دانشکده مهندسی صنایع

Corpus = "The man who passes the sentence should swing the sword."

#	Token	x
0	man	0
1	passes	1
2	sentence	0
3	should	0
4	swing	0
5	sword	0
6	the	0
7	who	0

(1 X 8)

×

W_input		
-0.078	0.018	0.033
0.068	0.170	-0.109
-0.158	-0.081	-0.151
0.150	0.064	0.145
-0.097	-0.055	0.188
0.036	0.071	0.059
0.168	-0.060	-0.058
0.098	0.015	0.096

(8X3)

=

h
0.068
0.170
-0.109

(1X3)

آموختار (۱)

شبکه‌های عصبی در مدلسازی زبانی

واج-پرش

مهدی غضنفری

۱۴۰۲

رمزگشایی از شبکه‌های عصبی[۱] در مدلسازی زبانی واج-پرش[۲]

Demystifying Neural Network in Skip-Gram Language Modeling

Acknowledgement

The materials on this post are based the on two NLP papers, Distributed Representations of Words and Phrases and their Compositionality (Mikolov et al., 2013) and word2vec Parameter Learning Explained (Rong, 2014).

۱. تغییر پارادایم در سبد واژگان[۳]:
گذار از روشهای شمارشی به روشهای پیش‌بینی

تا سال ۲۰۱۳، مدل‌های سنتی برای انجام پردازش زبان طبیعی، **مدل‌های مبتنی بر شمارش** بودند. آنها عمدتاً شامل محاسبه یک ماتریس همزمانی یا هم–رخداد[۴] برای ثبت روابط معنادار بین کلمات هستند. مثلا:

Document 1: "all that glitters is not gold"

Document 2: "all is well that ends well"

---

۱ Demystifying Neural Network in Skip-Gram Language Modeling
۲ https://aegis4048.github.io/demystifying_neural_network_in_skip_gram_language_modeling#eq-1
۳ Word Embedding
۴ Co-Occurence Matrix

جدول ۱ ماتریس هم–رخداد یا هم–وقوعی

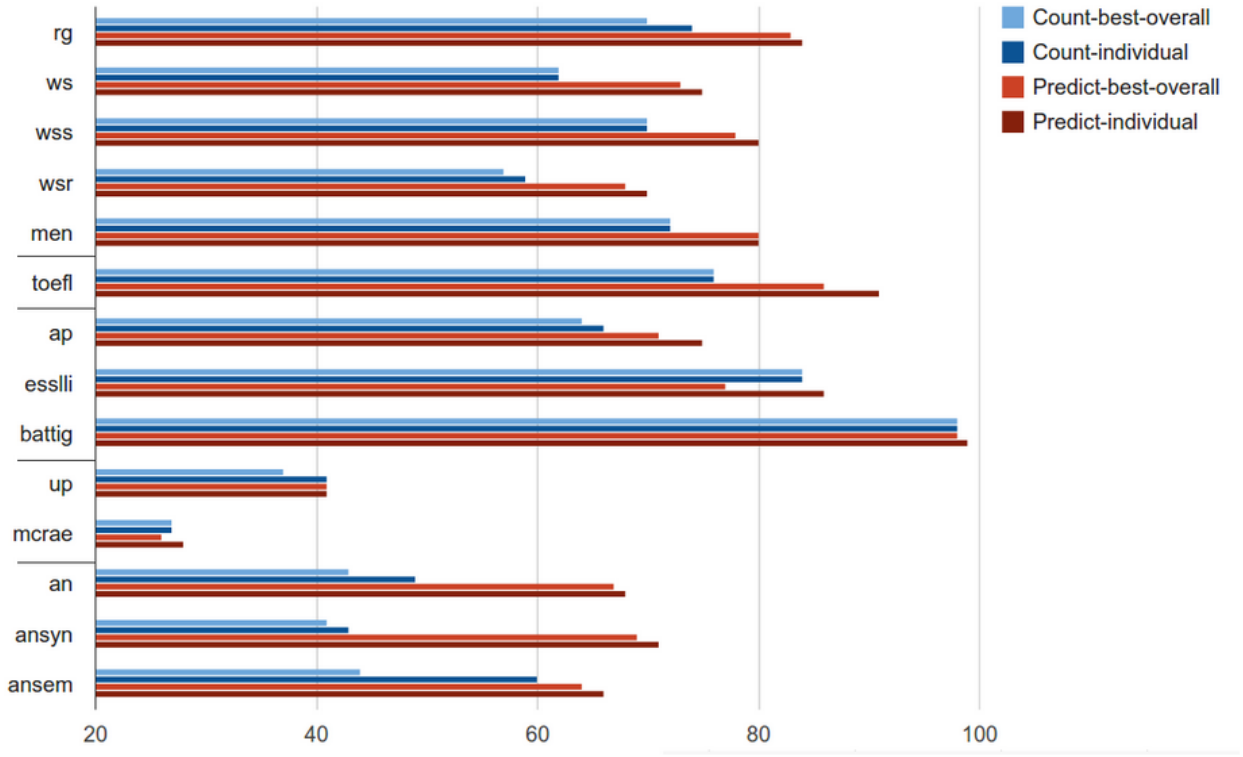
*	START	all	that	glitters	is	not	gold	well	ends	END
START	0	2	0	0	0	0	0	0	0	0
all	2	0	1	0	1	0	0	0	0	0
that	0	1	0	1	0	0	0	1	1	0
glitters	0	0	1	0	1	0	0	0	0	0
is	0	1	0	1	0	1	0	1	0	0
not	0	0	0	0	1	0	1	0	0	0
gold	0	0	0	0	0	1	0	0	0	1
well	0	0	1	0	1	0	0	0	1	1
ends	0	0	1	0	0	0	0	1	0	0
END	0	0	0	0	0	0	1	1	0	0

درک مدل‌سازی زبان مبتنی بر شمارش آسان است: کلمات مرتبط بیشتر از کلمات غیرمرتبط با هم مشاهده می‌شوند (و لذا شمارش می‌شوند). تلاش‌های زیادی برای بهبود عملکرد مدل با استفاده از تجزیه مقادیر منفرد[۱] (SVD)، پنجره–شیبدار[۲] و تجزیه ماتریس غیرمنفی انجام شد (Rohde et al. ms., 2005)، اما این مدل در به دست آوردن روابط پیچیده بین کلمات به خوبی عمل نکرد.

بعد از آن و در سال ۲۰۱۳، پارادایم موجود با کار توماس میکولوف شروع به تغییر کرد؛ زمانی که او تکنیک مدل‌سازی مبتنی بر پیش‌بینی به نام Word2Vec را در مقاله معروف خود تحت عنوان، «بازنمایی‌های توزیع شده کلمات و عبارات و ترکیب‌بندی آنها[۳]» پیشنهاد کرد. برخلاف شمارش هم‌زمانی یا هم–رخدادی کلمات[۴]، این مدل از شبکه‌های عصبی برای یادگیری نمایش هوشمند کلمات در فضای برداری استفاده می‌کند. سپس، مقاله‌ای در سال ۲۰۱۴ به ACL[۵] ارسال شد، «حساب نکنید، پیش‌بینی کنید! مقایسه سیستماتیک بردارهای معنایی شمارش زمینه در مقابل پیش‌بینی زمینه»[۶]، عملکرد مدل‌های مبتنی بر شمارش در مقابل مدل‌های مبتنی بر پیش‌بینی را کمی و مقایسه کرد.

---

۱ Singular Value Decomposition (SVD)
۲ Ramped Window
۳ Distributed Representations of Words and Phrases and their Compositionality
۴ Word Co-Occurrence
۵ Association for Computational Linguistics (ACL)
۶ Don’t count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors



شکل (۱) مقایسه عملکرد مدل‌ها

نوارهای آبی نشاندهنده مدلهای مبتنی بر شمارش (شمارش–پایه) و نوارهای قرمز برای مدلهای مبتنی بر پیش‌بینی (پیش‌بینی–پایه) هستند. بطور خلاصه مدل‌های مبتنی بر پیش‌بینی در کارهای مختلف زبانی با اختلاف زیادی از مدل‌های مبتنی بر شمارش بهتر عمل کردند.

۲ سبد کلمات مبتنی بر پیش‌بینی: Word2Vec Skip-Gram

یکی از مدلهای زبان مبتنی بر پیش‌بینی معرفی شده توسط میکولوف روش **واج–پرش** یا Skip-Gram است:

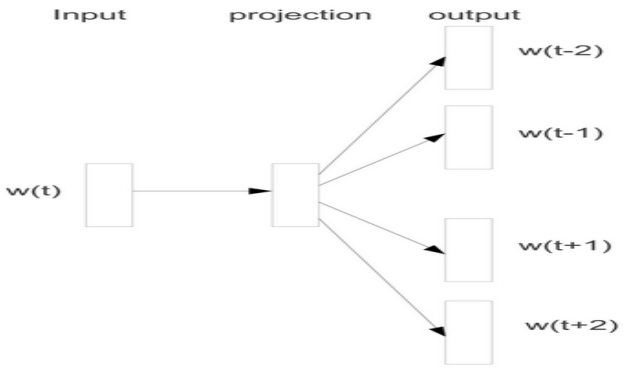
شکل (۲) نموداری است که در مقاله اصلی Word2Vec ارائه شده است. این شکل اساساً توضیح می‌دهد که مدل از یک شبکه عصبی با یک لایه پنهان (لایه بازتاب یا تصویر[۱]) برای پیش‌بینی صحیح کلمات متن زمینه همچون

$$w(t-2)،\ w(t-1)،\ w(t+1)،\ w(t+2)$$

بر اساس یک کلمه ورودی همچون $w(t)$ استفاده می‌کند. به عبارت دیگر، مدل تلاش می‌کند تا احتمال مشاهده هر چهار **کلمه زمینه** را با هم، با توجه به یک **کلمه مرکزی**، به حداکثر برساند (از یک به چند). از نظر ریاضی می‌توان این رویه را

[۱] Projection

توسط معادله (۱) نشان داد. هدف آموزش، یادگیری نمایش‌های برداری کلمه است که در پیش‌بینی کلمات نزدیک خوب عمل می‌کنند.



