3. Neuronowe modele języka

Modele n-gramowe rozszerzone o wydajne wygładzanie rekurencyjne (model Knesera-Neya, patrz sekcja 2.2.1) są zaskakująco dobre w praktyce i przez długie lata było bardzo ciężko uzyskać lepsze wyniki innymi metodami. Mają jednak one oczywiste wady: trudność z modelowaniem długich zależności oraz mało wydajne uogólniania wiedzy na nowe zdania. Uogólnianie odbywa się poprzez sklejanie małych fragmentów identycznych podsekwencji, np. dla modelu trzygramowego „Chłopak jedzie tramwajem do domu" jest zupełnie niepodobne do „Dziewczyna jedzie autobusem do kawiarni", gdyż żaden z trzygramów się nie pokrywa. Ponadto, modele tradycyjne stają się coraz to dokładniejsze gdy przechowujemy wiele zliczeń coraz dłuższych n-gramów, jednak stają się wtedy bardzo duże. Czasami mocne n-gramowe modele nawet z wieloma trikami przycinającymi rozmiar bazy danych potrafią sięgać rozmiaru stu gigabajtów. Bez komentarza pozostawmy pytanie w jaki sposób taki model można zmieścić w pamięci w telefonie.

Modele n-gramowe estymują prawdopodobieństwo poprzez proste zliczanie. Jeśli jednak przypomnisz sobie metody uczenia maszynowego to istnieje możliwość estymowania prawdopodobieństwa na wiele różnych sposobów. Czy jest możliwe wykorzystanie innych metod uczących do tworzenia modeli n-gramowych?

3.1 Modele logarytmiczno-liniowe

Model n-gramowy możemy wyrazić w ogólnej postaci jako:

$$P(s) = P(w_1^{|s|+1}) = \prod_{i=1}^{|s|+1} P(w_i | w_{i-n+1}^{i-1})$$

gdzie prawdopodobieństwa poszczególnych n-gramów $P(w_i|w_{i-n+1}^{i-1})$ muszą być w jakiś sposób estymowane i dotychczas robiliśmy to przy wykorzystaniu np. estymaty maksymalnej wiarygodności bez żadnych dalszych założeń (zliczanie) lub innych estymat z

wygładzaniem (zliczanie z dodatkami). Taką estymatę można jednak uzyskać stosują dowolny klasyfikator wieloklasowy z probabilistycznym wyjściem. Klasyfikator taki musiałby potraktować każde możliwe następne słowo jako klasę (czyli $|V|+1$ klas) a na wejście musiałby otrzymać reprezentację poprzednich słów tj. słów w warunku w_{i-n+1}^{i-1}.

Przykładem takiego klasyfikatora jest wielomianowa regresja logistyczna, w środowisku głębokiego uczenia maszynowego najczęściej znana pod nazwą sotfmax. Choć jest ona z pewnością bardzo dobrze znana, pozwólmy sobie na przypomnienie jej wzoru:

$$P(y=i|\mathsf{x}) = \frac{e^{\mathsf{w}^{(i)^T}\mathsf{x}+b^{(i)}}}{\sum_{j\in Y} e^{\mathsf{w}^{(j)^T}\mathsf{x}+b^{(j)}}}$$

gdzie Y oznacza zbiór wszystkich możliwych klas. W tym klasyfikatorze z każdą klasą $i \in Y$ skojarzony jest indywidualny wektor wag $\mathsf{w}^{(i)}$ oraz wyraz wolny $b^{(i)}$. Prawdopodobieństwo wyznacza się jako wynik modelu liniowego $\mathsf{w}^{(i)^T}\mathsf{x}+b^{(i)}$ znormalizowany funkcją softmax. Oznacza to, że prawdopodobieństwo i-tej klasy jest wprost proporcjonalne do $\mathsf{w}^{(i)^T}\mathsf{x}+b^{(i)}$ co możemy zapisać jako

$$P(y=i|\mathsf{x}) \propto \mathsf{w}^{(i)^T}\mathsf{x}+b^{(i)}$$

Klasyfikator softmax, podobnie jak regresja logistyczna, jest zwykle uczony za pomocą estymacji maksymalnej wiarygodności oraz może być wydajny obliczany na procesorach typu GPU.

Funkcja softmax ma wiele interesujących właściwości, które tutaj pokrótce przypomnimy:

- niezmienność ze względu na dodawanie stałej (niezwykle istotna w implementacji)
- możliwość uzyskania tej samej ekspresywności modelu poprzez usunięcie modelu liniowego dla jednej z klas (założenie że jest ona punktem referencyjnym i wynik modelu liniowego zawsze wynosi 0).
- Powyższa właściwość ta pozwala nam pokazać, że dwu-klasowy klasyfikator softmax jest tożsamy z (dwuklasową) regresją logistyczną oraz na pozbycie się jednego z wektorów w_i (czego zwykle nie robimy w praktyce by zachować stabilność numeryczną).
- wymnażając wartości przez dodatnią stałą $c > 1$ softmax dąży do rozkładu z $P(y = i|\mathsf{x}) = 1$ dla pewnego i oraz $P(y = j|\mathsf{x}) = 0$ dla wszystkich innych klas j.
- wymnażając wartości przez dodatnią stałą $c < 1$ softmax dąży do rozkładu jednorodnego.

Problem 3.1 Udowodnij właściwości 1—3 oraz przelicz po przykładzie dla własności 4 i 5.

Używając klasyfikatora softmax do tworzenia bigramowego modelu językowego, musimy wyrazić $P(w_i|w_{i-1})$ jako $P(y|\mathsf{x})$. W związku z powyższym, klasyfikator będzie miał $|V|+1$ klas, z których każda będzie odpowiadała konkretnemu słowu, które należy przewidzieć. Z kolei część warunkowa prawdopodobieństwa czyli poprzednie słowo w_{i-1} musi zostać opisana cechami i przedstawiona jako wektor cech x. Najprostszą taką reprezentacją jest kodowanie „jeden z n” (ang. *one-hot encoding*) czyli zaprezentowanie zmiennej dyskretnej poprzez wypełniony zerami wektor o długości równej liczbie możliwych wartości zmiennych z jedną wybraną pozycją podmienioną na wartość 1, wskazującą na wartość

zmiennej. W naszym przypadku będzie to wektor o długości $|V|+1$ z wartością jeden na indeksie odpowiadającym słowu w_{i-1}. Wektor zer z jedynką na pozycji i-tej będziemy oznaczać jako $\mathbb{1}_i$.

Rozpiszmy prawdopodobieństwo rozpoczęcia zdania od słowa „Ala", aby zobaczyć w jaki sposób klasyfikator softmax modeluje prawdopodobieństwa kolejnych słów.

$$P(Ala|\boxed{\text{START}}) = P(y = Ala|\mathsf{x} = \mathbb{1}_{\boxed{\text{START}}}) \propto \mathsf{w}^{(Ala)^T}\mathbb{1}_{\boxed{\text{START}}} + b^{(Ala)} = \mathsf{w}^{(Ala)}_{\boxed{\text{START}}} + b^{(Ala)}$$

gdzie ostatnia równość wynika z faktu, że wszystkie wagi są mnożone przez 0 i tylko jedna waga na pozycji odpowiadającej tokenowi $\boxed{\text{START}}$ jest mnożona przez 1 i zwracana jako wynik. Klasyfikator modeluje więc (nieznormalizowane) prawdopodobieństwo jako sumę dwóch liczb, które są ustalane w procesie nauki. Jedna z nich zależy tylko od przewidywanego słowa, niezależnie od poprzedniego słowa czyli możemy interpretować $b^{(Ala)}$ jako (nieznormalizowane) prawdopodobieństwo, że słowo „Ala" występuje w korpusie, niezależnie od warunku. Z kolei $\mathsf{w}^{(Ala)}_{\boxed{\text{START}}}$ możemy interpretować jako (nieznormalizowane) prawdopodobieństwo, że „Ala" występuje po tokenie $\boxed{\text{START}}$. Gdybyśmy zdecydowali się na ominięcie wyrazów wolnych, otrzymalibyśmy odpowiednik zwykłego modelu bi-gramowego tj. model posiadający po jednej liczbie dla każdego możliwego bigramu.

