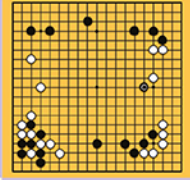
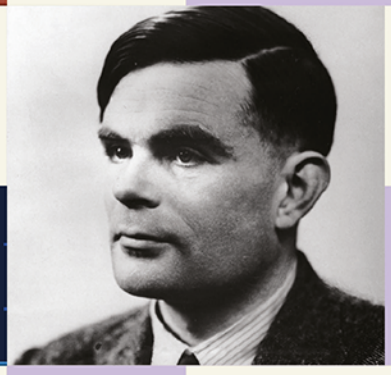
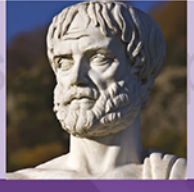






Stuart **Russell**

Peter **Norvig**

Sztuczna inteligencja

Nowe spojrzenie

Wydanie IV. Tom 2

Helion

Tytuł oryginału: Artificial Intelligence: A Modern Approach, 4th Edition

Tłumaczenie: Andrzej Grażyński

ISBN: 978-83-283-7773-8

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/szti42
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

SPIS TREŚCI

ROZDZIAŁ 24

GŁĘBOKIE UCZENIE W PRZETWARZANIU JĘZYKA NATURALNEGO

Głębokie sieci neuronowe, które zrewolucjonizowały uczenie maszynowe, nie mogły rzecz jasna ominąć kluczowej gałęzi tego uczenia, jaką jest przetwarzanie języka naturalnego. W tym rozdziale pokazujemy, jak przyczyniają się one do uchwycenia struktury języka naturalnego, zapewnienia jego płynności i wydatnego usprawnienia wielu innych zadań.

W rozdziale 23. omawialiśmy kluczowe elementy języka naturalnego, między innymi gramatykę i semantykę. Systemy oparte na parsingu i analizie semantycznej okazały się znakomite w realizacji wielu zadań, lecz ich wydajność jest ograniczona ze względu na stopień złożoności języków naturalnych i zachodzące w rzeczywistym świecie zjawiska i zdarzenia wpływające na ich ewolucję. Dysponując ogromną ilością dostępnego tekstu w formie czytelnej dla maszyn, moglibyśmy wysunąć hipotezę, że podejścia oparte na uczeniu sterowanym danymi mogą być bardziej efektywne; tę właśnie hipotezę przeanalizujemy dokładniej w tym rozdziale, posługując się narzędziami wykorzystywanymi przez systemy głębokiego uczenia, omawiane w rozdziale 21.

Rozpoczniemy od pokazania (w podrozdziale 24.1), jak można usprawnić uczenie maszynowe, traktując słowa języka jako punkty w wysokowymiarowej przestrzeni, a nie jako niepodzielne (atomowe) wartości. Przedmiotem podrozdziału 24.2 jest zastosowanie rekurencyjnych sieci neuronowych do rozpoznawania znaczenia i długodystansowego kontekstu w miarę sekwencyjnego przetwarzania tekstu. Treść podrozdziału 24.3 koncentruje się na tłumaczeniach maszynowych, jako jednym z najbardziej pomyślnych obszarów zastosowań głębokiego uczenia w NLP. Podrozdziały 24.4 i 24.5 poświęcone są modelom, które można trenować za pomocą dużych porcji nieetykietowanego tekstu, a następnie wykorzystywać do realizacji specyficznych zadań, ze zdumiewającym nieraz powodzeniem. Na zakończenie, w podrozdziale 24.6, przedstawiamy obecny stan wiedzy w zakresie zastosowań głębokiego uczenia w NLP i przewidywane kierunki rozwoju w tej dziedzinie.

24.1. Embeddingi słów

Potrzebujemy takiej reprezentacji słów, która nie wymaga ręcznej inżynierii cech, ale umożliwia generalizację powiązań między słowami — powiązań syntaktycznych („bezbarwny” i „idealny” to przymiotniki), semantycznych („kot” i „kotek” to przedstawiciele rodziny kotowatych), tematycznych („słonecznie” i „gołoledź” to zjawiska pogodowe), wyrażających sentyment („fantastyczny” to sentyment przeciwny do „żenujący”) itp.

Jak więc mamy zakodować słowo w formie wektora **x** jako wejściowego dla sieci neuronowej? Mógłby to być **wektor z gorącą jedynką** (ang. *one-hot vector*), czyli taki, w którym *i*-ty element równy jest 1, a pozostałe są zerowe, gdzie *i* jest numerem pozycji słowa w słowniku (patrz sekcja 21.2.1). Taka jednak reprezentacja w żaden sposób nie odzwierciedla podobieństw między słowami.

Wzorując się na maksymie językoznawcy Johna R. Firtha (1957) „Poznasz słowo po jego towarzystwie”[1] możemy reprezentować dane słowo w formie wektora, którego *k*-ty element jest liczbą całkowitą odzwierciedlającą jego występowanie w *n*-gramach. Numerujemy wszystkie możliwe *n*-gramy (dla ustalonego *n*), a w wektorze reprezentującym dane słowo *k*-ty element równy jest liczbie wystąpień tego słowa w *k*-tym *n*-gramie. Operowanie „surowymi” licznikami byłoby jednak nieporęczne — dla 100 000 słów liczba możliwych 5-gramów wynosi $100\,000^5 = 10^{25}$, choć znakomita większość wektorów w tej 10^{25}-wymiarowej przestrzeni to wektory rzadkie, w których większość elementów jest zerowa. Bardziej praktyczną generalizację otrzymamy, posługując się wektorami, których liczba elementów jest znacznie mniejsza, zredukowana do (powiedzmy) kilkuset. Taki zredukowany wektor, reprezentujący wystąpienie danego słowa w określonym kontekście, nazywamy jest (z angielska) **embeddingiem**[2] tego słowa. Embeddingi poszczególnych słów budowane są automatycznie w procesie uczenia, na podstawie danych (w jaki sposób — pokażemy to w dalszej części rozdziału). Fizycznie są one *wektorami liczbowymi* o ustalonym rozmiarze, na przykład

„aardvark” = [−0,7, +0,2, −3,2, ...]
„abacus” = [+0,5, +0,9, −1,3, ...]
„zyzzyzus”[3] = [−0,1, +0,8, −0,4, ...]

Co do konkretnych wartości elementów tych wektorów, to nie mają one samoistnego znaczenia, ale wynikają z wymogu, by podobne słowa transformowały się na podobne wektory. Zasadę tę ilustruje rysunek 24.1, na którym widać, jak embeddingi słów pokrewnych grupują się w odrębne klastery, zawierające nazwy państw, relacji rodzinnych, środków transportu i żywności.

Jak się okazuje, z powodów, których do końca nie rozumiemy, embeddingi słów wykazują jeszcze inne właściwości oprócz bliskości do podobnych słów. Załóżmy na przykład, że poszukujemy embeddingu **A**, odzwierciedlającego słowo „Ateny”, i embeddingu **B**, odzwierciedlającego słowo „Grecja”. Intuicyjnie można podejrzewać, że wektor stanowiący różnicę embeddingów **B** – **A** będzie *zakodowaną postacią związku* „państwo-stolica” między słowami. A więc innym parom słów pozostających w takim związku — „Francja-Paryż”, „Rosja-Moskwa”, „Zambia-Lusaka” — powinny odpowiadać (zasadniczo) takie same wektory różnicowe.