شکل (۲) معماری مدل اصلی روش واج–پرش (Skip-Gram)

نکته: CBOW و Skip-Gram

دو مدل برای Word2Vec وجود دارد: کوله واژگان پیوسته[۱] (CBOW) و روش واج–پرش[۲]. در حالی که مدل واج–پرش کلمات متنی را با استفاده از یک کلمه مرکزی داده شده پیش‌بینی می‌کند، مدل CBOW یک کلمه مرکزی را بر مبنای کلمات متنی داده شده پیش‌بینی می‌کند. به گفته میکولوف:

واج–پرش: با مقدار کمی از داده‌های آموزشی به خوبی کار می‌کند، حتی کلمات یا عبارات کمیاب را به خوبی نشان می‌دهد.

CBOW: چندین برابر سریعتر از واج–پرش برای آموزش بوده، و دقتش برای کلمات متداول اندکی بهتر است.

مدل واج–پرش به دلیل توانایی آن در پیش‌بینی کلمات نادر، اغلب اوقات انتخاب بهتری است، اما این به قیمت افزایش هزینه محاسباتی است. اگر زمان آموزش یک نگرانی بزرگ است، و داده‌های کافی برای غلبه بر مشکل پیش‌بینی کلمات نادر دارید، مدل CBOW ممکن است انتخاب مناسب‌تری باشد.

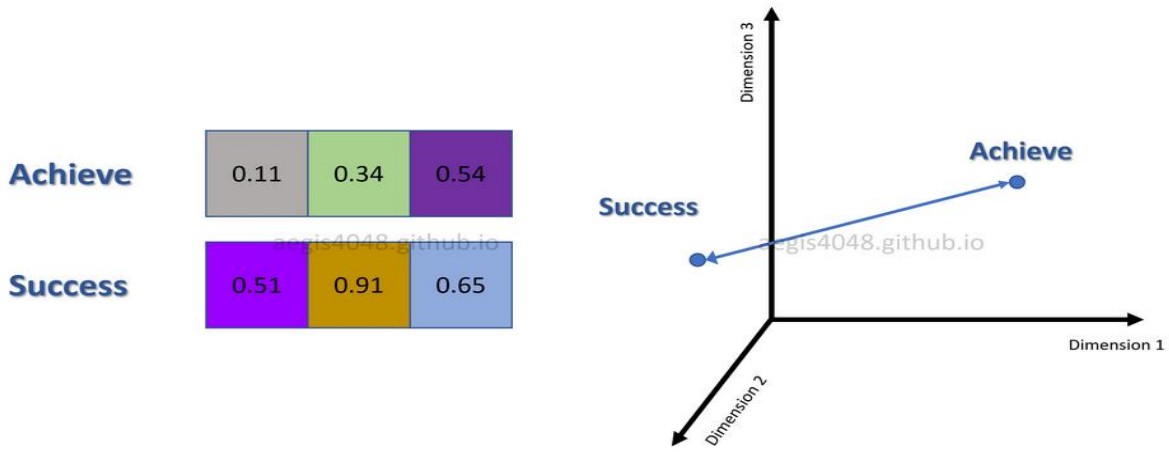
[۱] Continues Bag of Words (CBOW)
[۲] Skip-Gram

۲,۱ چرا کلمات زمینه متنی را پیش بینی کنیم؟

یک سوال طبیعی این است که چرا کلمات زمینه متن را پیش‌بینی می‌کنیم؟ باید درک کرد که هدف نهایی مدل واج–پرش پیش‌بینی کلمات زمینه متنی[۱] نیست، بلکه یادگیری هوشمند **نمایش برداری کلمات** است. پیش‌بینی کلمات متنی به دلیل ساختار شبکه عصبی واج–پرش به ناچار منجر به نمایش مناسبی برای بردار کلمات می‌شود. کار اصلی شبکه عصبی، فقط بهینه‌سازی ماتریس‌های وزن θ و پیش‌بینی صحیح خروجی است. در روش واج–پرش مربوط به Word2Vec، ماتریس‌های وزن در واقع نمایش برداری کلمات هستند. بنابراین، بهینه‌سازی ماتریس وزن، معادل نمایش مناسب برداری کلمات است.

۲,۲ کاربرد نمایش برداری کلمات چیست؟

در Word2Vec، کلمات به صورت بردار نمایش داده می‌شوند و کلمات مرتبط به یکدیگر، در یک فضای برداری نزدیک به هم قرار می‌گیرند. از نظر ریاضی، این بدان معنی است که فاصله برداری بین کلمات مرتبط کوچکتر از فاصله برداری بین کلمات غیرمرتبط است.



شکل (۳) فاصله برداری بین دو کلمه

به عنوان مثال در شکل (۳)، همبستگی[۲] بین "Success" و "Achieve" را می‌توان با محاسبه فاصله برداری بین آنها کمی‌سازی کرد (**توجه:** به منظور هدف آموزشی، بردارهای کلمه سه بعدی فرض شده‌اند، زیرا بردارها با ابعاد بالاتر را نمی‌توان

[۱] Context Words
[۲] Correlation

مصورسازی کرد. همچنین، فاصله بکار برده شده در این شکل، بصورت اقلیدسی است، اما در زندگی واقعی، از فاصله کسینوسی برای ارزیابی همبستگی‌های برداری استفاده می‌کنیم).

یکی از کاربردهای جالب **نمایش برداری کلمات** این است که می‌توان از آن برای قیاس[۱] استفاده کرد. بیایید بردارهای کلمه زیر را برای "Germany"، "Capital" و "Berlin" فرض کنیم.

$$vec(Germany) = [۱.۲۲ \quad ۰.۳۴ \quad -۳.۸۲]$$

$$vec(capital) = [۳.۰۲ \quad -۰.۹۳ \quad ۱.۸۲]$$

$$vec(Berlin) = [۴.۰۹ \quad -۰.۵۸ \quad ۲.۰۱]$$

برای پی بردن به پایتخت آلمان می‌توان بردار کلمه پایتخت را به بردار کلمه آلمان اضافه کرد.

$$vec(Germany) + vec(capital) = [۱.۲۲ \quad ۰.۳۴ \quad -۳.۸۲] + [۳.۰۲ \quad -۰.۹۳ \quad ۱.۸۲]$$
$$= [۴.۲۴ \quad -۰.۵۹ \quad -۲.۰۰]$$

از آنجایی که مجموع بردارهای کلمه "آلمان" و "پایتخت" مشابه بردار کلمه "برلین" است، مدل ممکن است به این نتیجه برسد که پایتخت آلمان برلین است.

$$[۴.۲۴ \quad -۰.۵۹ \quad -۲.۰۰] \cong [۴.۰۹ \quad -۰.۵۸ \quad ۲.۰۱]$$

$$vec(Germany) + vec(capital) \cong vec(Berlin)$$

توجه: قیاس همیشه کار نمی‌کنند

همه قیاس‌ها را نمی‌توان به این شکل حل کرد. تصویر فوق مانند یک جادو کار می‌کند ، اما قیاس مشکلات بسیاری دارد که با Word2Vec قابل حل نیست. تصویر فوق را به عنوان فقط یک مورد استفاده از Word2Vec فرض کنید.

۳ استخراج تابع هزینه

مدل واج–پرش به دنبال بهینه‌سازی ماتریس وزن (تعبیه یا سبد) کلمات با پیش‌بینی صحیح کلمات متنی با توجه به یک کلمه مرکزی است. به عبارت دیگر، مدل می‌خواهد احتمال پیش‌بینی صحیح همه کلمات متنی را در یک زمان، با توجه به یک کلمه مرکزی، به حداکثر برساند. به حداکثر رساندن احتمال پیش‌بینی کلمات متنی منجر به بهینه‌سازی ماتریس وزن (θ) می‌شود که به بهترین شکل کلمات را در یک فضای برداری نشان می‌دهد. θ ترکیبی از ماتریس های وزن ورودی و خروجی $[W_{input} \quad W_{output}]$ است.

[۱] Analogy Task

این مقادیر، ورودی تابع هزینه (*J*) بوده و به عنوان متغیر، بهینه می‌شود. از نظر ریاضی می‌توان آن را به صورت زیر بیان کرد:

$$\underset{\theta}{arg\max}\ p(w_1, w_2, \dots, w_C | w_{center};\ \theta) \qquad \text{معادله (۱)}$$

که در آن ***C* اندازه پنجره**، و ***w*** یک بردار کلمه است (که می‌تواند یک کلمه متنی یا یک کلمه مرکزی باشد). از آمار می‌دانیم، احتمال ***A*** با فرض معلوم بودن ***B*** به صورت ***P(A|B)*** بیان می‌شود. سپس، لگاریتم از معادله (۱) گرفته می‌شود تا عمل مشتق‌گیری را ساده کند.

$$\underset{\theta}{arg\max} \log p(w_1, w_2, \dots, w_C | w_{center};\ \theta) \qquad \text{معادله (۲)}$$

نکته: چرا لگاریتم بگیریم؟

در یادگیری ماشین، استفاده از لگاریتم تابع هدف برای ساده کردن مشتقات، یک روش معمول است. برای مثال، یک طبقه‌بند رگرسیون چند جمله‌ای به نام بیشینه–نرم[۱] دارای تابع احتمال زیر است:

$$p(x_i) = \frac{e^{x_i}}{\sum_{j=1} e^{x_j}}$$

لگاریتم گرفتن از این تابع، آنرا ساده می‌کند:

$$\log p(x_i) = x_i - \log \sum_{j=1} e^{x_j}$$

بسته به مدل، آرگومان (x_i) که وارد تابع احتمال (*p*) می‌شود، می‌تواند پیچیده باشد و ساده‌سازی تابع بیشینه–نرم اصلی به گرفتن مشتقات در آینده کمک می‌کند.