Analogicznie możemy zdefiniować model trzy-gramowy, jednak stoimy przed problemem reprezentacji dwóch ostatnich słów w postaci wektora cech. Możemy znów zastosować kodowanie 1 z n tym razem dla pary słów uzyskując wektor cech o długości $|V|^2$, jednak zwykle stosujemy oszczędniejszy wariant. Mianowicie, wykorzystujemy kodowanie 1 z n aby zakodować każde ze słów w warunku, a następnie konkatenujemy powstałe reprezentacje tj. uzyskując wektor cech o długości $2|V|$.

$$P(Ala|\boxed{\text{START}},\boxed{\text{START}}) = P(y = Ala|\mathsf{x} = [\mathbb{1}_{\boxed{\text{START}}}, \mathbb{1}_{\boxed{\text{START}}}]) \propto \mathsf{w}^{(Ala)}_{\boxed{\text{START}}(-1)} + \mathsf{w}^{(Ala)}_{\boxed{\text{START}}(-2)} + b^{(Ala)}$$

Tym razem nasz wektor miał dwie jedynki więc z iloczynu wektorowego zachowały się dwie różne wagi. Jedna reprezentuje czynnik powiązany z możliwością wystąpienia „Ali" jeśli na poprzedniej pozycji było $\boxed{\text{START}}$, a drugi reprezentuje szansę wystąpienia „Ali" jeśli dwie pozycje temu był $\boxed{\text{START}}$.

Zwróć uwagę, że model znacznie uprościł modelowane prawdopodobieństwo. Nie jest możliwe modyfikowanie prawdopodobieństwa biorąc pod uwagę kombinację słów w warunku. Rozważmy prawdopodobieństwo, że po „Zuzia jest" występuje słowo „lekarką". Powyższy model weźmie pod uwagę fakt że „lekarką" jest słowem które występuje z daną częstością, to jak często „lekarką" występuje po słowie „jest" oraz to jak często „lekarką" występuje jeśli dwa słowa temu było „Zuzia". Potraktuje jednak te trzy czynniki osobno i nie weźmie pod uwagę możliwych interakcji między zmiennymi tj. jak często „lekarka" występuje po kombinacji słów „Zuzia jest". W analizowanym przykładzie może to mieć duże znacznie. Po słowie „jest" może być bardzo wiele słów: prawidłowo odmienione rzeczowniki jak „lekarzem" czy „chomikiem", przymiotniki itd. Jeśli dwa słowa temu było „Zuzia" to wyrazy „lekarzem", „lekarką" i „chomikiem" mogą być podobnie prawdopodobne. Dopiero kombinacja „Zuzia jest" mocno pomniejsza szansę na wyraz „chomikiem" i preferuje wyraz „lekarką" nad „lekarzem" – tego jednak model nie jest w stanie zamodelować, choć model zliczający mógłby to zrobić.

Niemniej jednak zaletą użycia klasyfikatora softmax jest to, że możemy opisać poprzednie słowa bardziej złożonymi cechami niż kodowanie 1 z n, co przy dobrej inżynierii cech daje możliwość uzyskania dobrych wyników.

Kończąc opis tego modelu pokrótce podsumujmy jego użycie do

- estymacji prawdopodobieństwa zdania. Zgodnie z modelem n-gramowy rozbijamy prawdopodobieństwo zdania jak poprzednio, zastępując konkretne wartości uzyskane poprzez zliczenie wartościami uzyskanymi z klasyfikatora. Mając zdanie o długości n_s musimy więc wykonać n_s+1 predykcji klasyfikatorem.
- generacji zdania. Wykonujemy predykcję klasyfikatorem dla reprezentacji początku zdania (odpowiednio długa sekwencja START), a ze zwróconego rozkładu prawdopodobieństwa po klasach wybieramy losowo jedną w_1. Uruchamiamy klasyfikator z reprezentacją cech nowego warunku (np. dla trzygramu „START w_1") i powtarzamy procedurę tak długo aż nie wylosujemy tokenu STOP.

■ **Przykład 3.1** Model trzy-gramowy operujący na słowniku $V=\{A,B\}$ (tokeny startu i stopu pomijamy w zadaniu) wyestymował następujące prawdopodobieństwa:

$$P(A|A,A)=0.1 \qquad P(A|A,B)=0.2 \qquad P(A|B,A)=0.5 \qquad P(A|B,B)=0.4$$

co oznacza że odpowiednio $P(B|A,A)=0.9$, $P(B|A,B)=0.8$ itd. Pokaż, że opisany wyżej model trzygramowy oparty na klasyfikatorze softmax nie ma możliwości nauczenia się tego rozkładu.

Model oparty na klasyfikatorze softmax wyrazi te prawdopodobieństwa jako:

$$P(A|A,A) \propto \mathsf{w}^{(A)}_{A(-1)} + \mathsf{w}^{(A)}_{A(-2)} + b^{(A)}$$

$$P(A|A,B) \propto \mathsf{w}^{(A)}_{A(-1)} + \mathsf{w}^{(A)}_{B(-2)} + b^{(A)}$$

$$P(A|B,A) \propto \mathsf{w}^{(A)}_{B(-1)} + \mathsf{w}^{(A)}_{A(-2)} + b^{(A)}$$

$$P(A|B,B) \propto \mathsf{w}^{(A)}_{B(-1)} + \mathsf{w}^{(A)}_{B(-2)} + b^{(A)}$$

Analizując uzyskany rozkład otrzymujemy następujące relacje

$$\begin{aligned} P(A|A,A) < P(A|A,B) &\Rightarrow \mathsf{w}^{(A)}_{A(-1)} + \mathsf{w}^{(A)}_{A(-2)} + b^{(A)} < \mathsf{w}^{(A)}_{A(-1)} + \mathsf{w}^{(A)}_{B(-2)} + b^{(A)} \\ &\Rightarrow \mathsf{w}^{(A)}_{A(-2)} < \mathsf{w}^{(A)}_{B(-2)} \end{aligned}$$