Możemy wykorzystywać tę własność do rozwiązywania problemów polegających na analogiach związków między słowami, na przykład „»Ateny« i »Grecja« pozostają ze sobą w takim samym związku, jak »Oslo« i [co]?”. Niech **C** będzie embeddingiem słowa „Oslo”, a **D** niech oznacza embedding szukanego słowa; jako że domniemamy równość

$$\mathbf{B} - \mathbf{A} = \mathbf{D} - \mathbf{C}$$

więc embedding szukanego słowa określony będzie przez relację

$$\mathbf{D} = \mathbf{C} + (\mathbf{B} - \mathbf{A})$$

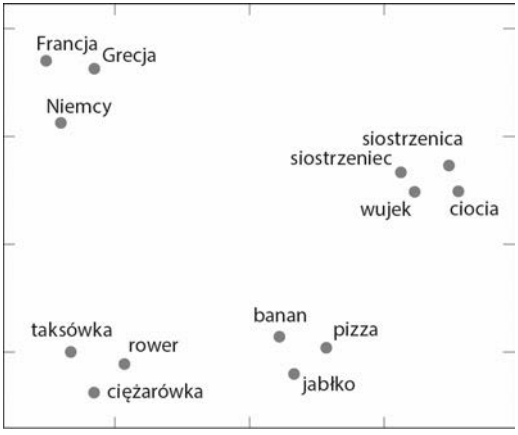
I gdy już obliczymy embedding **D**, zauważymy, że najbliższym z odpowiadających mu słów jest „Norwegia”. W tabeli 24.1 pokazujemy przykłady funkcjonowania tej arytmetyki wektorowej dla kilku innych relacji.

---

[1] „You shall know a word by the company it keeps” — *przyp. tłum.*

[2] W języku polskim termin ten jest często tłumaczony zarówno jako czynność „osadzania” słowa, przez analogię do „łączenia i osadzania obiektów” (ang. OLE — *Object Linking and Embedding*), jak i jako „osadzenie” będące rezultatem tej czynności. Tymczasem w znaczeniu OLE „osadzanie” i „osadzenie” odnosi się w istocie do *integrowania obiektów* (na przykład włączania obrazka do dokumentu tekstowego), natomiast *embedding* słowa jest wektorem stanowiącym wynik pewnej jego *transformacji* (z postaci znakowej na wektor liczbowy), niekojarzącej się z jakimś integrowaniem. Ponadto z perspektywy języka kontrowersyjne wydaje się użycie rzeczownika *odczasownikowego* na określenie obiektu, jakim jest wspomniany wektor. W związku z tym pozostawiam występujące w oryginale słowo „embedding” na oznaczenie owego wektora, zaś samą czynność jego tworzenia nazywał będę „embedowaniem” — *przyp. tłum.*

[3] Patrz *https://hrvwiki.net/Wikipedia_Zyzzyzus* — *przyp. tłum.*



RYSUNEK 24.1. Embeddingi słów wyliczone przez algorytm GloVe wytrenowany na zbiorze liczącym 6 miliardów słów. Na rysunku widoczny jest efekt rzutowania 100-wymiarowych wektorów słów na dwuwymiarową płaszczyznę; wektory reprezentujące podobne słowa znajdują się blisko siebie

TABELA 24.1. W niektórych przypadkach model embeddingów może dostarczać odpowiedzi na pytania w rodzaju „**A** ma się tak do **B**, jak **C** do [czego]?" za pomocą prostej arytmetyki wektorowej: dysponując embeddingami słów **A**, **B** i **C**, możemy wyliczyć nieznany embedding **D** z zależności **D** = **C** + (**B** – **A**) i następnie poszukiwać słowa, którego embedding jest najbardziej zbliżony do **D** (odpowiedź widoczna w kolumnie **D** wyliczona została automatycznie przez model, opisy w kolumnie *Relacja* zostały dodane ręcznie). Zaadaptowano z: Mikolov i in., (2013, 2014)

A	**B**	**C**	**D** = **C** + (**B** – **A**)	Relacja
Ateny	Grecja	Oslo	Norwegia	*Stolica*
Astana	Kazachstan	Harare	Zimbabwe	*Stolica*
Angola	kwanza	Iran	rial	*Waluta*
Miedź	Cu	Złoto	Au	*Symbol pierwiastka*
Microsoft	Windows	Google	Android	*System operacyjny*
Nowy Jork	New York Times	Baltimore	Baltimore Sun	*Gazeta*
Berlusconi	Silvio	Obama	Barack	*Imię*
Szwajcaria	Szwajcar	Kambodża	Kambodżanin	*Narodowość*
Einstein	naukowiec	Picasso	malarz	*Profesja*
brat	siostra	wnuczek	wnuczka	*Pokrewieństwo*
Chicago	Illinois	Stockton	Kalifornia	*Stan*
możliwe	niemożliwe	etyczne	nieetyczne	*Negacja*
mysz	myszy	dolar	dolary	*Liczba mnoga*
łatwe	najłatwiejsze	szczęśliwy	najszczęśliwszy	*Stopień najwyższy*
idę	szedłem	pływam	pływałem	*Czas przeszły*

Nie ma jednak gwarancji, że konkretny algorytm konstruowania embeddingów, uruchomiony dla konkretnego korpusu, prawidłowo uchwyci konkretne relacje semantyczne. Technika embeddingów stała się popularna bynajmniej nie dlatego, by skrywała w sobie jakąś magiczną zdolność bezbłędnego wychwytywania analogii relacyjnych między słowami korpusu, ale po prostu dlatego, że udowodniła swą wartość jako znakomita reprezentacja słów dla zadań NLP typu *downstream*, czyli operowania na gotowych tekstach: odpowiadania na pytania, tłumaczenia czy generowania streszczeń.

Wykorzystywanie embeddingów zamiast reprezentacji „z gorącą jedynką" okazuje się wielce pomocne w odniesieniu do niemal wszelkich zastosowań głębokiego uczenia w zadaniach NLP. W przypadku bowiem wielu z tych zadań można posługiwać się generycznymi **wstępnie wytrenowanymi** zbiorami embeddingów, oferowanymi przez wielu dostawców na potrzeby konkretnych zastosowań i pozwalającymi zaoszczędzić sporo czasu. Gdy piszemy te słowa, do najbardziej popularnych słowników tego typu należą WORD2VEC, GloVe (od *Global Vectors*) i FASTTEXT, oferujące embeddingi słów w 157 językach. Powrócimy do tej kwestii w sekcji 24.5.1.