گرفتن log بر وزن‌های بهینه شده (θ) تاثیر نمی‌گذارد، زیرا لگاریتم یک تابع صعودی یکنواخت است. این بدان معناست که افزایش مقدار در محور ***x*** منجر به افزایش مقدار در محور ***y*** می‌شود. این مهم است زیرا تضمین می‌کند که حداکثر مقدار تابع احتمال اصلی در همان نقطه با تابع احتمال ورود به سیستم اتفاق می‌افتد. از این رو:

$$\underset{\theta}{arg\max}\ p(x_i) = \underset{\theta}{arg\max} \log p(x_i)$$

در واج–پرش از تابع بیشینه–نرم برای طبقه بندی کلمات متنی استفاده می‌شود. بیشینه–نرم در واج–پرش معادله زیر را دارد:

[۱] Softmax

$$p(w_{context}|w_{center};\ \theta) = \frac{\exp\ (W_{output_{(context)}}.h)}{\sum_{i=1}^{V} \exp\ (W_{output_i}.h)}$$ معادله (۳)

$W_{output_{(context)}}$ یک بردار برای یک **کلمه متنی** از ماتریس تعبیه (سبد کلمات) خروجی است، و h بردار کلمه لایه پنهان (تصویر) برای یک **کلمه مرکزی** است. سپس تابع بیشینه–نرم به معادله ۲ متصل می‌شود تا یک تابع هدف جدید به دست آید که احتمال مشاهده تمام کلمات متن C را با توجه به یک کلمه مرکزی به حداکثر می‌رساند:

$$arg\max_{\theta} \log \prod_{c=1}^{C} \frac{\exp\ (W_{output_{(context)}}.h)}{\sum_{i=1}^{V} \exp\ (W_{output_i}.h)}$$ معادله (۴)

نکته: حاصل ضرب احتمالات

در آمار، احتمال مشاهده C رویداد به طور همزمان با حاصل ضرب احتمال هر رویداد محاسبه می‌شود.

$$p(x_1, x_2, \dots, x_C) = p(x_1) \times p(x_2) \times \dots \times p(x_C)$$

این عبارت را می‌توان با یک علامت حاصل ضرب خلاصه کرد:

$$p(x_1, x_2, \dots, x_C) = \prod_{c=1}^{C} p\ (x_c)$$

عرف معادله نویسی در یادگیری ماشین، به حداقل رساندن (نه به حداکثر رساندن) تابع هزینه است. برای پایبندی به این عرف، یک علامت منفی به معادله (۴) اضافه می‌کنیم. این کار را می‌توان انجام داد زیرا به حداقل رساندن یک تابع درستنمایی[۱] منفی معادل به حداکثر رساندن یک تابع درست نمایی مثبت است. بنابراین، تابع هزینه‌ای که می‌خواهیم به حداقل برسانیم به صورت زیر نوشته می‌شود:

$$J(\theta; w^{(t)}) = -\log \prod_{c=1}^{C} \frac{\exp\ (W_{output_{(context)}}.h)}{\sum_{i=1}^{V} \exp\ (W_{output_i}.h)}$$ معادله (۵)

که در آن c شاخص کلمات زمینه متنی در اطراف کلمه مرکزی (w_t) است. t نیز شاخص کلمه مرکزی در مجموعه‌ای به اندازه T است. با استفاده از ویژگی لگاریتم، می‌توان آن را به شکل زیر نوشت:

[۱] Log-Likelihood

$$J(\theta; w^{(t)}) = -\sum_{c=1}^{C} log \frac{\exp\left(W_{output_{(context)}}.h\right)}{\sum_{i=1}^{V} \exp\left(W_{output_i}.h\right)}$$ معادله (۶)

گرفتن یک لگاریتم از تابع بیشینه–نرم به ما امکان این را می‌دهد که عبارت را به اشکال ساده‌تر بازنویسی کنیم، زیرا می‌توانیم کسر را به جمع‌کننده صورت و مخرج تقسیم کنیم:

$$J(\theta; w^{(t)}) = -\sum_{c=1}^{C} (W_{output_{(c)}}.h) + C.log \sum_{i=1}^{V} \exp\left(W_{output_{(i)}}.h\right)$$ معادله (۷)

مقالات مختلف از نمادهای متفاوتی برای تابع هزینه استفاده می‌کنند. برای پایبندی به نماد استفاده شده در مقاله اصلی *Word2Vec*، می‌توان برخی از نمادها را در معادله (۷) تغییر داد. با این حال، همه آنها معادل هم هستند:

$$J(\theta; w^{(t)}) = -\sum_{-c \le j \le c, j \neq 0} \log p\left(w_{t+j} | w_t; \theta\right)$$ معادله (۸)

توجه داشته باشید که معادله (۷) و (۸) معادل هم هستند. هر دوی آنها **کاهش گرادیان تصادفی** را فرض می‌کنند، به این معنی که برای هر نمونه آموزشی (کلمه مرکزی) $w(t)$ در پیکره‌ای با اندازه T، یک به‌روزرسانی برای ماتریس وزن (θ) انجام می‌شود. تابع هزینه بیان شده در مقاله معادله ۹ **شیب نزولی دسته‌ای**[۱] را نشان می‌دهد، به این معنی که تنها یک به‌روز رسانی برای همه نمونه‌های آموزشی T انجام می‌شود:

$$J(\theta) = -\frac{1}{T}\sum_{T=1}^{T} \sum_{-c \le j \le c, j \neq 0} \log p\left(w_{t+j} | w_t; \theta\right)$$ معادله (۹)

با این حال، در *Word2Vec*، به دلیل هزینه محاسباتی بالا، تقریباً هرگز از شیب نزولی دسته‌ای استفاده نمی‌شود.

٤ اندازه پنجرهٔ واج-پرش

[۱] Batch Gradient Descent

رگرسیون بیشینه–نرم (یا رگرسیون لجستیک چند جمله‌ای) تعمیم رگرسیون لجستیک به مواردی است که می‌خواهیم چندین کلاس (دسته) را مدیریت کنیم. شکل کلی رگرسیون بیشینه–نرم به صورت زیر است:

$$J(\theta) = -\frac{1}{T}\sum_{T=1}^{T}\sum_{k=1}^{K} log\frac{\exp\left(\theta^{(k)\top}x^{(t)}\right)}{\sum_{i=1}^{K}\exp\left(\theta^{(k)\top}x^{(t)}\right)}$$ معادله (۱۰)

که در آن T تعداد نمونه های آموزشی و K تعداد برچسب‌هایی است که باید طبقه‌بندی شوند. در کاربردهای پردازش زبان طبیعی، $K=V$، چون V واژه منحصر به فرد وجود دارد که باید در یک فضای برداری طبقه‌بندی شوند. V به راحتی می‌تواند از ده‌ها هزار عبور کند. روش **واج–پرش** این را کمی تغییر می‌دهد و K را با متغیری به نام **اندازه پنجره** C جایگزین می‌کند. اندازه پنجره، فراپارامتر یا ابرپارامتری است که معمولا دامنه آن بصورت تجربی در محدوده [۱،۱۰] است. می‌دانیم که روش واج–پرش مدلی است که سعی می‌کند کلمات همسایهٔ یک کلمه مرکزی را پیش‌بینی کند. لازم نیست تمام واژگان V را که ممکن است ۱۰۰ کلمه یا بیشتر با آن فاصله داشته باشد، پیش‌بینی کند، بلکه فقط چند کلمه، ۱ تا ۱۰ کلمه همسایه در پیکره ادبی را پیش‌بینی می‌کند چرا که کلماتی که دور هستند اطلاعات کمتری درباره یکدیگر دارند. بنابراین، شکل اقتباس شده از معادله رگرسیون بیشینه–نرم برای بیشینه–نرم به صورت زیر بازنویسی می‌شود:

$$J(\theta) = -\frac{1}{T}\sum_{T=1}^{T}\sum_{-c\le j\le c, j\neq 0} log\frac{\exp\left(\theta^{(t+j)\top}x^{(t)}\right)}{\sum_{i=1}^{K}\exp\left(\theta^{(i)\top}x^{(t)}\right)}$$ معادله (۱۱)

این معادل رابطه (۹) است. توجه داشته باشید که K در مخرج هنوز برابر با V است، زیرا مخرج به عنوان یک عامل نرمالیزه کردن عمل می‌کند، همانطور که در زیر توضیح داده شده است. با این حال، اندازه K در مخرج هنوز می‌تواند با استفاده از نمونه گیری منفی[۱] به اندازه کوچکتر کاهش یابد.

۵ ساختار شبکه عصبی واج-پرش

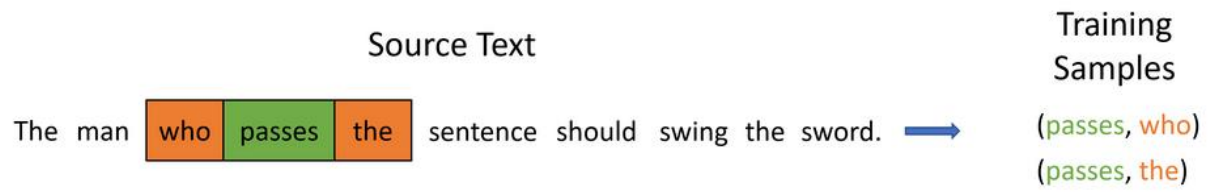
چگونه از شبکه عصبی برای به حداقل رساندن تابع هزینه توضیح داده شده در معادله ۱۱ استفاده می شود؟ برای به دست آوردن بینشی در مورد رابطه میان آنها باید ساختار مدل واج–پرش را بررسی کرد.

برای مثال، جمله زیر را در نظر بگیرید: ۱۰ کلمه وجود دارد ($T=10$) و ۸ کلمه منحصر بفرد ($V=8$)

[۱] Negative Sampling

"The man who passes the sentence should swing the sword."

البته در زندگی واقعی، پیکره واژگان بسیار بزرگتر از یک جمله است. ما از `window=1` استفاده می‌کنیم و فرض می‌کنیم که "*pass*" کلمه **مرکزی** فعلی بوده و کلمات زمینه **متنی** "*who*" و "*the*" هستند.