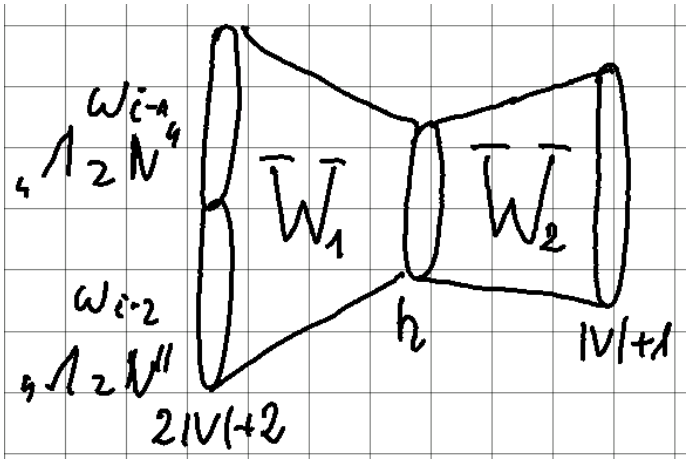
$$\begin{aligned} P(A|B,A) > P(A|B,B) &\Rightarrow \mathsf{w}^{(A)}_{B(-1)} + \mathsf{w}^{(A)}_{A(-2)} + b^{(A)} > \mathsf{w}^{(A)}_{B(-1)} + \mathsf{w}^{(A)}_{B(-2)} + b^{(A)} \\ &\Rightarrow \mathsf{w}^{(A)}_{A(-2)} > \mathsf{w}^{(A)}_{B(-2)} \end{aligned}$$

co prowadzi do sprzeczności. ■

3.2 Neuronowe modele autoregresywne

3.2.1 Modele autoregresywne

Analogicznie, jak wyżej dla modelu softmax, za estymatę $P(w_i|w_{i-n+1}^{i-1})$ możemy podstawić wynik wielowarstwowej sieci neuronowej. W najprostszym i dość praktycznym przypadku możemy zastosować sieć dwuwarstwową do której wejściem są słowa w reprezentacji „1 z n", warstwę ukrytą z h neuronami z dowolną funkcją aktywacji, a na wyjściu mamy



Rysunek 3.1: Model trzy-gramowy przy użyciu dwuwarstwowej sieci neuronowej z wektorami wag zapisanymi dla każdej z warstw jako W_1 i W_2.

warstwę softmax z $|V|+1$ neuronami tj. po jednym dla każdego słowa i tokenu $\boxed{\text{STOP}}$. Zastosowanie sieci neuronowej zamiast modelu softmax ma jednak pewne zalety: zwykle dużo mniejszą liczbę parametrów do nauki.

3.2.2 Modele z macierzą zanurzeń

Pomimo zastosowania sieci neuronowych, głównych problemów modeli n-gramowych nie udało nam się rozwiązać. Nadal mamy do czynienia ze słabym uogólnianiem wiedzy: reprezentacja w warstwie ukrytej składa się z neuronów które reprezentują warunek trzy-gramu po prostu jako sumę dwóch wag odpowiadającym słowom na pozycji $i-1$ i $i-2$. Nie rozwiązany został również problem długich zależności. Co prawda, liczba parametrów modelu rośnie liniowo z długością n-gramu $O(nh|V|)$, w przeciwieństwie do modeli zliczających gdzie rośnie ona wykładniczo, jednak przy słownikach o praktycznych wielkościach $|V|$ rzędu setek tysięcy zwiększenie n mocno zwiększa złożoność modelu. Nadal więc nie jest praktyczne trenowanie modeli z długimi n-gramami. Podobna uwaga odnosi się do zwiększania liczby neuronów w warstwie ukrytej h.

W przypadku modeli zliczających, poprawę zdolności uogólniania wiedzy uzyskaliśmy poprzez użycie modeli klasowych. Zamienialiśmy wtedy zdania w korpusie na abstrakcyjne symbole klas (grup) słów C_i, do których przypisane były słowa o podobnych własnościach semantycznych i syntaktycznych. Sprawiało to, że model poszukiwał w tekście zlepków postaci $C_{i-2}C_{i-1}C_i$ gdzie każdy z symboli to cała grupa słów – zwiększa to wykładniczo szansę na znalezienie takiego zlepka w korpusie i uogólnienie wiedzy. Spróbujmy zaproponować neuronowy odpowiednik modelu klasowego.

Modele zliczające słowa mają naturą dyskretną – naturalne więc było, że grupy C_i dla poszczególnych słów były twardymi grupami (ang. *hard clustering*) tj. każde (dyskretne) słowo zamieniało w konkretny, dyskretny symbol. Sieci neuronowe pracują na reprezentacjach ciągłych, więc neuronowy model klasowy zamiast zwracać dyskretne symbole grup będzie operował na ciągłej reprezentacji „grupy”. Poprzez $C(w_i)$ oznaczmy funkcję zwracającą ciągłą reprezentację słowa w_i, która jest podobna dla słów bliskich semantycznie i syntaktycznie tj. słowa o podobnej reprezentacji $C(w_i)$ stoją na podobnych pozycjach w zdaniu i mogą być wzajemnie zastępowane nadal tworząc prawidłowe zdania.

Używanie $C(w_i)$ zamiast bezpośrednio w_i w sieci neuronowej spowoduje taki sam efekt jak widzieliśmy w modelach klasowych. Podobne słowa będą miały podobne reprezentacje $C(w_i)$, a ponieważ funkcja modelowana przez sieć neuronową jest ciągła i gładka to mała zmiana wejścia spowoduje małą zmianę wyjścia [1]. Zamiana słowa na inne o podobnym znaczeniu spowoduje zatem bardzo małą zmianę w rozkładzie prawdopodobieństwa zwracanym przez funkcję neuronową. W związku z tym jeśli w korpusie zobaczymy trzy-gram „Pojechałem do szkoły" i – postępując zgodnie z maksymalizacją funkcji wiarygodności – zwiększymy mu prawdopodobieństwo, to zakładając że „Pojechałem" i „Śmignąłem" mają podobne reprezentacje równocześnie zwiększymy prawdopodobieństwo trzy-gramu „Śmignąłem do szkoły" (bo sieć neuronowa musi zwrócić podobny wynik). Oczywiście, jeśli „szkoły" i „domu" również mają podobne reprezentacje to również trzy-gramy „Pojechałem do domu" i „Śmignąłem do domu" staną się prawdopodobniejsze itd. Znów, zachodzi duże uogólnianie wiedzy!

Architektura modelu

Funkcja $C(w)$ ma zwracać dla danego słowa w jego reprezentację ciągłą, która zamiast jednej liczby przyjmuje zwykle postać wielowymiarowego wektora. Funkcja $C()$ ma $|V|+1$ możliwych wartości i dla każdej z nich zwraca wektor $\mathbb{R}^d$, możemy więc zdefiniować ją poprzez macierz C z d kolumnami i $|V|+1$ wierszami. Z chwilą gdy wywołamy np. $C(Ala)$ wystarczy odczytać wynik z odpowiedniego (d-wymiarowego) wiersza macierzy C_{Ala}.

Zakładając, że skądś mamy gotową macierz C jej użycie w modelu języka wydaje się trywialne. Poprzednio braliśmy słowa z warunku w_{i-1} i w_{i-2}, kodowaliśmy je kodowaniem „1 z n", konkatenowaliśmy powstałe wektory i umieszczaliśmy na wejście sieci neuronowej. Podobnie przy użyciu macierzy C: bierzemy słowa w_{i-1} i w_{i-2}, kodujemy je jako $C(w_{i-1}) = \mathsf{C}_{w_{i-1}}$ i $C(w_{i-2}) = \mathsf{C}_{w_{i-2}}$, uzyskane wektory konkatenujemy i umieszczamy je jako wejście sieci neuronowej. Macierz C nazywamy macierzą zanurzeń słów (ang. *word embedding matrix*).