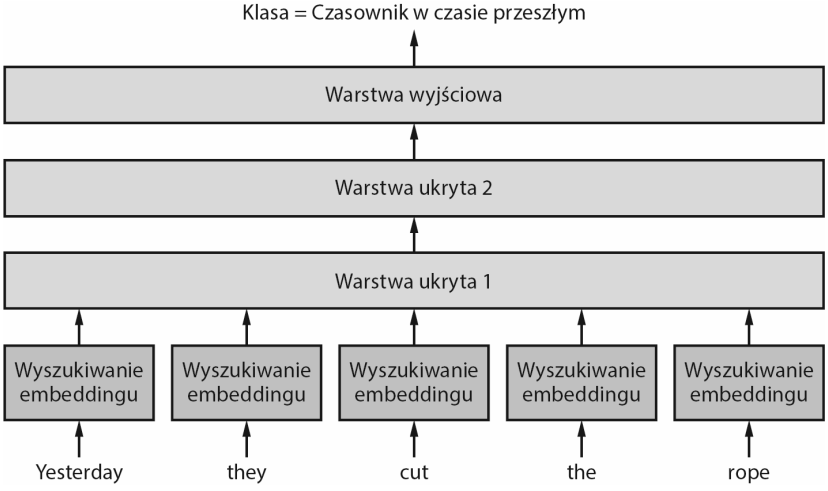
Oczywiście możliwe jest samodzielne trenowanie embeddingów — może się ono odbywać niejako przy okazji trenowania sieci neuronowej pod kątem określonych zadań, dla których przydatność wspomnianych generycznych embeddingów jest ograniczona czy wręcz żadna. Weźmy na przykład tagowanie części mowy, opisywane w sekcji 23.1.6, polegające na przewidywaniu przynależności poszczególnych słów zdania do określonych kategorii gramatycznych. Mimo iż jest to zadanie stosunkowo proste, to bynajmniej nie można uważać go za banalne, ponieważ dla wielu słów nie istnieje jedna „prawidłowa" kategoria — popularne słowo *cut* może być czasownikiem w czasie teraźniejszym — przechodnim (*I cut my hair* — ostrzygłem się) lub nieprzechodnim (*He cut a lecture* — opuścił wykład), czasownikiem w czasie przeszłym prostym (*he cut off smoking* — on rzucił palenie), bezokolicznikiem (ciąć, ścinać, odcinać, itp.), imiesłowem czasu przeszłego (*I was cut off while talking* — przerwano mi połączenie), przymiotnikiem (ścięty, rozcięty) lub rzeczownikiem (*price cut* — obniżka ceny). Jeśli pobliski przysłówek czasu odnosi się do przeszłości, sugeruje to, że to konkretne wystąpienie *cut* jest czasownikiem w czasie przeszłym; a zatem możemy mieć nadzieję, że embedding uchwyci aspekt przysłówkowy, odnoszący się do przeszłości.

Tagowanie części mowy to bardzo wdzięczny temat wprowadzający w zastosowania głębokiego uczenia w NLP, jest bowiem wolny od komplikacji charakterystycznej dla problemów bardziej zaawansowanych, na przykład odpowiadania na pytania, którym to problemem zajmiemy się w sekcji 24.5.3. Dla danego korpusu zdań etykietowanych tagami części mowy, możemy realizować równoczesne uczenie dwóch rzeczy: parametrów dla embeddingów słów i rozpoznawania części mowy. Proces ten przebiega w następujących krokach:

1. Wybieramy nieparzystą liczbę w jako szerokość tzw. okna przewidywania, służącego do tagowania słów; zwykle przyjmuje się $w = 5$, co oznacza, że tag danego słowa przewidywany jest na podstawie tegoż słowa oraz dwóch słów sąsiadujących z nim z lewej i dwóch z prawej strony. Dzielimy każde zdanie korpusu na nakładające się okna o szerokości w; każde z tych okien traktowane jest jako przykład treningowy w postaci ciągu w słów, etykietowany przewidywaną kategorią POS środkowego słowa.
2. Tworzymy słownik wszystkich unikalnych słów-tokenów występujących w danych treningowych więcej niż (powiedzmy) 5 razy; oznaczmy przez v liczbę wszystkich słów w tak utworzonym słowniku.
3. Sortujemy zawartość tego słownika w dowolnej kolejności (na przykład alfabetycznej).
4. Wybieramy d jako rozmiar embeddingu dla każdego słowa.
5. Tworzymy macierz wagową **E** o rozmiarze v wierszy × d kolumn; i-ty wiersz tej macierzy stanie się (po zakończeniu uczenia) embeddingiem i-tego słowa w słowniku. Inicjujemy tę macierz losowymi wartościami (albo, jeśli dysponujemy wstępnie wytrenowanymi embeddingami, zapisujemy te embeddingi jako wiersze macierzy).

6. Budujemy sieć neuronową, której wyjściem będzie przewidywana etykieta części mowy, jak pokazaliśmy na rysunku 24.2. Pierwsza warstwa tej sieci składa się z *w* kopii wspomnianej macierzy kontekstów. W sieci tej występują ponadto dwie warstwy ukryte $\mathbf{z}_1$ i $\mathbf{z}_2$, z macierzami wagowymi (odpowiednio) $\mathbf{W}_1$ i $\mathbf{W}_2$, po których następuje warstwa `softmax` zwracająca rozkład prawdopodobieństwa $\hat{\boldsymbol{y}}$ możliwych kategorii wyjściowych:

$$\mathbf{z}_1 = \sigma(\mathbf{W}_1 \mathbf{x})$$
$$\mathbf{z}_2 = \sigma(\mathbf{W}_2 \mathbf{z}_1)$$
$$\hat{\boldsymbol{y}} = \mathtt{softmax}(\mathbf{W}_{out} \mathbf{z}_2)$$



RYSUNEK 24.2. Model ze sprzężeniem w przód dla tagowania części mowy. Na podstawie składającego się z 5 słów okna jako wejścia model ten przewiduje kategorię gramatyczną (część mowy) dla środkowego słowa — w tym przypadku *cut*. Model zdolny jest do uwzględniania pozycji słowa w zdaniu, ponieważ każdy z 5 embeddingów mnożony jest przez różne części pierwszej ukrytej warstwy. Wartości parametrów dla embeddingów i dla trzech warstw są wynikiem zrównoleglonego treningu (*Yesterday they cut the rope* — „wczoraj [oni] przecięli linę")

7. Aby zakodować sekwencję *w* słów w postaci wektora wejściowego, po prostu wyszukujemy embeddingi dla tych słów i konkatenujemy je, tworząc wektor $\mathbf{x}$ złożony z $w \cdot d$ liczb rzeczywistych. Mimo iż embedding danego słowa nie zależy od pozycji, jaką słowo to zajmuje w sekwencji *w* słów, to od pozycji tej zależny jest *mnożnik* w warstwie $\mathbf{z}_1$, przez który mnożony jest wspomniany embedding; w ten oto sposób w wyniku kodowania słów uwzględniana jest ich relatywna pozycja w sekwencji.
8. Trenujemy macierz $\mathbf{E}$ oraz macierze wagowe $\mathbf{W}_1$, $\mathbf{W}_2$ i $\mathbf{W}_{out}$, wykorzystując metodę spadku gradientowego. Gdy wszystko przebiegnie pomyślnie, środkowe słowo *cut* zaetykietowane zostanie jako czasownik w czasie przeszłym (*past tense verb*) — za sprawą innych słów w oknie przewidywania, między innymi określenia temporalnego *yesterday* („wczoraj") oraz zaimka *they* („oni", ew. „one") w 3. osobie liczby mnogiej, występującego bezpośrednio przed *cut*.

Alternatywą dla modelu embeddingów może być **model znakowy**, w ramach którego informacją wejściową jest sekwencja znaków, z których każdy kodowany jest jako wektor „z gorącą jedynką". Model taki umożliwia uczenie się sposobów formowania znaków w słowa. Zdecydowanie jednak większość badań z zakresu NLP realizowana jest raczej na poziomie słów niż na poziomie oddzielnych znaków.

24.2. Rekurencyjne sieci neuronowe w NLP

Dysponujemy już dobrymi reprezentacjami pojedynczych, izolowanych słów, ale przecież język to uporządkowane sekwencje słów, dla których istotny jest **kontekst** wynikający ze słów w bliższym lub dalszym otoczeniu. Dla prostych zadań NLP, takich jak tagowanie części mowy, wystarczający okazuje się zwykle kontekst wyznaczany przez okna przewidywań o ustalonym, niewielkim rozmiarze (jak pięcioelementowe okna z poprzedniej sekcji).