شکل (٤) پنجره آموزش

به دلیل هدف آموزشی، یک شبکه عصبی سه بعدی ساخته خواهد شد. در *genism* این موضوع را می‌توان با تنظیم `size=3` پیاده‌سازی کرد. این مقداردهی باعث N=3 می‌شود. توجه داشته باشید که `size` نیز یک ابرپارامتر است که می‌تواند به صورت تجربی تنظیم شود. در زندگی واقعی، یک مدل *Word2Vec* معمولی دارای ۲۰۰–۶۰۰ نورون است.

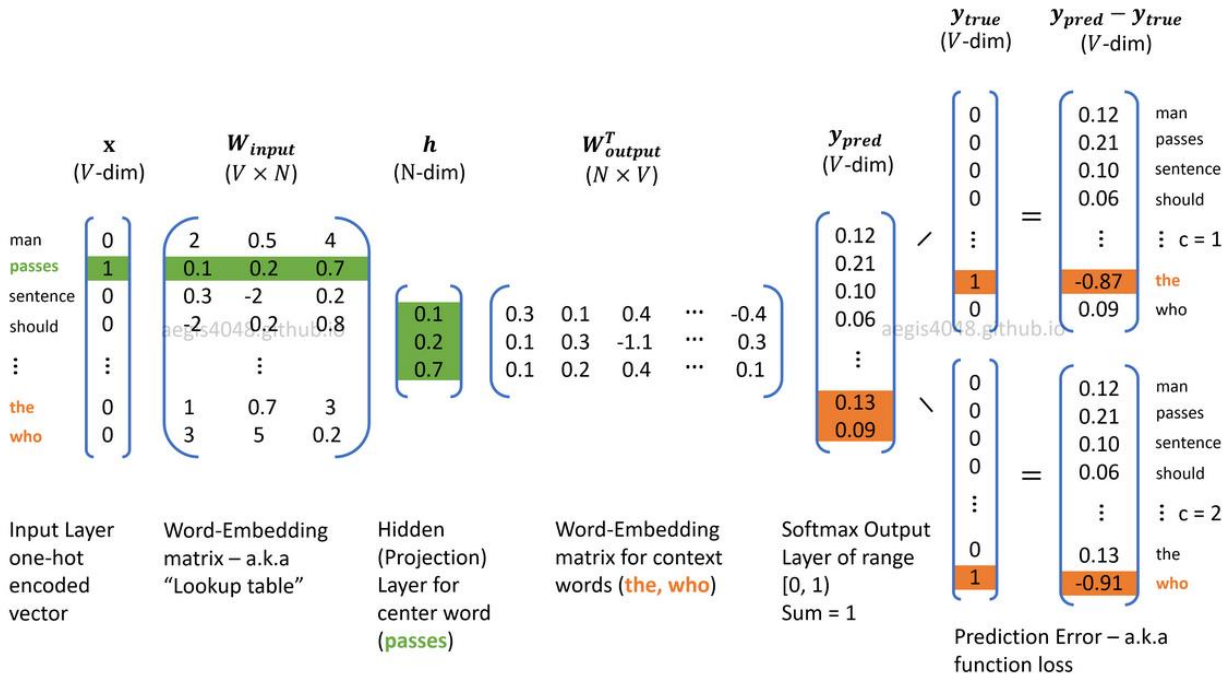
```
from gensim.models import Word2Vec

model = Word2Vec(corpus, size=3, window=1)
```

این بدان معناست که ماتریس وزن ورودی (W_{input}) دارای اندازه ۸×۳ و ماتریس وزن خروجی (W_{output}^{T}) دارای اندازه ۸×۳ خواهد بود. به یاد داشته باشید که مجموعه پیکره واژگان ما، یعنی:

"The man who passes the sentence should swing the sword."

دارای ۸ واژه منحصر به فرد است ($V=8$).



شکل (۵) ساختار مدل واج–پرش کلمه مرکزی فعلی "$passes$" است.

۵٫۱ آموزش: انتشار رو به جلو

ماتریس‌های تعبیه یا سبد کلمات (W_{input}، W_{output}) در روش واج–پرش از طریق انتشار به جلو و عقب، بهینه می‌شوند. برای هر تکرار انتشار رو به جلو و پس‌انتشار رو عقب، مدل یاد می‌گیرد که خطای پیش‌بینی را با بهینه‌سازی ماتریس وزن (θ) کاهش دهد، بنابراین ماتریس‌های تعبیه با کیفیت‌تری را به دست می‌آورد که روابط بین کلمات را بهتر نشان می‌دهد.

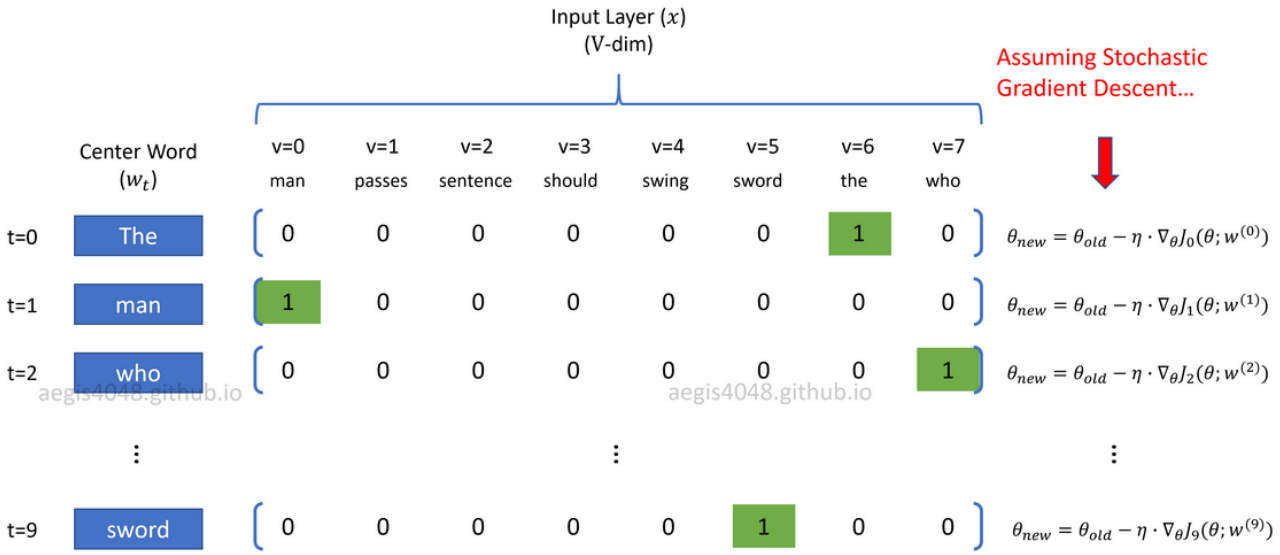
انتشار رو به جلو شامل به دست آوردن توزیع احتمال کلمات (y_{pred} در شکل ۵) به شرط یک کلمه مرکزی است، و انتشار رو به عقب شامل محاسبه خطای پیش‌بینی و به‌روزرسانی ماتریس‌های وزن (تعبیه) برای به حداقل رساندن خطای پیش‌بینی است.

۵٫۱٫۱ لایه ورودی (x)

لایه ورودی یک بردار V بعدی بصورت تک–نمود یا یکدانه[۱] رمزگذاری شده است. هر عنصر در بردار، به جز عنصری که با **کلمه مرکزی** (ورودی) مطابقت دارد، برابر صفر است. بردار ورودی در ماتریس وزن ورودی (W_{input}) به اندازه $V \times N$ ضرب می‌شود و یک لایه پنهان (تصویر) (h) از بردار $N\text{-}dim$ به دست می‌دهد. از آنجایی که لایه ورودی بصورت تک–

[۱] One-Hot

نمود کدگذاری شده است، باعث می‌شود ماتریس وزن ورودی (W_{input}) مانند یک جدول جستجو برای **کلمه مرکزی** رفتار کند. با فرض عدد دوره ۱ و **کاهش گرادیان تصادفی**[۱]، بردار ورودی برای هر کلمه در سند T بار به شبکه تزریق می‌شود و ماتریس وزن (θ) را T بار به‌روز می‌کند تا از نمونه های آموزشی یاد بگیرد. استخراج به‌روز رسانی تصادفی معادلات در زیر توضیح داده شده است.



شکل (٦) نمایش بردار ورودی بصورت تک–نمود کد شده و به‌روزرسانی پارامتر

نکته: کاهش گرادیان تصادفی

هدف هر مدل یادگیری ماشینی یافتن مقادیر بهینه یک ماتریس وزن (θ) برای به حداقل رساندن خطای پیش‌بینی است. یک معادله به‌روزرسانی کلی برای ماتریس وزن به صورت زیر است:

$$\theta_{new} = \theta_{old} - \eta \nabla_{J(\theta)}$$

η نرخ یادگیری است، $\nabla_{J(\theta)}$ گرادیان برای ماتریس وزن است، و $J(\theta)$ تابع هزینه است که برای هر مدل اشکال متفاوتی دارد. تابع هزینه برای مدل واج–پرش که در مقاله اصلی $Word2Vec$ پیشنهاد شده، معادله زیر است:

[۱] Stochastic Gradient Descent (SGD)

$$J(\theta) = -\frac{1}{T}\sum_{T=1}^{T}\sum_{-c\le j\le c, j\neq 0} \log p\left(w_{t+j}|w_t;\theta\right)$$

در اینجا، چیزی که باعث دردسر ما می‌شود عبارت $\frac{1}{T}\sum_{T=1}^{T}$ است، زیرا T می‌تواند در بسیاری از موارد کاربرد پردازش زبان طبیعی بزرگتر از میلیاردها یا بیشتر باشد. اساساً این عبارت به ما می‌گوید که میلیاردها تکرار باید محاسبه شود تا فقط یک به‌روزرسانی به ماتریس وزن (θ) انجام شود. به منظور کاهش این بار محاسباتی، نویسنده مقاله بیان می‌کند که کاهش گرادیان تصادفی (SGD) برای بهینه‌سازی پارامتر استفاده شده است. SGD عبارت $\frac{1}{T}\sum_{T=1}^{T}$ را از تابع هزینه حذف می‌کند و به‌روزرسانی پارامتر را برای هر مثال آموزشی، $w^{(t)}$ انجام می‌دهد:

$$J\left(\theta; w^{(t)}\right) = -\sum_{-c\le j\le c, j\neq 0} \log p\left(w_{t+j}|w_t;\theta\right)$$

سپس، معادله جدید به روزرسانی پارامتر برای SGD به عبارت زیر تبدیل می‌شود:

$$\theta_{new} = \theta_{old} - \eta\nabla_{J(\theta;w^{(t)})}$$

کاهش گرادیان اصلی، ۱ پارامتر را برای T نمونه آموزشی به‌روزرسانی می‌کند، اما معادله به‌روزرسانی جدید با استفاده از SGD، پارامتر T را برای T نمونه‌های آموزشی به‌روزرسانی می‌کند. با این حال، این به قیمت نوسان (یا واریانس) بیشتر در به حداقل رساندن خطای پیش‌بینی است.