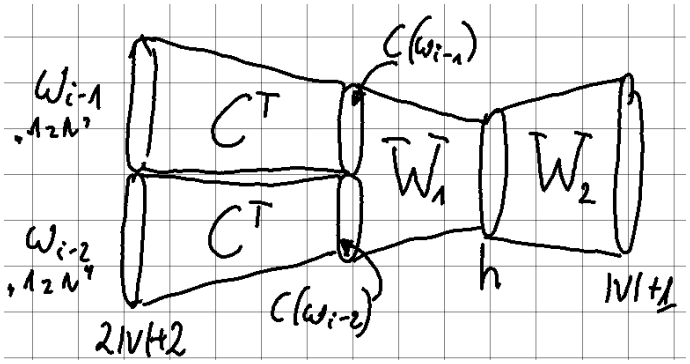
Problematyczne jednak wydaje się uzyskanie macierzy C, podobnie jak poprzednio trudne było uzyskanie wyniku grupowania słów. Jednakże, operację odczytu odpowiedniego wiersza z macierzy C możemy zapisać przy użyciu wektora „1 z n":

$$C(w) = \mathbb{1}_w^T \mathsf{C} = \mathsf{C}_w$$

co prowadzi nas do niezwykłego odkrycia: taką operację wykonuje zwykła warstwa sieci neuronowej mająca na wejściu tylko jedno słowo zakodowane w postaci „1 z n"! Powyższe stwierdzenie jest prawdziwe o ile neurony w tej warstwie nie będą miały wyrazów wolnych $b^{(i)}$, a wektory wag dla poszczególnych neuronów będą kolejnymi kolumnami macierzy C!

Dlaczego jest to niesamowite? W przypadku tradycyjnych modeli klasowych dużym problemem była jednoczesna nauka prawdopodobieństw modelu oraz przypisania do klas, co działo się w dwóch osobnych etapach. Brak możliwości prostej optymalizacji powodował konieczność używania heurystyk i bardzo kosztowne uczenie. W modelach neuronowych znika ten problem: zarówno „rozproszone klasy słów" jak i model języka możemy trenować łącznie za pomocą wstecznej propagacji błędu!

Ostatecznie więc na wejściu sieci mamy znów słowa w postaci „1 z n", które wchodzą do pierwszej warstwy sieci neuronowej, która zawiera $(n-1)d$ neuronów dla modelu n-gramowego. Wymiarowość ta bierze się z faktu, że jedna reprezentacja słowa ma d



Rysunek 3.2: „Klasowy" model trzy-gramowy (model z macierzą zanurzeń) przy użyciu dwuwarstwowej sieci neuronowej. Zwróć uwagę, że neurony w pierwszej warstwie służą tylko do odczytu zanurzenia słów i mają takie same wagi określone przez macierz C.

wymiarów, a każde z $(n-1)$ słów musi zostać być zaprezentowane jego reprezentacją ciągłą. Jednak ta pierwsza warstwa sieci różni się od typowej warstwy sieci. Neurony w pierwszej warstwie od 1 do d odpowiadają za odczyt pierwszego słowa w warunku więc mają one połączania tylko do pierwszych $|V|$ cech wejścia (kodowanie „1 z n" pierwszego słowa). Kolejna partia neuronów w tej warstwie tj. neurony od $d+1$ do $2d$ mają połączenia tylko do cech od $|V|+1$ do $2|V|$ itd. Co więcej każda z tych partii neuronów ma *takie same wagi* które są określone przez kolumny macierzy C. Architektura ta jest zwizualizowana na rysunku 3.2.

Na koniec zauważmy, że liczba parametrów nowego modelu jest rzędu $O(dV + nhd + hV)$ czyli powiększenie długości analizowanego n-gramu o 1 skutkuje zwiększeniem parametrów o hd. W modelu bez macierzy zanurzeń było to $h|V|$ gdzie $|V| \gg d$.

Podsumowanie

Modele z macierzą zanurzeń po raz pierwszy zostały zaprezentowane w pracy [1]. W przeprowadzonym eksperymencie na korpusie Browna (język angielski) najlepszy model 5-gramowy Kneser-Neya uzyskał nieokreśloność równą 321, najlepszy model klasowy (trzy-gram z 500 klasami, wygładzanie rekurencyjne) uzyskał 312, a neuronowy model z macierzą zanurzeń osiągnął nieokreśloność równą 268. Interpolacja uzyskanego modelu neuronowego z modelem trzy-gramowym (z równymi wagami) pozwoliła na uzyskanie nieokreśloności 252. Oznacza to redukcję nieokreśloności w stosunku do klasowego modelu n-gramowego o 24%.

Sieci neuronowe użyte w eksperymencie były dość małe na dzisiejsze standardy. Użyto modeli 5-gramowych z maksymalnie setką neuronów ukrytych aktywowanych tangensem hiperbolicznym, z wymiarowością reprezentacji $d = \{30, 60\}$. Stosowano również połączenia rezydualne pomiędzy pierwszą i ostatnią warstwą sieci. Tak ograniczony rozmiar sieci nie wynikał tylko z braku dostępności procesorów GPU: uczenie neuronowych modeli języka bez dodatkowych trików ich przyśpieszających jest kosztowne nawet dzisiaj. Aby uświadomić sobie skalę problemu: w opisywanej pracy trening modelu (przecież małej sieci) na korpusie AP News trwał trzy tygodnie na klastrze 40 CPU. W tym czasie udało się zrobić tylko 5 pełnych epok uczenia.

3.3 Przyśpieszanie neuronowych modeli języka

Neuronowe modele języka oferują zwykle lepszą jakość działania niż modele n-gramowe, a w przypadku uczenia modeli opartych na długich gramach mają też niższe wymagania pamięciowe. Te zalety jednak przychodzą w zamian za wielokrotnie dłuższy czas predykcji i uczenia. Model n-gramowy zaimplementowany przy użyciu odpowiednich struktur danych może odczytywać prawdopodobieństwa kolejnych słów w zasadzie niezależnie od liczby przechowywanych n-gramów w pamięci. Jednak model neuronowy nie tylko musi wykonać kilka znacznie kosztowniejszych od odczytu z pamięci operacji mnożenia macierzy, ale także musi obliczyć prawdopodobieństwa wystąpienia wszystkich możliwych słów ze słownika. Dzieje się tak dlatego, że warstwa softmax najpierw oblicza logity wszystkich wyjść, a dopiero potem je normalizuje. Samo obliczenie wszystkich logitów to mnożenie przez bardzo dużą macierz rzędu $|V| \times h$ gdzie $|V|$ jest rzędu setek tysięcy a h to liczba cech będąca wejściem do softmaxa. Jest to dominująca obliczeniowo operacja całego neuronowego modelu języka.

■ **Przykład 3.2** Załóżmy[a] neuronowy model z macierzą zanurzeń, który posiada zanurzenia o wymiarze $d = 300$ i jedną warstwę ukrytą złożoną ze $h = 100$ neuronów. Model zależy od $n = 5$ poprzednich słów i pracuje na słowniku o wielkości $|V| = 100000$. Ile procent operacji wykonuje się w ostatniej warstwie sieci? Dla uproszczenia załóż, że zarówno operacja obliczenia wartości jednego składnika iloczynu wektorowego (pomnóż i dodaj) jak i odczytu jednego elementu macierzy C zajmuje jedną jednostkę czasu.