Zadania bardziej ambitne, jak odpowiadanie na pytania czy rozwiązywanie odwołań (ang. *reference resolution*), mogą jednak wymagać uwzględnienia znacznie szerszego kontekstu. Na przykład w zdaniu *Eduardo told me that Miguel was very sick so I took* **him** *to the hospital* („Eduardo powiedział mi, że Miguel jest bardzo chory, więc zabrałam go do szpitala") rozpoznanie, że **him** odnosi się do *Miguel*, a nie do *Eduardo*, wymaga kontekstu rozciągającego się na wszystkie słowa 15-wyrazowego zdania.

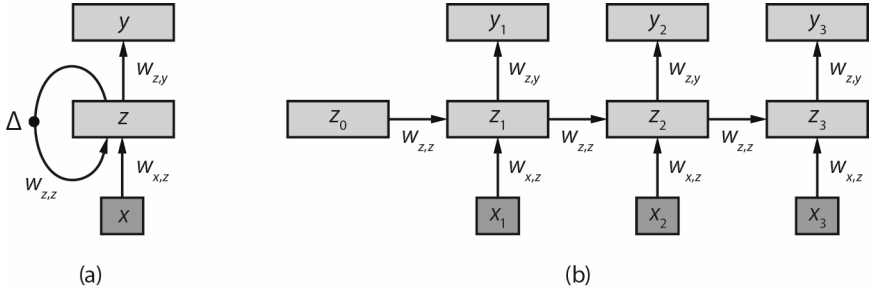
24.2.1. Modele językowe wykorzystujące sieci RNN

Rozpoczniemy od problemu stworzenia **modelu językowego** zapewniającego wystarczający kontekst. Jak pamiętamy, model językowy to w istocie rozkład prawdopodobieństwa względem sekwencji słów, pozwalający na przewidywanie kolejnego słowa w tekście na podstawie wszystkich poprzednich słów. Modele takie stanowią integralne elementy rozwiązań wielu złożonych problemów NLP.

Budowanie modelu językowego w oparciu zarówno o *n*-gramy (patrz podrozdział 23.1), jak i o sieci ze sprzężeniem w przód (patrz podrozdział 21.1) z oknem przewidywania o ustalonej szerokości wiąże się z tą trudnością, że albo szerokość ta jest zbyt mała dla wymaganego kontekstu, albo liczba wymaganych parametrów modelu jest nadmiernie duża (jedno zresztą nie wyklucza drugiego).

Ponadto sieci ze sprzężeniem w przód cechują się swoistą **asymetrią**: czegokolwiek nauczy się sieć odnośnie (powiedzmy) słowa *him* występującego na 12. pozycji w zdaniu, okaże się to raczej nieprzydatne w odniesieniu do takiego samego słowa występującego w innym miejscu tegoż zdania; powodem tego jest oczywiście różnica wag na różnych pozycjach.

W podrozdziale 21.6 omawialiśmy podstawy **rekurencyjnych sieci neuronowych** (RNN), zaprojektowanych z myślą o przetwarzaniu szeregów czasowych, po jednym elemencie szeregu w danej chwili. Natychmiast nasuwa się analogia z przetwarzaniem tekstu na poziomie kolejnych słów — analogia sugerująca przydatność sieci RNN do tego celu. Rysunek 24.3, przedstawiający schemat podstawowej sieci RNN, jest repliką rysunku 21.8.



RYSUNEK 24.3. (a) Schematyczny diagram podstawowej sieci RNN, w której ukryta warstwa **z** zawiera rekurencyjne połączenia; symbol Δ reprezentuje opóźnienie. Każdy wektor wejściowy **x** jest embeddingiem następnego słowa w zdaniu. (b) Ta sama sieć rozwinięta na trzy kroki czasowe, do postaci sieci ze sprzężeniem w przód. Zauważmy, że wagi współdzielone są przez wszystkie kroki czasowe

W modelu językowym opartym na sieci RNN każde słowo wejściowe zakodowane zostaje w postaci embeddingu $\mathbf{x}_t$. Ukryta warstwa $\mathbf{z}_t$ tej sieci przekazuje wejście z jednego kroku czasowego do następnego. Jesteśmy zainteresowani klasyfikacją wieloklasową, rolę klas pełnią słowa zawarte w słowniku. Wyjście $\mathbf{y}_t$ jest więc rozkładem prawdopodobieństwa `softmax` względem możliwych postaci następnego słowa w zdaniu.

Architektura RNN zapewnia rozwiązanie problemu zbyt dużej liczby parametrów: ich liczba w macierzach wagowych $w_{*,z,z}$, $w_{*,x,z}$ i $w_{*,z,y}$ pozostaje stała, niezależnie od liczby słów — czyli jest rzędu $O(1)$. Kontrastuje to z sieciami ze sprzężeniem w przód, wymagającymi $O(n)$ parametrów, oraz modelami n-gramowymi, w których liczba wymaganych parametrów jest rzędu $O(v^n)$, gdzie v oznacza rozmiar słownika.

Architektura RNN rozwiązuje także problem asymetrii, ponieważ wagi są takie same dla wszystkich pozycji słów w zdaniu.

W niektórych przypadkach architektura RNN może także rozwiązywać problem ograniczenia kontekstu. Teoretycznie nie istnieje ograniczenie na długość dystansu, na jaki model może spoglądać wstecz, na wcześniejsze słowa wejściowe. Każda aktualizacja warstwy wejściowej $\mathbf{z}_t$ ma dostęp zarówno do bieżącego słowa wejściowego $\mathbf{x}_t$, jak i poprzedniej warstwy ukrytej $\mathbf{z}_{t-1}$, co oznacza, że informacja na temat dowolnego słowa wejściowego może być przechowywana w warstwie ukrytej w nieskończoność, oraz kopiowana (być może wraz z odpowiednią modyfikacją) między danym krokiem czasowym i krokiem następnym. W praktyce oczywiście owa „nieskończoność" ograniczona jest rozmiarem dostępnej pamięci dla warstwy $\mathbf{z}$, nie jest więc możliwe przechowywanie wszystkiego na temat wszystkich słów.

Modele RNN sprawdzają się bardzo dobrze w rozwiązywaniu wielu problemów, ale nie wszystkich — i trudne może być przewidywanie, czy rozwiązywanie konkretnego problemu zakończy się powodzeniem. Jednym z czynników zwiększających szansę takiego powodzenia jest fakt, że proces treningowy zachęca sieć do alokowania pamięci w warstwie $\mathbf{z}$ przede wszystkim na potrzeby tych aspektów wejściowych, których faktyczna użyteczność została już wykazana.

Proces trenowania modelu RNN opisywaliśmy w sekcji 21.6.1. Wektory wejściowe $\mathbf{x}_t$ reprezentują słowa odczytywane z korpusu treningowego, a obserwowanymi wyjściami są te same słowa przesunięte o 1. Na przykład dla tekstu treningowego „witaj świecie" pierwszym wektorem wejściowym $\mathbf{x}_1$ będzie embedding słowa „witaj", a pierwszym wyjściem $\mathbf{y}_1$ będzie embedding słowa „świecie". Trenujemy nasz model w celu przewidywania następnego słowa i spodziewamy się, że będzie on ten cel realizował wykorzystując ukrytą warstwę do reprezentowania użytecznych informacji. Jak wyjaśnialiśmy w sekcji 21.6.1, obliczamy różnicę między wyjściem obserwowanym a wyjściem prognozowanym przez sieć i propagujemy tę różnicę wstecz, utrzymując te same wagi dla wszystkich kroków czasowych.