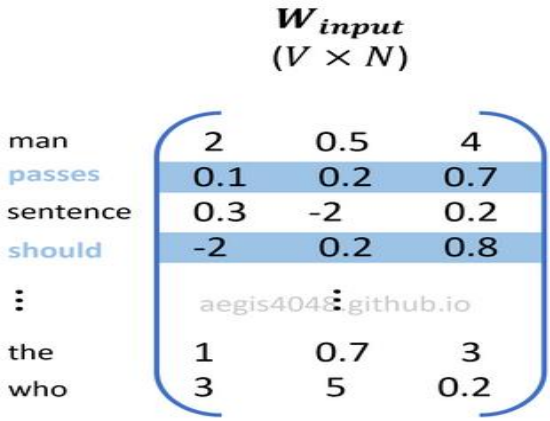
۵،۱،۲ ماتریس وزن ورودی و خروجی (W_{input}، W_{output})

چرا مدل واج–پرش تلاش می‌کند تا **کلمات زمینه** متنی[۱] را با یک **کلمه مرکزی** پیش‌بینی کند؟ چگونه پیش‌بینی کلمات زمینه متنی به کمی‌سازی کلمات و نمایش آنها در فضای برداری کمک می‌کند؟ در واقع، هدف نهایی مدل، پیش‌بینی کلمات زمینه متنی نیست، بلکه ساخت ماتریس‌های تعبیه یا سبد کلمه است (W_{input}، W_{output}) که بهترین رابطه را بین کلمات در یک فضای برداری نشان دهد. روش واج–پرش با استفاده از یک شبکه عصبی به این امر دست می‌یابد: ماتریس‌های وزن

[۱] Context Words

(تعبیه) را با تنظیم ماتریس وزن برای به حداقل رساندن خطای پیش‌بینی، بهینه می‌کند ($y_{pred} - y_{true}$). هنگامی که درک کنید که ماتریس تعبیه چگونه مانند یک جدول جستجو[۱] عمل می‌کند، این امر منطقی‌تر خواهد شد.

هر ردیف در یک ماتریس تعبیه کلمات، یک **بردار-کلمه** برای هر کلمه است. ماتریس تعبیه یا سبد کلمات زیر را به عنوان W_{input} در نظر بگیرید.



شکل (۷) ماتریس تعبیه یا سبد کلمات، W_{input}

واژه های مورد بررسی ما «*passes*» و «*should*» هستند. "*passes*" دارای بردار کلمه

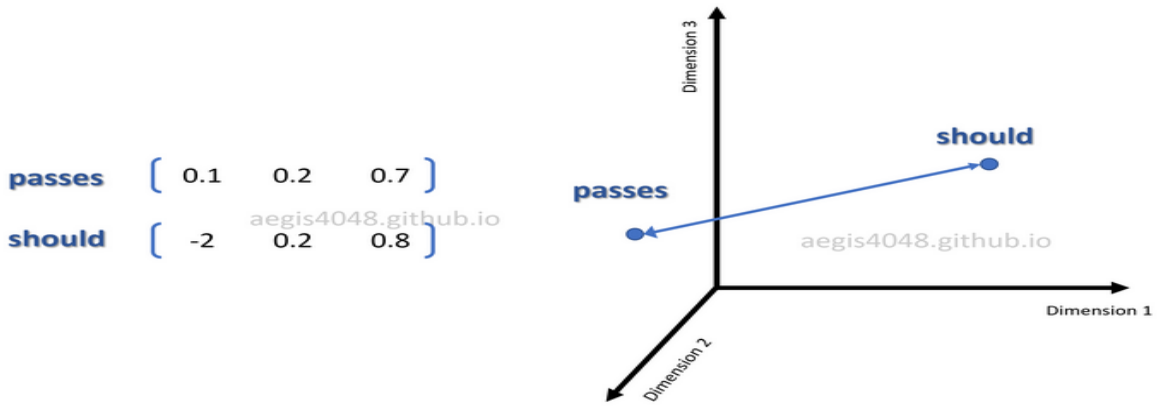
$$[0.1 \quad 0.2 \quad 0.7]$$

و "*should*" دارای بردار کلمه

$$[-2 \quad 0.2 \quad 0.8]$$

است. از آنجایی که ما اندازه ماتریس وزن را روی اندازه `size=3` قرار می‌دهیم، ماتریس سه بعدی است و می تواند در فضای برداری سه بعدی تجسم شود:

[۱] Look-Up Table



شکل (۸) تجسم سه بعدی بردارهای کلمه در ماتریس تعبیه یا تنیدگی

بهینه‌سازی ماتریس‌های تعبیه (وزن) θ منجر به نمایش کلمات در یک فضای برداری با کیفیت بالا می‌شود و مدل می‌تواند روابط معناداری را بین کلمات به دست آورد.

نکته: θ در تابع هزینه

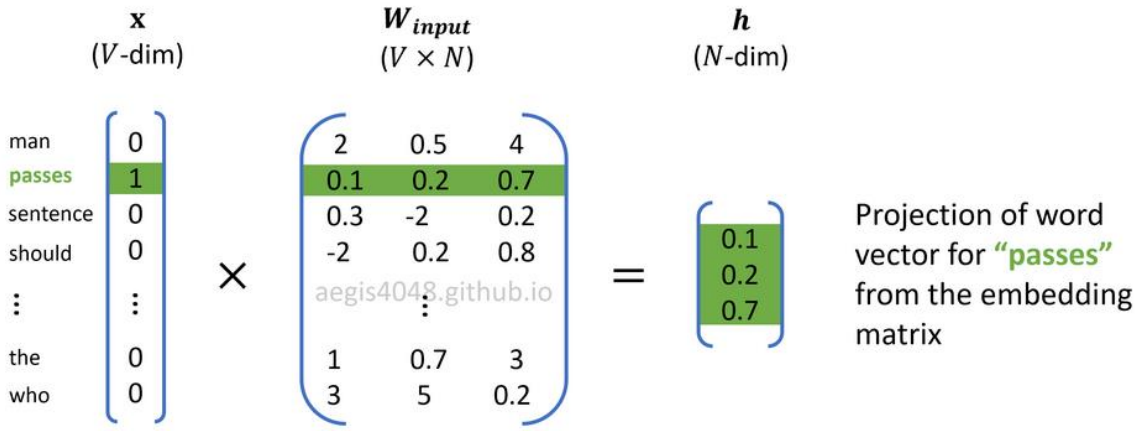
دو ماتریس وزنی وجود دارد که باید در مدل واج–پرش بهینه شوند: W_{input} و W_{output}. اغلب اوقات در شبکه عصبی، وزن‌ها به صورت θ بیان می‌شوند. در روش واج–پرش، θ ترکیبی از ماتریس‌های وزن ورودی و خروجی $[W_{input} \;\; W_{output}]$ است.

$$\theta = [W_{input} \;\; W_{output}] = \begin{bmatrix} u_{the} \\ u_{passes} \\ \vdots \\ u_{who} \\ u_{the} \\ u_{passes} \\ \vdots \\ u_{who} \end{bmatrix} \in \mathbb{R}^{۲NV}$$

θ دارای اندازه ۲V×N است که در آن V تعداد واژگان منحصر به فرد در یک پیکره است، و N نماینده بُعد بردارهای کلمه در ماتریس‌های تعبیه‌شده است. ۲ در V ضرب می‌شود زیرا دو ماتریس وزن W_{input} و W_{output} وجود دارد. u یک بردار کلمه از W_{input} و v یک بردار کلمه از W_{output} است. بردارهای هر کلمه، بردارهای ردیفی N بعدی از ماتریس‌های تعبیه ورودی و خروجی هستند.

۵,۱,۳ لایه پنهان (تصویرساز)

روش واج–پرش از یک شبکه عصبی با یک لایه پنهان استفاده می‌کند. در زمینه پردازش زبان طبیعی، لایه پنهان اغلب به عنوان یک لایه تصویرساز یا بازتاب نامیده می‌شود، زیرا h اساساً یک بردار N بعدی است که بر اساس بردار ورودی تک–نمود کدگذاری‌شده اصطلاحا تصویر شده یا بدست آمده است.