Łączna liczba operacji w modelu to $(|V|+1)(1+h)+h(1+nd)+nd$ gdzie pierwszy składnik odpowiada za warstwę softmax. Podstawiając:

$$\frac{(|V|+1)(1+h)}{(|V|+1)(1+h)+h(1+nd)+nd} \approx 98.5\%$$

■

[a] zadanie inspirowane obliczeniami w [1]

Naiwnym rozwiązaniem tego problemu jest po prostu ograniczenie rozmiaru słownika do małej liczby i potraktowanie reszty słów jako UNK. Trochę lepszym sposobem jest użycie w predykcji jakiegoś modelu n-gramowego z prawdopodobieństwem określanym poprzez prawdopodobieństwo UNK w modelu neuronowym. Predykcje mają wtedy postać:

$$P(w|c) = P_{NN}(w|c) + P_{NN}(\text{UNK}|c)P_{n-\text{gram}}(w|c)$$

gdzie c to iluś gramowy kontekst słowa – w szczególności model n-gramowy może używać inną liczbę n-gramów niż model neuronowy (dla uproszczenia notacji zakłada się, że sieć neuronowa przewiduje wartość 0 dla każdego słowa spoza jej słownika). Jednak oba te rozwiązania mają swoje, raczej oczywiste, wady. Istnieje na szczęście kilka sposobów na przyśpieszenie softmaxa. W tym rozdziale omówimy metodę próbkowania ważonego, która pozwala na znaczne przyśpieszenie procesu uczenia oraz metodę hierarchicznego softmaxa która pozwala na przyśpieszenie zarówno uczenia jak i predykcji – jednak czasami za cenę delikatnie słabszych wyników.

3.3.1 Próbkowanie ważone

Przypomnijmy jak wygląda gradient ostatniej warstwy sieci neuronowej przy optymalizowaniu funkcji wiarygodności.

$$\begin{aligned}
\frac{\partial}{\partial\theta}\log P_\theta(w|c) &= \frac{\partial}{\partial\theta}\log\frac{e^{f_\theta(w)}}{\sum_{v\in V}e^{f_\theta(v)}} = \frac{\partial}{\partial\theta}\left[f_\theta(w) - \log\sum_{v\in V}e^{f_\theta(v)}\right] \\
&= \frac{\partial f_\theta(w)}{\partial\theta} - \frac{\partial}{\partial\theta}\log\sum_{v\in V}e^{f_\theta(v)} \\
&= \frac{\partial f_\theta(w)}{\partial\theta} - \frac{1}{\sum_{v'\in V}e^{f_\theta(v')}}\frac{\partial}{\partial\theta}\sum_{v\in V}e^{f_\theta(v)} \\
&= \frac{\partial f_\theta(w)}{\partial\theta} - \frac{1}{\sum_{v'\in V}e^{f_\theta(v')}}\sum_{v\in V}\frac{\partial}{\partial\theta}e^{f_\theta(v)} \\
&= \frac{\partial f_\theta(w)}{\partial\theta} - \frac{1}{\sum_{v'\in V}e^{f_\theta(v')}}\sum_{v\in V}e^{f_\theta(v)}\frac{\partial f_\theta(v)}{\partial\theta} \\
&= \frac{\partial f_\theta(w)}{\partial\theta} - \sum_{v\in V}\frac{e^{f_\theta(v)}}{\sum_{v'\in V}e^{f_\theta(v')}}\frac{\partial f_\theta(v)}{\partial\theta} \\
&= \frac{\partial f_\theta(w)}{\partial\theta} - \sum_{v\in V}P_\theta(v|c)\frac{\partial f_\theta(v)}{\partial\theta}
\end{aligned} \tag{3.1}$$

Pierwszy term tego wyrażenia wymaga od nas obliczenia gradientu po wartości logitu dla danego słowa w co nie jest problemem, natomiast obliczenie drugiego składnika wymaga niestety zsumowania po wszystkich słowach w słowniku i obliczenie całego znormalizowanego rozkładu zwracanego przez sieć.

Warto zauważyć, że $\sum_{v\in V}P_\theta(v|c)\frac{\partial f_\theta(v)}{\partial\theta}$ jest wartością oczekiwaną gradientu funkcji f przy rozkładzie P_θ określonym przez sieć. Możemy więc całe wyrażenie zapisać jako:

$$\frac{\partial}{\partial\theta}\log P_\theta(w|c) = \frac{\partial f_\theta(w)}{\partial\theta} - \mathbb{E}_{P_\theta(v|c)}\left[\frac{\partial f_\theta(v)}{\partial\theta}\right] \tag{3.2}$$

Jak pamiętamy ze statystyki, wartość oczekiwaną można przybliżyć (wyestymować) poprzez wylosowanie kilku próbek z rozkładu i policzenie średniej arytmetycznej. Na mocy prawa wielkich liczb estymata taka, wraz z rosnącym rozmiarem próbki, zbiega do prawdziwej wartości oczekiwanej rozkładu. Wydaje się, że wystarczy wylosować kilka słów, policzyć tylko dla nich gradient oraz uśrednić. Jeśli rozmiar próbki jest dużo mniejszy niż $|V|$ to obliczenia stały się dużo szybsze. Problem polega jednak na tym, że słowa muszą być wylosowane z konkretnego rozkładu, konkretnie P_θ... które jest określone przez sieć neuronową i którego obliczenie wymaga obliczanie całego softmaxa.

Z pomocą przychodzi próbkowanie ważone[1] (ang. *importance sampling*), które pozwala nam na wyestymowanie wartości oczekiwanej danego rozkładu P_θ poprzez losowanie z innego, referencyjnego rozkładu Q pod warunkiem że dla wylosowanych próbek

[1] Na MISiO uczyliście się o *likelihood weighting* co zostało przetłumaczone jako „ważone próbkowanie". W rzeczywistości „metoda ważenia prawdopodobieństwem" (?) jest jednym z konkretnych wcieleń metody ważonego próbkowania. Nazwa ważone próbkowanie odnosi się również w innych polskich źródłach do ogólnej postaci algorytmu, którą prezentuję tutaj. Przepraszam za zamieszanie w tłumaczeniach ;(

jesteśmy w stanie obliczyć wartość nieznormalizowanego prawdopodobieństwa P_θ dla konkretnej próbki. Będzie więc od nas wymagała obliczenia jedynie wartości logitu dla kilku słów bez konieczności normalizacji!

Pozostaje otwarte pytanie co można użyć jako rozkład referencyjny Q – tu przyda się nasza wiedza z modeli n-gramowych. Modele n-gramowe są bardzo szybkie w nauce i zwracają całkiem niezły model języka, stanowią więc znakomitą podstawę do zbudowania rozkładu referencyjnego. W praktyce często stosuje się modele bi-gramowe czy nawet unigramowe.

> ! Próbkowanie ważone nie jest metodą próbkowania – jest to metoda przybliżania wartości oczekiwanej!