Gdy proces treningowy zostanie zakończony, możemy generować losowe teksty za pomocą wytrenowanego modelu. Przekazujemy temu modelowi początkowy wektor wejściowy $\mathbf{x}_1$ i obserwujemy wyprodukowane wyjście $\mathbf{y}_1$, które jest rozkładem `softmax` względem możliwych słów. Próbkujemy pojedyncze słowo z tego rozkładu, rejestrujemy to słowo jako wyjście dla kroku czasowego t i dostarczamy je na wejście sieci jako słowo wejściowe $\mathbf{x}_2$. Powtarzamy ten proces dowolną liczbę razy. Próbkując z rozkładu $\mathbf{y}_1$ możemy dokonać wyboru strategii próbkowania: możemy mianowicie konsekwentnie wybierać słowo najbardziej prawdopodobne, możemy faworyzować poszczególne słowa proporcjonalnie do ich prawdopodobieństwa, albo też „przepróbkowywać" słowa mniej prawdopodobne w celu większego urozmaicenia generowanego wyjścia. Wybrana strategia jest hiperparametrem modelu i należy ją konsekwentnie stosować we wszystkich krokach.

Oto próbka losowego tekstu wygenerowanego w powyższy sposób przez model wytrenowany na bazie dzieł Szekspira (Karpathy, 2015):

> *Marry, and will, my lord, to weep in such a one were prettiest;*
> *Yet now I was adopted heir*
> *Of the world's lamentable day,*
> *To watch the next way with his father with his face?*

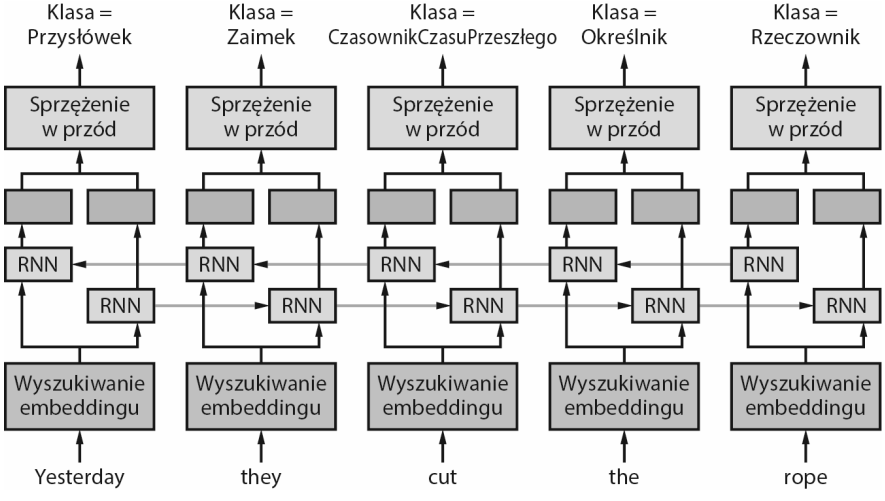
24.2.2. Klasyfikacja z użyciem sieci RNN

Rekurencyjne sieci neuronowe przydają się także w innych zadaniach NLP, między innymi tagowaniu części mowy i rozwiązywaniu koreferencji. W obu tych przypadkach warstwa wejściowa i warstwy ukryte pozostają niezmienione, lecz w przypadku tagera części mowy wyjściowy rozkład `softmax` dotyczy tagów POS, a w przypadku rozwiązywania koreferencji — możliwych poprzedników. Gdy na przykład sieć otrzyma na wejściu zdanie *Eduardo told me that Miguel was very sick so I took* **him** *to the hospital*, powinna przypisać wysokie prawdopodobieństwo słowu *Miguel*.

Trenowanie sieci RNN pod kątem takich zadań przebiega niemal tak samo, jak w przypadku modelu językowego RNN, z tą jednak różnicą, że dane treningowe *wymagają etykiet* — tagów części mowy lub wskaźników odwołań. Kolekcjonowanie takich danych jest znacznie trudniejsze w porównaniu z danymi nieetykietowanymi, które dla trenowania modelu językowego są wszystkim, co potrzebne.

Zadaniem modelu językowego jest przewidywanie *n*-tego słowa na podstawie poprzednich słów, natomiast w zadaniach klasyfikacyjnych nie ma żadnego powodu, byśmy ograniczali się wyłącznie do słów poprzedzających, bo równie owocne może być przeglądanie tekstu w przód. W naszym przykładzie obiekt koreferencji, do którego odnosi się zaimek *him*, byłby inny, gdyby zakończenie zdania brzmiało *to see Miguel*, a nie *to the hospital*, co doskonale ilustruje znaczenie przeglądania w przód. Liczne badania dowiodły zresztą, że człowiek analizujący wzrokowo tekst nie czyni tego stricte od lewej do prawej.

By uchwycić prawostronny kontekst słowa, możemy posłużyć się **dwukierunkową siecią RNN** (ang. *bidirectional RNN*) konkatenującą w sobie dwa oddzielne modele — „od lewej do prawej" i „od prawej do lewej". Efekt zastosowania takiej sieci do tagowania części mowy widoczny jest na rysunku 24.4.



RYSUNEK 24.4. Dwukierunkowa sieć RNN dla tagowania części mowy

W wielowarstwowej sieci RNN $\mathbf{z}_t$ będzie ukrytym wektorem ostatniej warstwy. W dwukierunkowej sieci RNN wektor $\mathbf{z}_t$ wybierany jest zazwyczaj jako kombinacja wektorów obu modeli: „od lewej do prawej" i „od prawej do lewej".

Sieci RNN mogą być także używane do klasyfikacji na poziomie całych zdań (a nawet na poziomie dokumentów), kiedy to wyjściem jest pojedyncza wartość, zamiast strumienia wartości wyjściowych, po jednej dla każdego kroku czasowego. Przykładem klasyfikacji tego rodzaju jest **analiza sentymentu**, polegająca na ocenie tekstu jako *Pozytywnego*

albo *Negatywnego* — na przykład recenzja *This movie was poorly written and poorly acted* („Ten film to kiepski scenariusz i słaba gra aktorów") ma wydźwięk zdecydowanie *Negatywny*. (Niektóre systemy analizy sentymentu wykorzystują więcej niż dwie kategorie albo dokonują oceny w postaci liczbowej).

Wykorzystywanie modelu RNN *na poziomie zdań* jest nieco bardziej skomplikowane, musimy bowiem uzyskać reprezentację $\mathbf{y}$ całego zdania jako *agregację* reprezentacji wyjściowych $\mathbf{y}_t$ poszczególnych słów tego zdania. Najprostszym sposobem uzyskania takiej agregacji jest wykorzystanie ukrytego stanu sieci odpowiadającego *ostatniemu słowu* zdania wejściowego — bieżący krok czasowy zawiera wówczas informację będącą funkcją całego zdania. Tak otrzymana agregacja jest jednak wyraźnie obciążona w tym sensie, że w większym stopniu odzwierciedla końcówkę zdania niż jego początek. Bardziej sprawiedliwą alternatywą jest pooling wszystkich ukrytych wektorów; najprostszą jego odmianą jest **pooling uśredniający** (ang. *averaging pooling*), którego wynikowa agregacja jest średnią wektorów składowych:

$$\tilde{\mathbf{z}} = \frac{1}{s}\sum_{t=1}^{s} \mathbf{z}_t$$

Wynikowy, d-wymiarowy wektor $\tilde{\mathbf{z}}$ może zostać następnie podany na jedną lub więcej warstw sprzężenia w przód, zanim pojawi się w warstwie wyjściowej.

