]	[correctly]	this ← sentence	left-arc
[ROOT, parseed]	[correctly]	parseed → <del>correctly</del> <sup>sentence</sup>	right-arc
[ROOT, parseed, correctly]	[]		shift
[ROOT, <del>parseed</del> <sup>parseed</sup> ]	[]	parseed → correctly	right-arc
[ROOT]	[]	ROOT → parseed	right-arc

(b) هرگاه می‌تواند با تمام کلمات موجود در متن dependency شکل دهد پس الزاماً

متن n کلمه باشد و هر کلمه با ۱ تا n-۱ کلمه دیگر بتواند رابطه داشته باشد.

$$\text{Sum} = (n-1)^n$$



i) I was heading to a wedding fearing my death. (ف (۲)

head ~ verb phrase  $\frac{1}{2}$   $\frac{1}{2}$  wedding  $\rightarrow$  fearing

استاد  $\frac{1}{2}$   $\frac{1}{2}$  verb phrase attachment error  $\frac{1}{2}$   $\frac{1}{2}$

heading  $\rightarrow$  fearing ~~fearing~~

ii) It makes me want to rush out and rescue people from dilemmas of their own making.

head ~ second conjunct  $\frac{1}{2}$   $\frac{1}{2}$  and  $\leftarrow$  rescue

سید است  $\frac{1}{2}$   $\frac{1}{2}$  coordination attachment error  $\frac{1}{2}$   $\frac{1}{2}$

rush  $\rightarrow$  and

iii) It is on loan from a guy named Joe O'Neil in Midland

, Texas.

head ~ prepositional phrase  $\frac{1}{2}$   $\frac{1}{2}$  named  $\rightarrow$  Midland

prepositional phrase attachment error  $\frac{1}{2}$   $\frac{1}{2}$

guy  $\rightarrow$  Midland

iv) Modifier Attachment error  $\frac{1}{2}$   $\frac{1}{2}$  most  $\leftarrow$  element

most  $\leftarrow$  crucial

در تست داده های کوچکتر که با دستور python run.py -d اجرا شد هر epoch از

train شامل ۴۸ عضو بود که معیار evaluation که UAS بود <sup>که در بهترین حالت</sup> از مقدار ۵۹

آغاز شد و در epoch آخر به مقدار ۹۹٫۹۸ رسید اما در مواردی از ابتدا در محدوده ۲۰ تا ۳۰

به کار خود پایان می داد. در موارد دیگری UAS از epoch ۴ تا ۵ شروع به کاهش می کرد که

نشان دهنده overfitting در train داده ها بود.

در تست داده های اصلی در بهترین حالت UAS از مقدار ۸۲ آغاز و با مقدار ۸۶٫۹

پایین یافت اما در مواردی مقادیر کم هم داشت. میزان loss یا خطا در انتهای کار طبق

پسین بین کمتر از ۰٫۰۸ شد. مقدار دقیق آن ۰٫۰۶۵ شد.

علاوه بر موارد در ادامه گزارش پیوسته می شود.

```

valuating on dev set
445850it [00:00, 27799557.35it/s]
dev UAS: 88.01
new best dev UAS! Saving model.

epoch 6 out of 10
100% | 1848/1848 [01:59<00:00, 15.50it/s]
Average Train Loss: 0.07918166341784197
valuating on dev set
445850it [00:00, 14642527.24it/s]
dev UAS: 87.94

epoch 7 out of 10
100% | 1848/1848 [03:06<00:00, 9.93it/s]
Average Train Loss: 0.07479967679062383
valuating on dev set
445850it [00:00, 13220813.87it/s]
dev UAS: 87.88

epoch 8 out of 10
100% | 1848/1848 [03:03<00:00, 10.08it/s]
Average Train Loss: 0.07146170287638967
valuating on dev set
445850it [00:00, 13544805.37it/s]
dev UAS: 88.65
new best dev UAS! Saving model.

epoch 9 out of 10
100% | 1848/1848 [02:54<00:00, 10.59it/s]
Average Train Loss: 0.06827970123668382
valuating on dev set
445850it [00:00, 13062366.86it/s]
dev UAS: 88.46

epoch 10 out of 10
100% | 1848/1848 [03:00<00:00, 10.27it/s]
Average Train Loss: 0.06530269179533500
valuating on dev set
445850it [00:00, 15097765.18it/s]
dev UAS: 88.40

=====
TESTING
=====
restoring the best model weights found on the dev set
final evaluation on test set
919736it [00:00, 22446520.43it/s]
test UAS: 88.94
Done!

```



```

C:\Users\user\Documents> python3 train.py --data_dir data --model_dir model --num_epochs 10 --batch_size 32 --num_workers 4 --device cuda:0
Epoch 1/10: 100% | 48/48 [00:01:00:00, 28.56it/s]
Average Train Loss: 0.23696634204437316
Evaluating on dev set
25250it [00:00, 12559147.39it/s]
dev UAS: 65.81
New best dev UAS! Saving model.

Epoch 2/10: 100% | 48/48 [00:01:00:00, 29.45it/s]
Average Train Loss: 0.21712318093826374
Evaluating on dev set
25250it [00:00, 11419119.14it/s]
dev UAS: 65.21

Epoch 3/10: 100% | 48/48 [00:01:00:00, 30.01it/s]
Average Train Loss: 0.20415512441347042
Evaluating on dev set
25250it [00:00, 15708416.59it/s]
dev UAS: 67.00
New best dev UAS! Saving model.

Epoch 4/10: 100% | 48/48 [00:01:00:00, 30.02it/s]
Average Train Loss: 0.18842711175481477
Evaluating on dev set
25250it [00:00, 11417630.04it/s]
dev UAS: 68.00
New best dev UAS! Saving model.

Epoch 5/10: 100% | 48/48 [00:01:00:00, 30.06it/s]
Average Train Loss: 0.17620050068944693
Evaluating on dev set
25250it [00:00, 9661895.39it/s]
dev UAS: 68.21
New best dev UAS! Saving model.

Epoch 6/10: 100% | 48/48 [00:01:00:00, 27.80it/s]
Average Train Loss: 0.16345931372294822
Evaluating on dev set
25250it [00:00, 11390645.62it/s]
dev UAS: 69.68
New best dev UAS! Saving model.

cs224n) F:\terme 8\NLP\Homeworks\3\2021\az>

```