Wyprowadzenie metody rozpocznijmy od podstawowej jej wersji w której wymagane są znormalizowane rozkłady, a potem uogólnimy ją na wersję z nieznormalizowanymi rozkładami. Uprościmy też notację: będziemy przybliżać wartość oczekiwaną $\mathbb{E}_P[h(X)]$ pewnej funkcji h na wartościach z rozkładu $P(x)$ poprzez losowanie z rozkładu $Q(x)$. Rozkład Q to dowolny rozkład dla którego prawdziwa jest implikacja $Q(x) = 0 \Rightarrow P(x) = 0$. Zapiszmy wzór na wartość oczekiwaną:

$$\mathbb{E}_P[h(X)] = \int P(x)h(x)dx$$

gdzie całka zastępowana jest sumą dla zmiennych dyskretnych. Następnie przemnóżmy wyrażenie przez $1 = \frac{Q(x)}{Q(x)}$ bez zmiany wyniku.

$$\mathbb{E}_P[h(X)] = \int P(x)h(x)dx = \int P(x)h(x)\frac{Q(x)}{Q(x)}dx = \int Q(x)h(x)\frac{P(x)}{Q(x)}dx = \mathbb{E}_Q\left[h(X)\frac{P(X)}{Q(X)}\right]$$

Możemy więc przybliżyć $\mathbb{E}_P[h(X)]$ generując k losowych próbek x_i z rozkładu Q i obliczając średnią arytmetyczną:

$$\mathbb{E}_Q\left[h(X)\frac{P(X)}{Q(X)}\right] \approx \frac{1}{k}\sum_{i=1}^{k} h(x_i)\frac{P(x_i)}{Q(x_i)}$$

wyrażenie $r_i = \frac{P(x_i)}{Q(x_i)}$ nazywa się wagą próbki (ang. *importance weight*). Powyższa średnia pomimo tego że została obliczona na próbkach z rozkładu Q nadal wymaga obliczenia prawdopodobieństwa próbek $P(x_i)$ – czyli w naszym zastosowaniu obliczenia pełnego softmaxa.

Wyraźmy rozkłady prawdopodobieństwa P i Q w postaci nieznormalizowanej:

$$P(x) = \frac{\widetilde{P}(x)}{Z_p} \qquad Q(x) = \frac{\widetilde{Q}(x)}{Z_q}$$

gdzie $Z_p = \int \widetilde{P}(x)dx$ i $Z_q = \int \widetilde{Q}(x)dx$ to stałe normalizujące rozkład do jedynki, nazywane sumami statystycznymi (ang. *partition functions*). W przypadku $\widetilde{P}$ z funkcji softmax będzie równe $e^{f_\theta(w)}$, a $Z_p = \sum_{v \in V} e^{f_\theta(v)}$. Wprowadźmy nowe oznaczenia rozkładów do wzoru:

$$\mathbb{E}_P[h(X)] = \int Q(x)h(x)\frac{P(x)}{Q(x)}dx = \int Q(x)h(x)\frac{\frac{\widetilde{P}(x)}{Z_p}}{\frac{\widetilde{Q}(x)}{Z_q}}dx = \frac{Z_q}{Z_p}\int Q(x)h(x)\frac{\widetilde{P}(x)}{\widetilde{Q}(x)}dx = \frac{Z_q}{Z_p}\mathbb{E}_Q\left[h(X)\frac{\widetilde{P}(X)}{\widetilde{Q}(X)}\right]$$

Co możemy przybliżyć poprzez pobranie k próbek z rozkładu Q i obliczenie:

$$\mathbb{E}_P[h(X)] \approx \frac{Z_q}{Z_p}\frac{1}{k}\sum_{i=1}^{k}\left[h(x_i)\frac{\widetilde{P}(x)}{\widetilde{Q}(x)}\right] = \frac{Z_q}{Z_p}\frac{1}{k}\sum_{i=1}^{k}[\widetilde{r}_i h(x_i)]$$

gdzie $\widetilde{r}_i = \frac{\widetilde{P}(x)}{\widetilde{Q}(x)}$ to waga próbki obliczona na podstawie nieznormalizowanych prawdopodobieństw. Nadal jednak musimy znać stałe normalizujące Z_p i Z_q do których obliczenia musimy zsumować/zcałkować po wszystkich wartościach zmiennych. Jak się jednak okazuje – możemy przybliżyć stałe normalizujące używając tych samych próbek! Dla ułatwienia dalszych przekształceń zapiszmy odwrócony ułamek $\frac{Z_p}{Z_q}$:

$$\frac{Z_p}{Z_q} = \frac{1}{Z_q}\int \widetilde{P}(x)dx$$

Zauważmy, że

$$Q(x) = \frac{\widetilde{Q}(x)}{Z_q} \qquad \Rightarrow \qquad Z_q = \frac{\widetilde{Q}(x)}{Q(x)}$$

podstawiając:

$$\frac{Z_p}{Z_q} = \frac{1}{Z_q}\int \widetilde{P}(x)dx = \int \frac{\widetilde{P}(x)}{\widetilde{Q}(x)}Q(x)dx = \mathbb{E}_Q\left[\frac{\widetilde{P}(x)}{\widetilde{Q}(x)}\right] \approx \frac{1}{k}\sum_{i=1}^{k}\widetilde{r}_i$$

Wykorzystując wyestymowany stosunek stałych normalizujących otrzymujemy:

$$\mathbb{E}_P[h(X)] \approx \frac{Z_q}{Z_p}\frac{1}{k}\sum_{i=1}^{k}[\widetilde{r}_i h(x_i)] \approx \frac{1}{\frac{1}{k}\sum_{j=1}^{k}\widetilde{r}_j}\frac{1}{k}\sum_{i=1}^{k}[\widetilde{r}_i h(x_i)] = \sum_{i=1}^{k}\left[\frac{\widetilde{r}_i}{\sum_{j=1}^{k}\widetilde{r}_j}h(x_i)\right]$$

Powyższa estymata jest zgodna i asymptotycznie nieobciążona[2], ponadto do jej obliczenia potrzebujemy jedynie obliczenia nieznormalizowanego prawdopodobieństwa kilku słów. Przekładając to na uczenie neuronowego modelu języka, potrzebujemy jedynie policzyć nieznormalizowaną wartość softmaxa dla $k << |V|$ wylosowanych słów. Zauważ, że efektem działania metody będzie to że tylko kilka wektorów wag softmaxa będzie miało niezerowe gradienty (tylko wektory służące do predykcji wylosowanych słów).

! Metoda próbowania ważonego z nieznormalizowanymi prawdopodobieństwami czasami nazywana jest obciążonym próbkowaniem ważonym (ang. *biased importance sampling*) lub próbkowaniem ważonym bez normalizacji.

Ile próbek należy losować i na jakie przyśpieszenie procesu uczenia możemy liczyć? Np. w pracy [3] użyto próbkowania ważonego do nauki neuronowego modelu języka opartego na sieciach rekurencyjnych używając $k = 8192$ próbek do estymowania gradientu, ale współdzielił wylosowane słowa w ramach jednej paczki (ang. *batch*). W opisywanym w pracy modelu doprowadziło to do 100-krotnego zredukowania liczby obliczeń. Należy

[2] Próbkowanie ważone daje estymaty nieobciążone, gdy pracuje na rozkładach prawdopodobieństwa. Tutaj użyliśmy próbkowania również do estymacji stałej normalizującej co daje estymatę obciążoną.

też dodać, że w praktyce zaobserwowano wzrost wariancji estymat otrzymanych próbkowaniem ważonym w trakcie uczenia modelu, dlatego czasami zwiększa się rozmiar próbki k wraz z kolejnymi epokami uczenia sieci.