24.2.3. Pamięci LSTM w zadaniach NLP

Jak wcześniej stwierdziliśmy, sieci RNN zdolne są w wielu przypadkach do niwelowania problemu niedostatecznego kontekstu. Teoretycznie można w nich przekazywać dowolną informację od jednej ukrytej warstwy do następnej, dla dowolnej liczby kroków czasowych. W praktyce jednak przekazywana informacja może zostać utracona lub zniekształcona, mniej więcej tak, jak w grze w „głuchy telefon": gracze ustawiają się w kolejkę, pierwszy szepcze drugiemu do ucha jakiś komunikat, drugi gracz szeptem przekazuje ów komunikat trzeciemu, itd.; zazwyczaj komunikat otrzymany przez ostatniego z graczy wyraźnie różni się od oryginału wymyślonego przez pierwszego gracza. W sieciach RNN problem ten jest analogią zjawiska **zanikającego gradientu**, opisywanego w sekcji 21.1.2 — z tą różnicą, że poszczególne warstwy odpowiadają teraz kolejnym krokom czasowym, a nie kolejnym stopniom zagłębienia.

W sekcji 21.6.2 przedstawiliśmy podstawy modelu **długotrwałej pamięci krótkoterminowej** (**LSTM**). Jest to odmiana sieci RNN, której bramki wolne są od problemu zniekształcania komunikatu przy kopiowaniu go między krokami czasowymi. Sieć RNN może wybierać do *zapamiętywania* niektóre części wejścia, kopiować je do następnego kroku czasowego i zapominać o innych częściach. Rozważmy przykładowe zdanie:

The athletes, who all won their local qualifiers and advanced to the finals in Tokyo, now ...

(„Lekkoatleci, którzy wygrali lokalne kwalifikacje i awansowali do finałów w Tokio, teraz ..."). Jeśli spytamy model, które ze słów[4]: *competes* czy *compete* powinno wystąpić w miejscu wielokropka, spodziewamy się otrzymać odpowiedź *compete* jako zgodną z podmiotem *The athletes*. LSTM może nauczyć się tworzyć ukrytą cechę dla określonej osoby i liczby, i kopiować w przód tę cechę bez zmian aż do momentu, gdy okaże się konieczne dokonanie opisanego wyboru. Warto zaznaczyć, że „regularne" sieci RNN (a także modele n-gramowe) często mylą się w przypadku długich zdań, w których podmiot oddzielony jest od orzeczenia długim ciągiem słów.

[4] *Competes* („rywalizuje") to forma specyficzna dla 3. osoby liczby pojedynczej, w zdaniu mamy do czynienia z 3. osobą liczby mnogiej (*they*), więc musi być *compete* — *przyp. tłum.*

24.3. Modele „sekwencja na sekwencję”

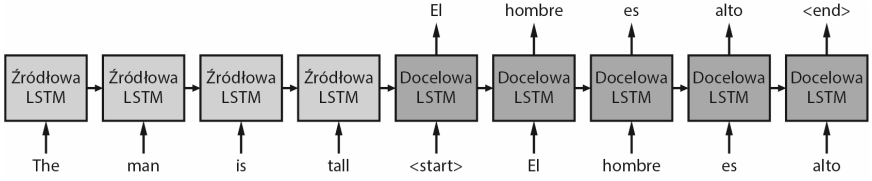
Jednym z najintensywniej badanych obszarów NLP jest **tłumaczenie maszynowe** (ang. MT — *Machine Translation*), którego celem jest tłumaczenie zdania zapisanego w języku źródłowym (na przykład hiszpańskim) na równoważne zdanie wyrażone w **języku docelowym** (na przykład angielskim). Trenowanie modeli MT odbywa się zwykle w oparciu o duże korpusy zawierające pary równoważnych zdań wyrażonych w obu językach, a jego celem jest nauczenie modelu dokładnego tłumaczenia nowych zdań, niewystępujących w zbiorach treningowych.

Czy możliwe jest budowanie systemów MT w oparciu o sieci RNN? Oczywiście możemy zakodować zdanie źródłowe w formie sieci RNN i gdyby tłumaczenie zdań polegało wyłącznie na odwzorowywaniu „jeden-do-jednego” słów języka źródłowego na słowa języka docelowego, zadanie MT sprowadzałoby się do prostego tagowania słów — hiszpańskie słowo *perro* („pies”) zostałoby otagowane przez angielski odpowiednik *dog*. W rzeczywistości tłumaczenie zdań (i w ogóle sekwencji słów) nie daje się zdekomponować na tłumaczenie oddzielnych słów; przykładowo hiszpańskiemu określeniu *caballo de mar* („konik morski”) odpowiada pojedyncze słowo angielskie *seahorse*, a w tłumaczeniu *perro grande* → *big dog* („duży pies”) odpowiadające sobie słowa występują w odwrotnej kolejności. Zmiana kolejności słów w wyniku tłumaczenia może być bardziej spektakularna; w języku angielskim podmiot zwykle rozpoczyna zdanie, ale na przykład w języku, którym posługują się mieszkańcy Republiki Fidżi, występuje on przeważnie na końcu zdania. Skoro więc nie odwzorowanie na poziomie słów, to w jaki sposób mamy generować zdania w języku docelowym?

Wygląda na to, że powinniśmy generować jedno słowo na raz, jednocześnie śledząc kontekst, zapamiętując te części źródła, które nie zostały jeszcze przetłumaczone i śledząc te, które już przetłumaczone zostały (byśmy nie musieli się powtarzać). W przypadku niektórych zdań może być i tak, że będziemy musieli przeczytać całe zdanie źródłowe przed wygenerowaniem pierwszego słowa zdania docelowego.

Uwidacznia to związek generowania tekstu w ramach MT ze standardowym modelem językowym RNN, opisywanym w podrozdziale 24.2. Istotnie, model RNN wytrenowany pod kątem języka angielskiego, wygeneruje raczej sekwencję *big dog* niż *dog big*. Nie zapominajmy jednak, że mamy do czynienia z *tłumaczeniem* — nie chodzi nam więc o generowanie losowych zdań w języku docelowym, lecz zdań stanowiących odpowiedniki zdań tekstu źródłowego. Najprostszym sposobem realizacji tego zadania jest użycie dwóch sieci RNN, z których jedna (nazwijmy ją „źródłową”) wytrenowana jest w języku źródłowym, a druga (docelowa) w języku docelowym. Podajemy następnie zdanie źródłowe na wejście źródłowej RNN, po czym wykorzystujemy jej finalny ukryty stan jako początkowy ukryty stan docelowej RNN. Tak oto każde słowo docelowe zostaje samoczynnie uzależnione zarówno od całego zdania źródłowego, jak i od wygenerowanych poprzednio słów docelowych.

Taką architekturę sieci neuronowej nazywamy podstawowym modelem „**sekwencja na sekwencję**” (ang. *sequence-to-sequence model*) — przykład takowego przedstawiamy na rysunku 24.5. Modele takie wykorzystywane są najczęściej w tłumaczeniu maszynowym, ale również w kilku innych zadaniach, takich jak automatyczne generowanie podpisów (tytułów) obrazów, streszczenia oraz skracanie tekstu, bez zmiany jego znaczenia.