شکل (۹) محاسبه لایه پنهان (لایه تصویرساز یا بازتاب)

h با ضرب ماتریس تعبیه کلمه ورودی در بردار ورودی $V\text{-}dim$ (V بُعدی) به دست می‌آید.

$$h = W_{input}^{T}.x \in \mathbb{R}^{N} \qquad \text{معادله (۱۲)}$$

۵,۱,۴ لایه خروجی بیشنه-نرم (y_{pred})

لایه خروجی یک توزیع احتمال $V\text{-}dim$ از تمام کلمات منحصر به فرد در پیکره است که از یک کلمه مرکزی مشخص حاصل می‌شود. در آمار، احتمال شرطی A به شرط B به صورت $p(A|B)$ نشان داده می‌شود. در روش واج–پرش، ما از نماد $p(w_{context}|w_{center})$ برای نشان دادن احتمال شرطی مشاهده یک کلمه زمینه متنی به شرط مشاهده یک کلمه مرکزی استفاده می‌کنیم. با استفاده از تابع بیشینه–نرم داریم.

$$p(w_{context}|w_{center}) = \frac{\exp\left(W_{output_{(context)}}.h\right)}{\sum_{i=1}^{V}\exp\left(W_{output_{(i)}}.h\right)} \in \mathbb{R}^{1} \qquad \text{معادله (۱۳)}$$

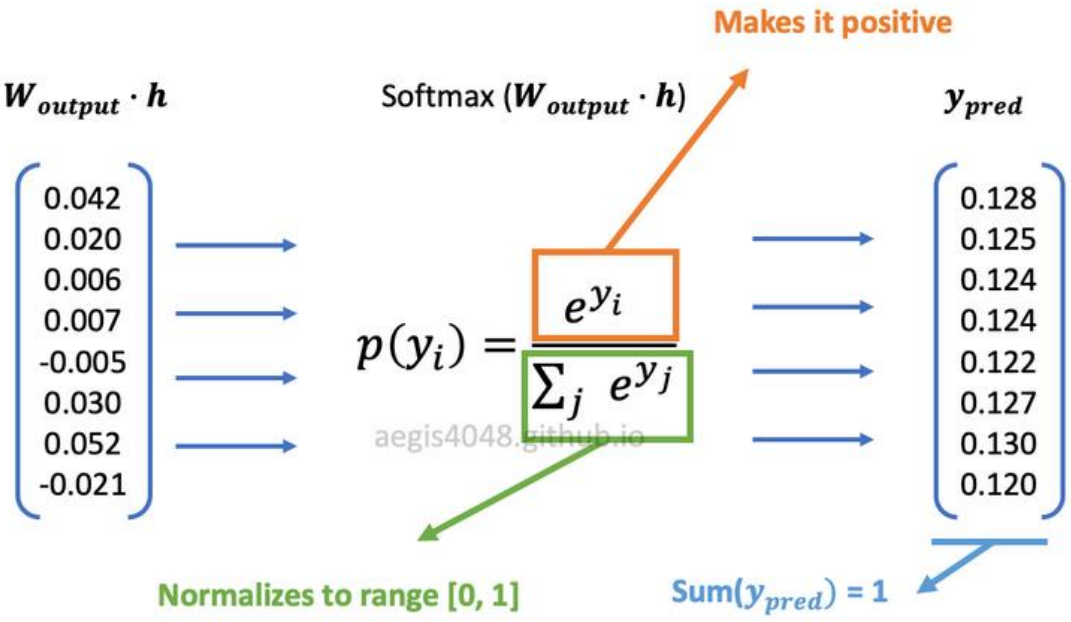
که $W_{output_{(i)}}$ بردار ردیف iام از ماتریس تعبیه خروجی به اندازه $N\times۱$ است، $W_{output_{(context)}}$ نیز یک بردار ردیفی به اندازه $N\times۱$ از ماتریس تعبیه خروجی مربوط به کلمه متنی است. V اندازه واژه منحصربه‌فرد در پیکره است و h لایه پنهان (تصویر ساز یا بازتاب) با اندازه $(N\times 1)$ است. خروجی یک مقدار اسکالر ۱×۱ احتمالی از محدوده $[۰،۱]$ است.

این احتمال V بار محاسبه می‌شود تا یک توزیع احتمال شرطی از مشاهده هر واژه‌ی منحصربه‌فرد در پیکره، با توجه به یک کلمه مرکزی به دست آید.

$$\begin{bmatrix} p(w_۱|w_{center}) \\ p(w_۲|w_{center}) \\ \vdots \\ p(w_V|w_{center}) \end{bmatrix} = \frac{\exp\left(W_{output}.h\right)}{\sum_{i=۱}^{V}\exp\left(W_{output_{(i)}}.h\right)} \in \mathbb{R}^V$$

معادله (۱۴)

W_{output} در مخرج معادله ۱۳ دارای اندازه $V\times N$ است. ضرب W_{output} در h با اندازه $N\times 1$ یک بردار حاصل ضرب نقطه‌ای به اندازه $V\times 1$ به دست می‌دهد. این بردار حاصل از ضرب نقطه‌ای ورودی تابع بیشینه–نرم است:



شکل (۱۰) تبدیل تابع بیشینه–نرم

تابع نمایی تضمین می‌کند که مقادیر تبدیل‌شده مثبت هستند و ضریب نرمالسازی در مخرج تضمین می‌کند که مقادیر دارای محدوده $[۰,۱]$ هستند. نتیجه، توزیع احتمال شرطی در مشاهده هر واژه منحصر به فرد در پیکره، با توجه به یک کلمه مرکزی است.

نکته: نمونه‌گیری منفی

تابع بیشینه–نرم در روش واج–پرش معادله زیر را دارد:

$$p = \frac{\exp\left(W_{output}.h\right)}{\sum_{i=1}^{V}\exp\left(W_{output_{(i)}}.h\right)} \in \mathbb{R}^{V}$$

یک مشکل در استفاده از بیشینه–نرم در روش واج–پرش وجود دارد – از نظر محاسباتی بسیار پرهزینه است، زیرا نیاز به بررسی (اسکن) کل ماتریس تعبیه خروجی (W_{output}) دارد تا توزیع احتمال همه کلمات V را محاسبه کند که V می‌تواند میلیون‌ها یا بیشتر باشد. علاوه براین، ضریب نرمال‌سازی در مخرج نیز به تکرار V نیاز دارد. هنگامی که در کدها پیاده سازی می‌شود، ضریب نرمال سازی فقط یک بار محاسبه می‌شود و به عنوان یک متغیر پایتون در حافظه پنهان ذخیره می‌شود، که پیچیدگی الگوریتم $O(V+V)\approx O(V)$ را ایجاد می‌کند.

به دلیل این ناکارآمدی محاسباتی، بیشینه–نرم در اکثر پیاده سازیهای روش واج–پرش استفاده نمی‌شود. در عوض، ما از جایگزینی به نام نمونه‌گیری منفی با تابع سیگموئید استفاده می‌کنیم، که مسئله را به مجموعه‌ای از وظایف طبقه‌بندی صفر و یکی (باینری) مستقل با پیچیدگی الگوریتم برابر $O(K+1)$ بازنویسی می‌کند، که در آن K معمولاً دارای محدوده [۵،۲۰] است. سپس توزیع احتمال جدید به صورت زیر تعریف می‌شود:

$$p = \frac{1}{1+\exp\left(-\left(\{c_{pos}\}\cup W_{neg}\right).h\right)} \in \mathbb{R}^{K+1}$$

$K=20$ برای پیکره کوچک و $K=5$ برای پیکره بزرگ استفاده می‌شود. نمونه برداری منفی بسیار ارزان‌تر از روش واج–پرش با بیشینه–نرم است، زیرا K بین ۵ تا ۲۰ است، در حالی که V می‌تواند میلیون‌ها باشد. علاوه بر این، هیچ تکرار اضافی برای محاسبه ضریب نرمالسازی در مخرج لازم نیست، زیرا تابع سیگموئید یک طبقه‌بندی‌کننده رگرسیون باینری است. پیچیدگی الگوریتم توزیع احتمال واج–پرش برابر $O(V)$ است، در حالی که در نمونه گیری منفی برابر $O(K+1)$ است. این نشان می‌دهد که چرا نمونه‌گیری منفی مقدار قابل توجهی از هزینه محاسباتی را در هر تکرار می‌کاهد.

۵،۲ آموزش: انتشار رو به عقب

انتشار رو به عقب شامل محاسبه خطاهای پیش‌بینی و به روزرسانی ماتریس وزن (θ) برای بهینه‌سازی نمایش برداری کلمات است. با فرض کاهش گرادیان تصادفی، معادلات به روزرسانی کلی زیر را برای ماتریس وزن (θ) داریم:

$$\theta_{new} = \theta_{old} - \eta \nabla_{J(\theta;w^{(t)})}$$ معادله (۱۵)

η نرخ یادگیری است، $\nabla_{J(\theta;w^{(t)})}$ گرادیان ماتریس وزن است و $J(\theta; w^{(t)})$ تابع هزینه تعریف شده در معادله (۶) است. از آنجایی که θ ترکیبی از ماتریس‌های وزن ورودی و خروجی $([W_{input} \ W_{output}])$ است که در بالا توضیح داده‌شد، دو معادله به‌روزرسانی برای هر ماتریس تعبیه شده وجود دارد:

$$W_{input}^{(new)} = W_{input}^{(old)} - \eta . \frac{\partial J}{\partial W_{input}}$$ معادله (۱۶)

$$W_{output}^{(new)} = W_{output}^{(old)} - \eta . \frac{\partial J}{\partial W_{output}}$$ معادله (۱۷)

به لحاظ ریاضی می‌توان نشان داد که گرادیان $W_{input} \ W_{output}$ فرم زیر را دارند:

$$\frac{\partial J}{\partial W_{input}} = x . (W_{output}^{T} \sum_{c=1}^{C} e_c)$$ معادله (۱۸)

$$\frac{\partial J}{\partial W_{output}} = h . \sum_{c=1}^{C} e_c$$ معادله (۱۹)

گرادیان‌ها را می‌توان در معادله ۱۶ و معادله ۱۷ جایگذاری کرد:

$$W_{input}^{(new)} = W_{input}^{(old)} - \eta . x . (W_{output}^{T} \sum_{c=1}^{C} e_c)$$ معادله (۲۰)

$$W_{output}^{(new)} = W_{output}^{(old)} - \eta . h . \sum_{c=1}^{C} e_c$$ معادله (۲۱)

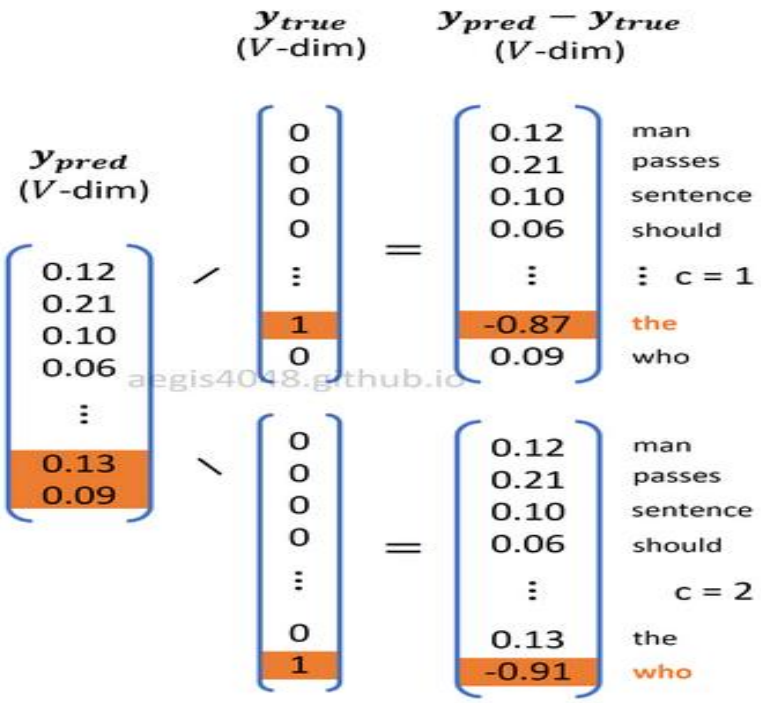
W_{output} و W_{input} ماتریس وزن ورودی و خروجی هستند، x یک لایه ورودی کدگذاری شده تک–نمود ، C اندازه پنجره، و e_c خطای پیش‌بینی برای c–امین کلمه متنی در پنجره است. توجه داشته باشید که h (لایه پنهان) معادل $W_{input}^{T} x$ است.