Analogiczną do przedstawionej techniką przyśpieszania uczenia neuronowego modelu języka, jest estymacja poprzez rozróżnianie szumu (ang. *noise-contrastive estimation*), która redukuje problem do problemu klasyfikacji binarnej [6] – rozróżniania czy słowo jest z korpusu czy wygenerowane sztucznie. Istnieje związek pomiędzy tą techniką, a próbkowaniem ważonym: technika próbkowania ważonego może być przedstawiona jako wieloklasowa wersja estymacji rozróżniającej szum. Pomimo tego, że estymacja rozróżniająca szum była motywowana dużą wariancją próbkowania ważonego i propozycją bardziej stabilnej metody, próbkowanie ważone jest preferowane przez niektórych badaczy i praktyków oraz zostało wykorzystane do stworzenia silnych modeli języka [3].

3.3.2 Hierarchiczny softmax

W poprzednim rozdziale opisaliśmy technikę przyśpieszania nauki modelu języka, jednak nie modyfikującą warstwy softmax w trakcie predykcji. W szczególności, gdy chcesz przypisać prawdopodobieństwo do danego zdania – nadal musisz za każdym razem ewaluować całą warstwę softmax. Opisana w tym podrozdziale technika hierarchicznego softmax'a pozwala na przypisanie prawdopodobieństwa słowu przy ewaluacji $O(\log_2 |V|)$ prawdopodobieństw (lub nawet mniej!).

Dwuwarstwowy softmax

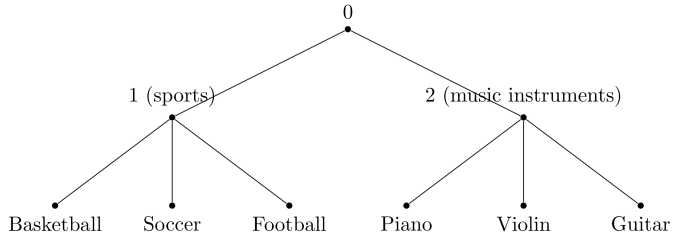
W przypadku problemów w rozwiązaniu dużego zadania, naturalna wydaje się próba podziału go na kilka mniejszych. Skoro trudność polega na długim czasie obliczeń który jest spowodowany potrzebą obliczenia wszystkich $|V|$ prawdopodobieństw, nasuwa się pomysł wprowadzenia etapu pośredniego. Model najpierw może przewidzieć klasę słowa, a dopiero potem wybrać spośród słów w danej klasie. Zakładając, że liczba klas k jest dużo mniejsza od $|V|$ może to doprowadzić do znacznego przyśpieszenia działania systemu. Zamiast obliczać $|V|$ prawdopodobieństw, najpierw obliczamy k prawdopodobieństw przypisań do klas, a potem średnio $\frac{|V|}{k}$ prawdopodobieństw przypisanych do słów w danej klasie. Np. przy k=1000 i $|V| = 100000$ daje to ok. 90-krotne przyśpieszenie.

Dwuwarstwowy softmax (ang. *two-layered hierarchical softmax*) definiuje prawdopodobieństwo słowa w następujący sposób:

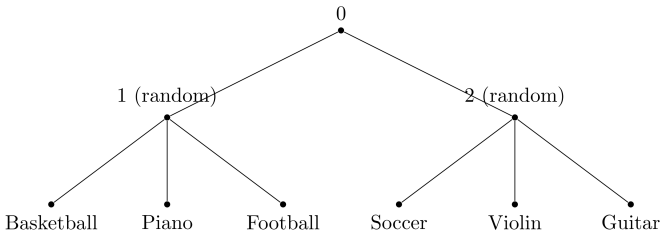
$$P_\theta(w|c) = P_\theta(w|c, C(w))P_\theta(C(w)|c)$$

Reprezentacja warstwy ukrytej reprezentująca kontekst c jest użyta jako wejście do zwykłego softmaxa, którego wyjściami są klasy przypisane do słów. Następnie każda z klas ma swojego własnego softmaxa, który na podstawie (tej samej) reprezentacji ukrytej c przewiduje jedno ze słów które mają daną klasę.

Takie klasyfikatory można przedstawić w postaci dwu-warstwowej hierarchii drzewiastej w której wierzchołki są klasyfikatorami softmax, a liście są odpowiednimi słowami. Przykład takiej hierarchii jest pokazany na rysunku 3.3. W trakcie uczenia softmaxowi na szczycie hierarchii pokazujemy po prostu klasę prawidłowego słowa dla którego standardowo liczymy gradient. Podobnie softmaxowi w niższej warstwie pokazujemy prawidłowe słowo i liczymy gradient bez żadnych zmian. Jedną zmianą jest to że do warstwy liczącej



Rysunek 3.3: Przykład struktury dwuwastwowego softmaxa dla 6 słów. [4]



Rysunek 3.4: Przykład struktury dwuwastwowego softmaxa dla 6 słów, która będzie trudna do nauki. [4]

reprezentację ukrytą c dochodzą dwa gradienty (z dwóch wybranych softmaxów), które po prostu sumujemy.

W czasie predykcji, jeśli chcemy obliczyć prawdopodobieństwo konkretnego słowa wystarczy uruchomić klasyfikatory na ścieżce do tego słowa i wymnożyć uzyskane dwie liczby/prawdopodobieństwa.

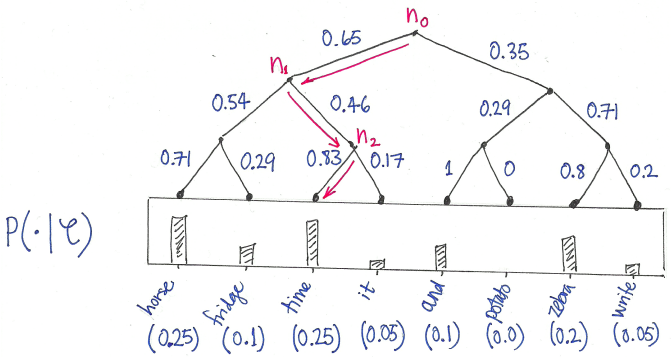
Głównym problemem takiego podejścia jest dobre zdefiniowanie klas, które będą użyte w pierwszej warstwie hierarchii. Nietrudno zgadnąć, że dwuwarstwowy softmax pokazany na rysunku 3.4 będzie działał gorzej niż ten z rysunku 3.3. W pierwszym przypadku softmax na szczycie hierarchii (wierzchołek z etykietą 0) musi przewidzieć czy kolejne słowo będzie dot. sportu czy muzyki. Ma więc intuicyjnie prostsze zadanie niż oryginalny softmax, bo nie musi się zastanawiać które konkretnie słowo będzie użyte, a tylko czego będzie dotyczyć. W drugim przypadku klasyfikator 0 musi implicite wiedzieć że kolejne słowo będzie jakimś konkretnym o muzyce np. „Piano" i odesłać nas do konkretnego niższego klasyfikatora.

Pewnym pomysłem jest wykorzystanie do budowy tej hierarchicznej struktury grupowania Browna z pożądaną liczbą grup/klas, jednak jego koszt obliczeniowy jest dość duży. Powrócimy do tego problemu w kolejnym rozdziale, ale czasami stosuje się po prostu losowy podział na grupy.