RYSUNEK 24.5. Podstawowy model „sekwencja na sekwencję”. Każdy blok reprezentuje jeden krok czasowy LSTM (dla prostoty pominęliśmy embeddingi i warstwy wyjściowe). W kolejnych krokach przekazujemy do sieci kolejne słowa zdania źródłowego *The man is tall* („Ten mężczyzna jest wysoki”), zakończonego ogranicznikiem `<start>`, oznaczającym, że sieć może rozpocząć generowanie zdania docelowego. Końcowy ukryty stan na końcu zdania źródłowego wykorzystywany jest jako ukryty stan na starcie zdania docelowego. Następnie każde słowo zdania docelowego w kroku t wykorzystywane jest jako wejście w kroku $t + 1$ aż do momentu, gdy sieć wygeneruje znacznik <end> oznaczający, że generowanie zdania docelowego zakończyło się

Podstawowe modele „sekwencja na sekwencję” okazały się przełomem w przetwarzaniu języka naturalnego, a w szczególności w tłumaczeniu maszynowym. Według artykułu Wu i in. (2016b) przyczyniły się one do 60% redukcji błędów w stosunku do poprzednich metod MT. Nie są one jednak wolne od mankamentów, z których trzy najważniejsze to:

- **Polaryzacja w kierunku najbliższego kontekstu**. Wszystko, co sieć RNN zdecyduje się zapamiętywać odnośnie przeszłości, musi być zgodne z jej ukrytym stanem. Jeśli na przykład sieć przetwarza aktualnie 57. słowo (czyli krok czasowy $t = 57$) sekwencji liczącej 70 słów, to ukryty stan sieci prawdopodobnie zawierał więcej informacji na temat kroku $t = 56$ niż na temat kroku $t = 5$. Jest tak dlatego, że każdorazowo, gdy ukryty wektor jest aktualizowany, pewna część istniejącej informacji musi zostać zastąpiona przez nową informację. Takie zachowanie jest zamierzoną konsekwencją projektu modelu i jest szczególnie użyteczne w zadaniach NLP, gdzie najbliższy kontekst jest przeważnie ważniejszy od dalszego. W sytuacji jednak, gdy dalszy kontekst jest równie ważny, czy nawet kluczowy dla zadania, może zostać zmarginalizowany lub wręcz zagubiony przez model RNN i nawet LSTM może napotkać trudności nie do przezwyciężenia.
- **Ustalony limit rozmiaru kontekstu**. W modelu tłumaczeniowym RNN całe zdanie źródłowe skompresowane zostaje do postaci pojedynczego ukrytego wektora stanu o ustalonej wymiarowości. W pamięciach LSTM wykorzystywanych w systemach NLP z górnej półki wymiarowość ta wynosi typowo 1024, jeśli więc chcemy reprezentować w jej ramach zdanie zawierające (powiedzmy) 64 słowa, to wypada nam 1024/64 = 16 wymiarów na słowo — stanowczo zbyt mało dla skomplikowanych zdań. Powiększanie rozmiaru wspomnianego wektora skutkuje spowalnianiem treningu i stwarza ryzyko przeuczenia modelu.
- **Sekwencyjność spowalniająca przetwarzanie**. Jak wyjaśnialiśmy w podrozdziale 21.3, sieci neuronowe przyczyniają się do znacznego wzrostu wydajności, ponieważ przetwarzają dane treningowe partiami, by efektywnie wykorzystywać sprzętowe możliwości zrównoleglania obliczeń oferowane przez procesory macierzowe. Tak się jednak składa, że sieci RNN ograniczone są raczej do sekwencyjnego przetwarzania słowa po słowie i możliwości zrównoleglania są w ich przypadku co najwyżej szczątkowe.

24.3.1. Mechanizmy uwagi w modelach NLP

Docelowa RNN może być uzależniona od *wszystkich* ukrytych wektorów źródłowej RNN, nie tylko od ostatniego. Nie występują wtedy problemy polaryzacji kontekstu ani ograniczonego rozmiaru kontekstu, model ma wówczas równoprawny dostęp do każdego poprzedniego słowa. Jednym ze sposobów zapewnienia takiego dostępu jest konkatenacja wszystkich ukrytych wektorów źródłowej RNN, jednak takie rozwiązanie skutkuje gwałtownym wzrostem liczby wag, z konsekwencjami w postaci wydłużenia czasu obliczeń i zaistnienia ryzyka przeuczenia modelu. Rozwiązanie bardziej efektywne zasadza się na spostrzeżeniu, że docelowa RNN generuje po jednym słowie na raz, a więc w danej chwili (kroku czasowym) prawdopodobnie tylko niewielka część wejścia będzie rzeczywiście istotna (relewantna) dla danego słowa docelowego.

Co najważniejsze, owe „relewantne części” informacji źródłowej mogą być różne dla poszczególnych słów docelowych. Załóżmy, że wytrenowaliśmy sieć pod kątem tłumaczenia z języka angielskiego na hiszpański, a na jej wejściu pojawiła się sekwencja *The front door is red* („Drzwi frontowe są czerwone”), po której następuje znacznik końca zdania, uprawniający do rozpoczęcia generowania hiszpańskich słów. Sieć powinna więc zwrócić uwagę na słowo *The* i wygenerować słowo *La*, następnie zwrócić uwagę na słowo *door* i wygenerować *puerta*, i tak dalej.

Owa koncepcja „zwracania uwagi” odzwierciedlona została w projekcie sieci w postaci jej komponentu zwanego (po prostu) **uwagą** (ang. *attention*), za pomocą którego można kreować „kontekstowe podsumowanie” zdania źródłowego w postaci wektora o ustalonej wymiarowości. Ów kontekstowy wektor $\mathbf{c}_i$ używany jest następnie jako dodatkowe wejście do docelowej RNN. Model „sekwencja na sekwencję” wykorzystujący taki komponent nazywany jest **uwagowym modelem „sekwencja na sekwencję”** (ang. *attentional sequence-to-sequence model*). Jeśli standardową docelową sieć RNN zapiszemy w postaci

$$\mathbf{h}_i = RNN(\mathbf{h}_{i-1}, \mathbf{x}_i)$$

to docelową sieć RNN dla wspomnianego modelu zapisać można w tej konwencji jako

$$\mathbf{h}_i = RNN(\mathbf{h}_{i-1}, [\mathbf{x}_i ; \mathbf{c}_i])$$

gdzie $[\mathbf{x}_i ; \mathbf{c}_i]$ oznacza konkatenację wektorów wejściowego ($\mathbf{x}_i$) i kontekstowego ($\mathbf{c}_i$), przy czym ten ostatni definiowany jest następująco:

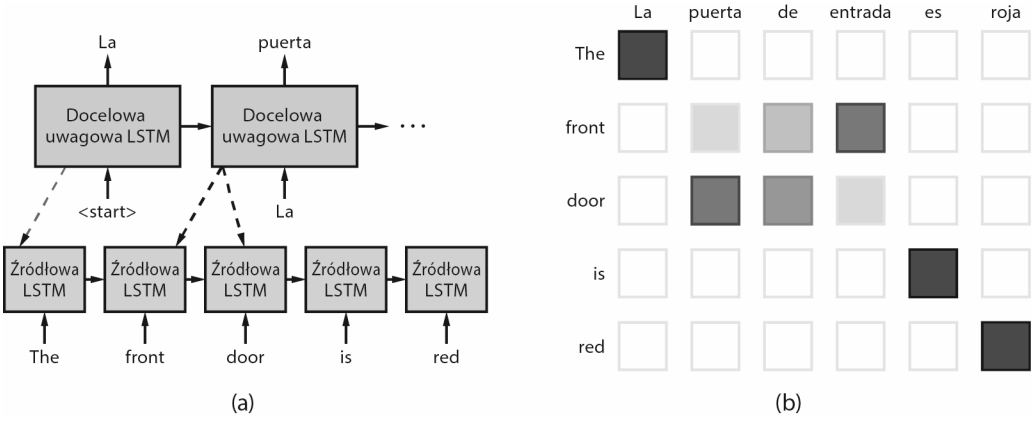
$$r_{ij} = \mathbf{h}_{i-1} \cdot \mathbf{s}_j$$