نکته: استفاده از بیشینه-نرم

اگرچه معادله (۲۱) صریحاً آن را نشان نمی‌دهد ، تابع بیشینه–نرم در خطای پیش بینی (e_c) اعمال می‌شود. خطای پیش‌بینی تفاوت بین احتمال پیش‌بینی‌شده و واقعی ($y_{pred} - y_{true}$) است که در زیر نشان داده شده‌است. احتمال پیش‌بینی‌شده y_{pred} با استفاده از تابع بیشینه–نرم و از معادله ۱۳ محاسبه می‌شود.

۵,۱,۱ خطای پیش بینی ($y_{pred} - y_{true}$)

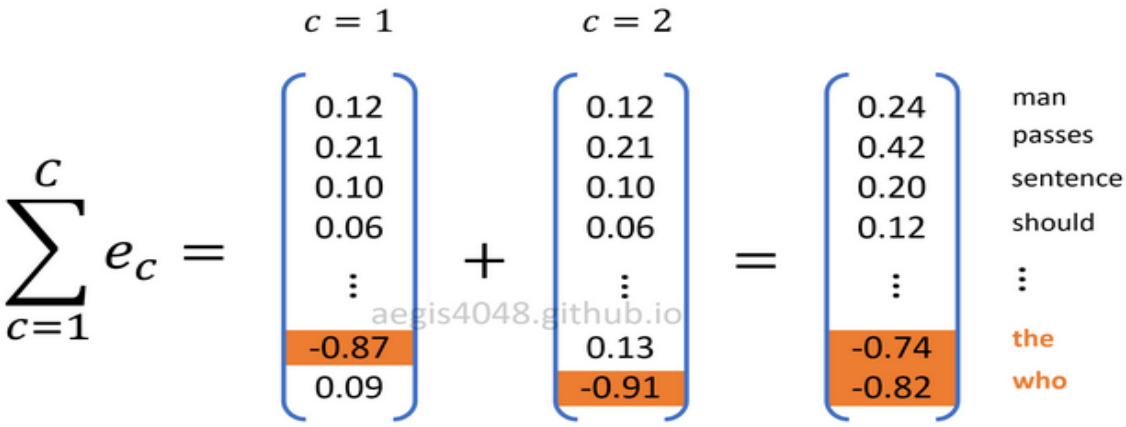
مدل بیشینه–نرم ماتریس وزن (θ) را برای کاهش خطای پیش‌بینی بهینه می‌کند. خطای پیش‌بینی تفاوت بین توزیع احتمال کلمات محاسبه‌شده از لایه خروجی بیشینه–نرم (y_{pred}) و توزیع احتمال واقعی (y_{true}) c–امین کلمه متنی است. درست مانند لایه ورودی، y_{true} یک بردار رمزگذاری شده تک–نمود است که در آن تنها یک عنصر در بردار که با کلمه متنی c مطابقت دارد برابر ۱ است و بقیه همه ۰ هستند.



شکل(۱۱) پنجره خطای پیش‌بینی

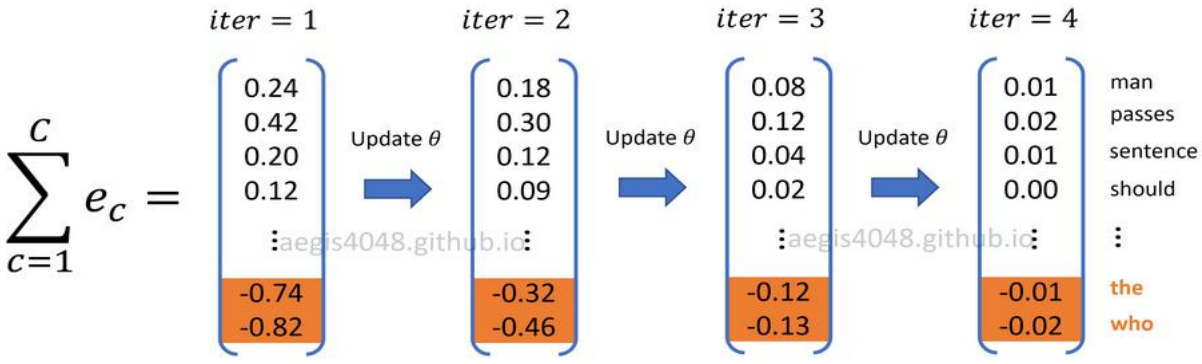
این شکل، دارای پنجره‌ای به اندازه ۲ است، بنابراین دو خطای پیش‌بینی محاسبه شده‌است. از نکات بالا در مورد اندازه پنجره می‌دانیم که طبقه‌بندی‌کننده رگرسیون بیشینه–نرم اصلی (معادله ۱۰) دارای K برچسب برای طبقه‌بندی است که در کاربرد پردازش زبان طبیعی آن، $K=V$ است، زیرا V کلمه برای طبقه‌بندی وجود دارد. استفاده از اندازه پنجره، معادله (۱۰) را به معادله (۱۱) تبدیل می‌کند و پیچیدگی الگوریتم را به طور قابل توجهی کاهش می‌دهد، زیرا این مدل تنها به محاسبه خطاهای پیش‌بینی برای ۲ تا ۱۰ کلمه همسایه (یعنی به اندازه مقدار ابرپارامتر C است) به جای محاسبه همه V کلمه نیاز دارد. خطاهای پیش بینی برای همه واژه‌ها بعضا ممکن است میلیون‌ها یا بیشتر باشد.

سپس، خطاهای پیش‌بینی برای همه C کلمه زمینه متنی، با هم جمع شده تا گرادیان‌های وزن را محاسبه کند. این کار بر اساس روابط (۱۸) و (۱۹) انجام می‌شود.



شکل (۱۲) مجموع خطاهای پیش‌بینی

همانطور که ماتریس‌های وزن بهینه می‌شوند، خطای پیش‌بینی برای همه کلمات در بردار خطای پیش‌بینی $\sum_{c=1}^{C} e_c$ به ۰ همگرا می‌شود.



شکل (۱۳) همگرایی خطاهای پیش‌بینی به صفر با انجام بهینه‌سازی

٦ نمایش عددی

برای تسهیل نمایش، تصاویری از اکسل برای نشان دادن مفهوم به‌روزرسانی ماتریس وزن از طریق انتشار به جلو و انتشار به عقب استفاده خواهند شد.

پیش‌روی: محاسبه لایه مخفی (بازتاب)

کلمه مرکزی "passes" است. اندازه پنجره برابر با ۱ است `size=1` ، و دو واژه "the" و "who" به عنوان کلمات زمینه در متن در نظر گرفته شوند. لایه مخفی (h) از ماتریس وزن ورودی جستجو می‌شود و با استفاده از معادله ۱۲ محاسبه می‌شود.

Corpus = **"The man who passes the sentence should swing the sword."**

#	Token	x
0	man	0
1	**passes**	**1**
2	sentence	0
3	should	0
4	swing	0
5	sword	0
6	**the**	**0**
7	**who**	**0**

(1 X 8)

×

W_input		
-0.078	0.018	0.033
0.068	0.170	-0.109
-0.158	-0.081	-0.151
0.150	0.064	0.145
-0.097	-0.055	0.188
0.036	0.071	0.059
0.168	-0.060	-0.058
0.098	0.015	0.096

(8X3)

=

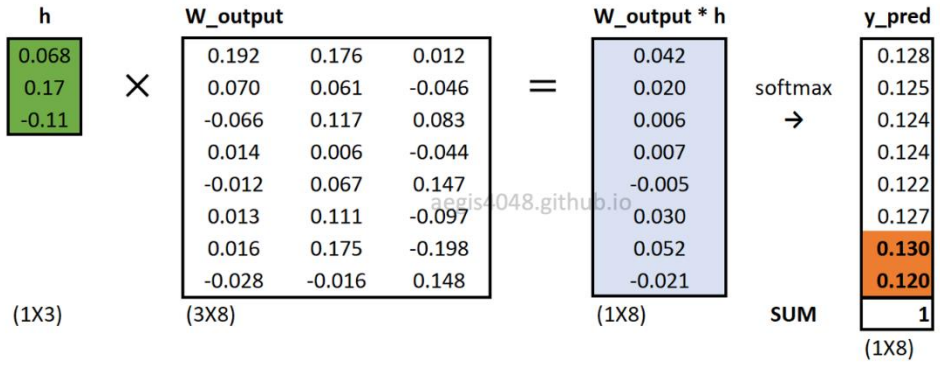
h
0.068
0.170
-0.109

(1X3)

شکل (١٤) محاسبه لایه مخفی (بازتاب)