Hierarchiczny softmax

W poprzednim rozdziale opisaliśmy technikę w której podzieliliśmy softmaxa na dwie warstwy. Jednak.. dlaczego tylko dwie? Czy stworzenie struktury z większą liczbą warstw nie ograniczy dalej liczby prawdopodobieństw i nie przyśpieszy działania modelu? Oczywiście tak, a pewnym granicznym przypadkiem dodawania tych warstw jest softmax hierarchiczny posiadający hierarchię w której klasyfikator na każdym poziomie ma podjęcia tylko binarną decyzję: czy powinno się pójść w lewą czy w prawą odnogę drzewa.

Przykład takiej hierarchii znajduje się na rysunku 3.5, gdzie każdy węzeł zawiera



Rysunek 3.5: Przykład struktury hierarchicznego softmaxa [7].

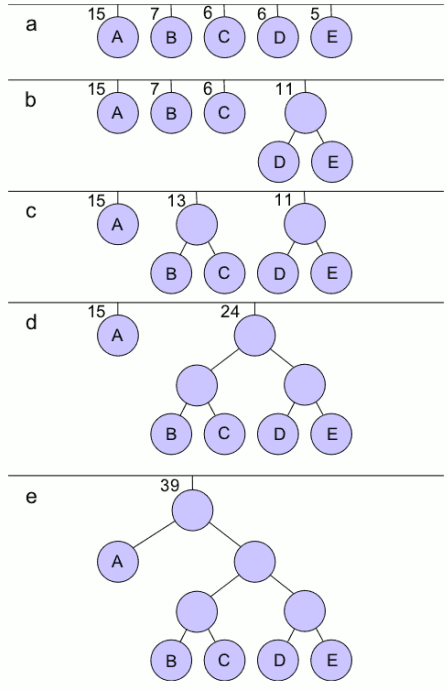
regresję logistyczną decydującą o tym którą odnogę trzeba wybrać (regresja logistyczna jest binarnym odpowiednikiem softmaxa). Aby policzyć prawdopodobieństwo danego słowa należy – tak jak poprzednio – wymnożyć prawdopodobieństwa klasyfikatorów znajdujących się na odpowiedniej ścieżce.

Zwróć uwagę, że w przeciwieństwie do dwuwarstwowego softmaxa gdzie mieliśmy więcej parametrów niż oryginalny softmax (poprzez klasyfikator na szczycie hierarchii) w przypadku dokładnie zbalansowanego drzewa binarnego liczba parametrów hierarchicznego softmaxa jest taka sama jak oryginalnego. Obliczenie całego rozkładu prawdopodobieństwa wszystkich słów wymaga wtedy takiej samej liczby operacji (ewaluacja wszystkich klasyfikatorów), ale już znalezienie prawdopodobieństwa konkretnego słowa wymaga tylko $O(\log_2 |V|)$ ewaluacji klasyfikatorów. Podobnie popranie próbki z rozkładu modelowanego przez hierarchiczny softmax wymaga tylko $O(\log_2 |V|)$ operacji – przechodzimy od szczuty hierarchii w dół poprzez rzucanie monetą ważoną prawdopodobieństwami zwracanymi przez wierzchołek. Możemy też przyśpieszyć znajdowanie najbardziej prawdopodobnego słowa albo poprzez zastosowanie algorytmu zachłannego $O(\log_2 |V|)$ lub któregoś z algorytmów przeszukiwania drzew z odcięciami.

Pozostaje oczywiście pytanie: jak stworzyć drzewo hierarchii?

- losowo – najprostszy sposób, który daje gorszą jakość działania niż oryginalny softmax, ale warto podkreślić że nie dużo gorszą
- optymalizując szybkość predykcji i uczenia. Możemy skonstruować drzewo wynikające z kodowania Huffmana. W taki drzewie słowa, które pojawiają się często w korpusie mają przypisane krótkie ścieżki, podczas gdy słowa których będziemy rzadko potrzebować będą miały przypisane dłuższe ścieżki. Powstałe drzewo nie jest zbalansowanym drzewem binarnym, które oferuje złożoności $O(\log_2 |V|)$, jednak oferowana jest lepsza złożoność $O(\log_2 PP(Unigram))$ gdzie $PP(Unigram)$ to nieokreśloność modelu unigramowego. Ponieważ nieokreśloność zawsze jest mniejsza równa niż $|V|$ oferowana złożoność jest lepsza.

  Aby utworzyć drzewo Huffmana potrzebujemy stworzyć listę słów wraz z ich licznościami. Następnie pobieramy z listy dwa słowa z najmniejszą licznością i łączymy je w jeden wierzchołek. Nowo utworzony wierzchołek jest dodawany na listę z licznością równą sumie liczności jego dzieci. Dopóki na liście niepozostanie jeden wierzchołek, powyższa operacja jest powtarzana. Kolejne kroki działania algorytmu są pokazane na rysunku 3.6.
- starając się optymalizować jakość predykcji. Podobnie jak w przypadku softmaxa



Rysunek 3.6: Kolejne kroki tworzenia drzewa Huffmana.

dwuwarstwowego, zła struktura drzewa może doprowadzić do pogorszenia jakości predykcji w stosunku do oryginalnego softmaxa. Do konstrukcji drzewa można wykorzystać informacje o podobieństwie słów ze specjalnych słowników lub innych, ręcznie przygotowanych zasobów lingwistycznych (np. WordNet). Alternatywnie można wykorzystać jakieś wersje grupowania Browna, specjalnie ukierunkowane na zwracanie bardziej zbalansowanych drzew binarnych. Można też wykorzystać grupowanie hierarchiczne na zanurzeniach słów (które jednak najpierw trzeba jakoś nauczyć).

Korzystanie z hierarchicznego softmaxa sprawia, że neuronowe modele języka są znacznie szybsze – nawet do 300 razy, jednak zwykle za cenę delikatnie niższej jakości. Poświęcenie dodatkowego czasu na dobre zaprojektowanie hierarchii słów daje jednak możliwość uzyskania takich samych lub lepszych rezultatów niż oryginalny softmax [5].

Dodatki

Materiały powtórkowe

Opis neuronowego modelu języka znajduje się w pracy [1], a opis metod przyśpieszania predykcji modelu można odnaleźć w rozdziale 12.4.3 [2].

Materiały dla chętnych

Dla zainteresowanych polecam opis uogólnienia hierarchicznego softmaxa do klasyfikacji ekstremalnej [8] z gwarancjami teoretycznymi. Made in Poland ;)

Bibliografia

[1] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, i Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, 2003.

[2] Ian Goodfellow, Yoshua Bengio, i Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[3] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, i Yonghui Wu. Exploring the limits of language modeling, 2016. URL `https://arxiv.org/pdf/1602.02410.pdf`.

[4] Lei Mao. Hierarchical softmax, 2019. URL `https://leimao.github.io/article/Hierarchical-Softmax/`.

[5] Andriy Mnih i Geoffrey E Hinton. A scalable hierarchical distributed language model. In *Advances in neural information processing systems*, pages 1081–1088, 2009.

[6] Andriy Mnih i Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, page 419–426, Madison, WI, USA, 2012. Omnipress. ISBN 9781450312851.

[7] Benjamin Wilson. Hierarchical softmax, 2017. URL `http://building-babylon.net/2017/08/01/hierarchical-softmax/`.

[8] Marek Wydmuch, Kalina Jasinska, Mikhail Kuznetsov, Róbert Busa-Fekete, i Krzysztof Dembczyński. A no-regret generalization of hierarchical softmax to extreme multi-label classification. In *Proceedings of NeurIPS 2018*, 2018.