پیش‌روی: لایه خروجی بیشینه-نرم

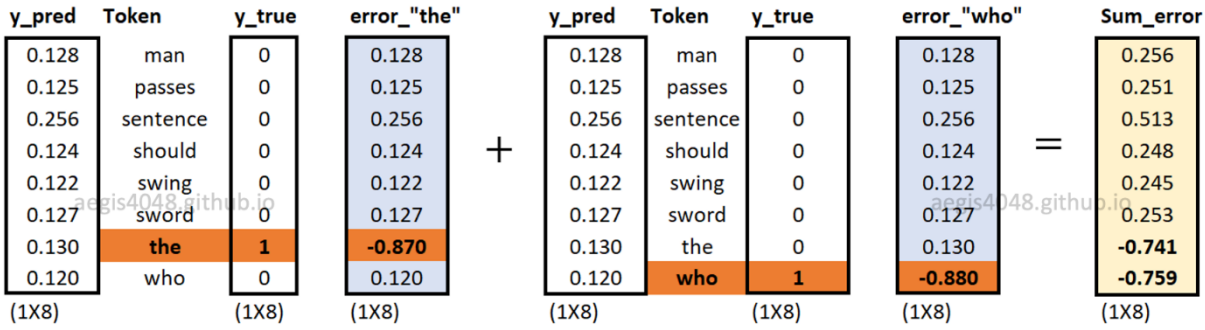
لایه خروجی، یک توزیع احتمال برای تمام کلمات، با فرض داشتن یک **کلمه مرکزی** معلوم است. این احتمال با استفاده از معادله (۱۴) محاسبه می‌شود. توجه داشته باشید که تمام پنجره‌های زمینه‌ای به یک لایه خروجی مشترک (y_{pred}) دسترسی دارند. تنها خطاها (e_c) متفاوت هستند.



شکل (۱۵) لایه خروجی بیشینه–نرم

پس‌روی: مجموع خطاهای پیش‌بینی

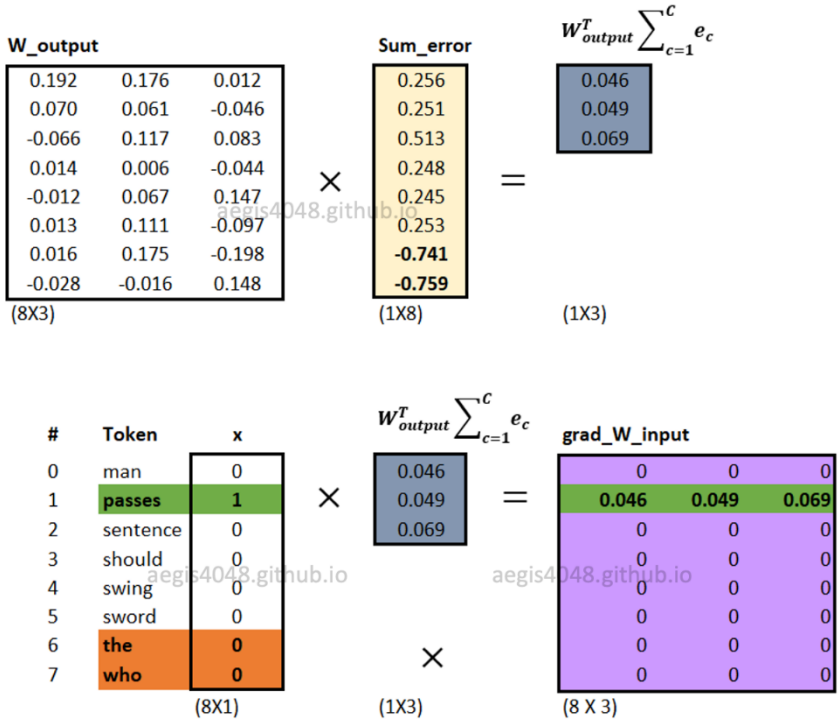
C خطای مختلف پیش‌بینی محاسبه شده و سپس جمع می‌شوند. در این مورد، از آنجا که ما پنجره را به ۱ تنظیم کردیم، `window=1` تنها دو خطا محاسبه می‌شود.



شکل (۱۶) خطاهای پیش‌بینی کلمات زمینه‌ای

پس‌روی: محاسبه گرادیان نسبت به وزن ورودی ∇W_{input}

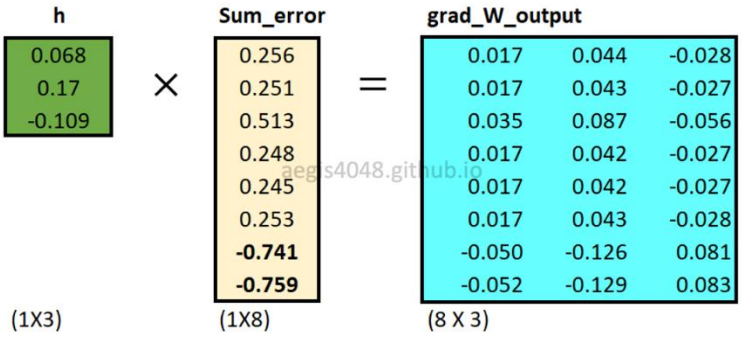
گرادیان‌های ماتریس وزن ورودی $(\frac{\partial J}{\partial W_{input}})$ با استفاده از معادله (۱۸) محاسبه می‌شوند. توجه داشته باشید که ضرب $W_{output}^{T} \sum_{c=1}^{C} e_c$ در بردار ورودی (x) که بصورت تک–نمود کدشده، باعث می‌شود تا شبکه عصبی تنها یک بردار کلمه را به‌روزرسانی کند که متناظر با کلمه ورودی (مرکزی) است.



شکل (۱۷) محاسبه گرادیان نسبت به وزن ورودی ∇W_{input}

پس‌روی: محاسبه گرادیان ماتریس وزن خروجی ∇W_{output}

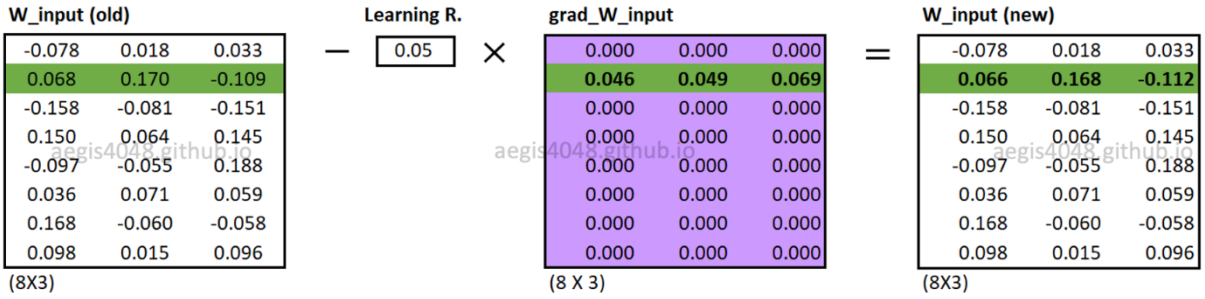
گرادیان‌های ماتریس وزن خروجی $(\frac{\partial J}{\partial W_{output}})$ با استفاده از معادله (۱۹) محاسبه می‌شوند. برخلاف ماتریس وزن ورودی (W_{input})، تمام بردارهای کلمه در ماتریس وزن خروجی (W_{output}) به‌روزرسانی می‌شوند.



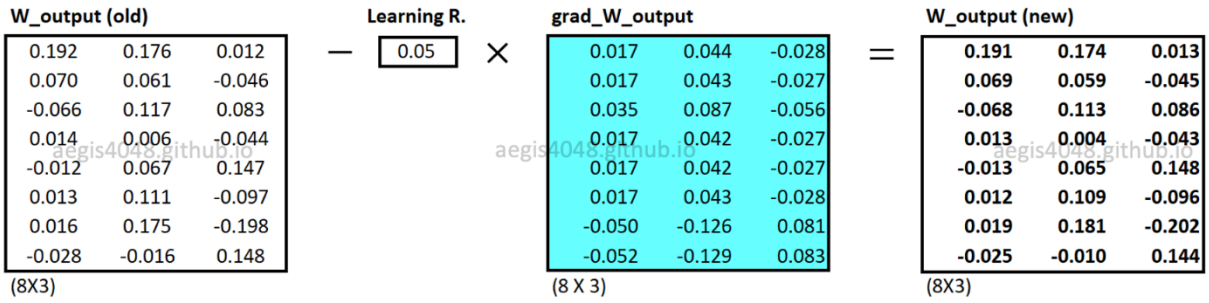
شکل (۱۸) محاسبه گرادیان ماتریس وزن خروجی ∇W_{output}

پس‌روی: به‌روزرسانی ماتریس‌های وزن

ماتریس‌های وزن ورودی و خروجی ($[W_{input} \quad W_{output}]$) با استفاده از روابط در معادله (۲۰) و معادله (۲۱) به‌روزرسانی می‌شوند.



شکل (۱۹) به‌روزرسانی W_{input}



شکل (۲۰) به‌روزرسانی W_{output}

توجه داشته باشید که برای هر تکرار در فرآیند یادگیری، تمام وزن‌ها در W_{output} به‌روزرسانی می‌شوند، اما تنها یک بردار سطر که متناظر با کلمه مرکزی است، در W_{input} به‌روزرسانی می‌شود. زمانی که مدل به اتمام به‌روزرسانی هر دو ماتریس وزن می‌رسد، یک تکرار کامل می‌شود. سپس مدل به تکرار بعدی با کلمه مرکزی بعدی منتقل می‌شود. با این حال، به‌یاد داشته باشید که این از معادله (۸) به عنوان تابع هزینه استفاده کرده و از کاهش گرادیان تصادفی بهره می‌گیرد. این بدان معناست که برای هر مثال آموزشی، یک به‌روزرسانی انجام می‌شود. اگر به عنوان تابع هزینه از معادله (۹) استفاده شود (که به ندرت رخ می‌دهد)، در این صورت یک به‌روزرسانی برای همه T مثال آموزشی در پیکره واژگان انجام می‌شود.

منبع

https://aegis٤٠٤٨.github.io/demystifying_neural_network_in_skip_gram_language_modeling#eq-١