$$a_{ij} = \frac{e^{r_{ij}}}{\sum_k e^{r_{ik}}}$$

$$\mathbf{c}_j = \sum_j a_{ij} \cdot \mathbf{s}_j$$

gdzie $\mathbf{h}_{i-1}$ jest wektorem docelowej RNN, który będzie użyty do przewidywania słowa docelowego w kroku czasowym i, a $\mathbf{s}_j$ jest wektorem wyjściowym źródłowej RNN dla słowa źródłowego j (czyli kroku czasowego j). Zarówno $\mathbf{h}_{i-1}$ jak i $\mathbf{s}_j$ są wektorami d-wymiarowymi, gdzie d jest ukrytym rozmiarem. Wartość r_{ij} jest więc surową „punktacją uwagową" (ang. *attention score*) między bieżącym stanem docelowym a j-tym słowem źródłowym. Punktacje te są następnie normalizowane do prawdopodobieństwa a_{ij} według rozkładu `softmax` dla wszystkich słów źródłowych. Ostatecznie prawdopodobieństwa te wykorzystywane są do generowania średniej ważonej wektorów źródłowej sieci RNN, czyli $\mathbf{c}_i$ (która to średnia również jest wektorem d-wymiarowym).

Przykład zastosowania uwagowego modelu „sekwencja na sekwencję" widoczny jest w części (a) rysunku 24.6, zawierającego kilka istotnych szczegółów. Po pierwsze, komponent uwagi nie posiada sam z siebie wyuczonych wag i obsługuje sekwencje o zmiennej długości, zarówno po stronie źródłowej, jak i docelowej. Po drugie, analogicznie do wielu innych modeli opartych na sieciach neuronowych, komponent ten jest całkowicie ukryty. Programista nie może dyktować modelowi, jakich ten ma używać informacji — model uczy się tego samodzielnie. Komponent uwagi może być łączony z wielowarstwowymi sieciami RNN, zwykle komponenty takie znajdują się wówczas w każdej warstwie.



RYSUNEK 24.6. (a) Uwagowy model „sekwencja na sekwencję" dla tłumaczenia z języka angielskiego na hiszpański (*The front door is red* — „Drzwi frontowe są czerwone"). (b) Przykład uwagowej macierzy prawdopodobieństw dla dwujęzycznej pary zdań, ciemniejsze kwadraty reprezentują większe wartości a_{ij}. Wartości prawdopodobieństw uwagowych sumują się do 1 w każdej kolumnie

Sformułowanie funkcjonalności komponentu uwagi w kategoriach rozkładu `softmax` służy trojakiemu celowi. Po pierwsze, komponent ten staje się różniczkowalny, co jest konieczne do użycia go w propagacji wstecz; propagacja ta wciąż ma miejsce w źródłowej i docelowej RNN, mimo iż jej komponenty uwagi nie posiadają własnych wyuczonych wag. Po drugie, formuła probabilistyczna umożliwia modelowi uchwycenie pewnych typów długodystansowych kontekstualizacji, które inaczej pozostawałyby niezauważalne dla źródłowej RNN, ponieważ komponent uwagi może rozważać całą sekwencję źródłową naraz i nauczyć się wybierać z niej to, co istotne. Po trzecie wreszcie, probabilistyczny komponent uwagi pozwala sieci na reprezentowanie niepewności — jeśli sieć nie wie dokładnie, jakie słowo źródłowe ma przetłumaczyć w następnej kolejności, może uwzględnić kilka opcji, opatrując je prawdopodobieństwami, i następnie wybrać właściwe słowo, wykorzystując docelową RNN.

W przeciwieństwie do większości komponentów sieci neuronowych, prawdopodobieństwa komponentu uwagi są intuicyjnie zrozumiałe dla ludzi. Przykładowo, w przypadku tłumaczenia maszynowego prawdopodobieństwa te mogą być odnoszone do kojarzenia na poziomie pojedynczych słów, stosowanego często przy ręcznym tłumaczeniu; przykład takiego kojarzenia przedstawiamy w części (b) rysunku 24.6.

Modele „sekwencja na sekwencję” w naturalny sposób przystosowane są do tłumaczenia maszynowego, ale niemal każde zadanie NLP można zakodować w postaci problemu rozwiązywalnego przez taki model. Przykładowo, system odpowiadania na pytania można trenować na zbiorze par, których pierwszym elementem jest pytanie, a drugim prawidłowa odpowiedź ograniczona specjalnym znacznikiem końca.

24.3.2. Dekodowanie

W czasie treningu model „sekwencja na sekwencję” dąży do maksymalizacji prawdopodobieństwa wystąpienia każdego słowa w docelowej sekwencji treningowej, w zależności od źródła i wszystkich poprzednich słów docelowych. Po zakończeniu treningu dostarczamy modelowi zdanie źródłowe i oczekujemy wygenerowania odpowiadającego mu zdania docelowego. Jak wynika to z rysunku 24.6, możemy generować po jednym słowie docelowym na raz, po czym propagujemy wstecz słowo, które wygenerowaliśmy w następnym kroku czasowym. Procedura ta nosi nazwę **dekodowania** (ang. *decoding*).

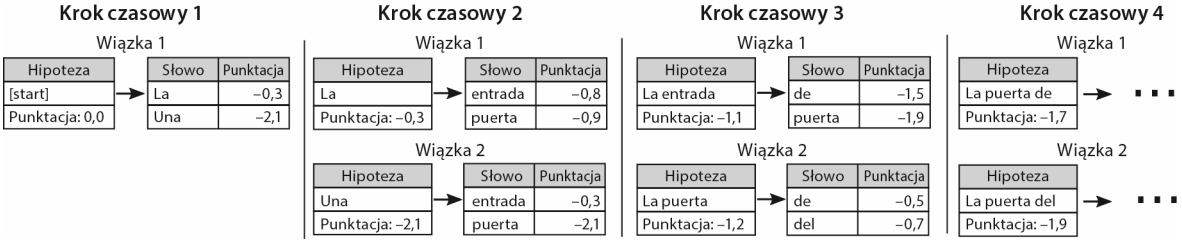
Najprostszą formą dekodowania jest wybieranie, w każdym kroku czasowym, słowa o największym prawdopodobieństwie i przekazywanie tego słowa jako wejścia do następnego kroku czasowego. Taką strategię nazywamy **zachłannym dekodowaniem** (ang. *greedy decoding*), ponieważ po wygenerowaniu każdego ze słów docelowych system bezgranicznie polega na hipotezie, którą sformułował do tej pory. Problem jednak w tym, że celem dekodowania jest maksymalizacja prawdopodobieństwa *całej sekwencji docelowej*, a tego zachłanna strategia nie jest w stanie zagwarantować. Rozpatrzmy na przykład użycie zachłannego dekodera do przetłumaczenia cytowanego wcześniej zdania *The front door is red.*

Poprawnym tłumaczeniem jest *La puerta de entrada es roja* („Drzwi wejściowe są czerwone”). Załóżmy, że docelowa RNN poprawnie wygeneruje pierwsze słowo *La* jako odpowiednik *The*, po czym zachłanny dekoder w charakterze drugiego słowa zaproponuje *entrada* jako odpowiednik *front*. I tu pojawi się błąd — zgodnie z hiszpańskim szykiem wyrazów, podmiot *puerta* powinien wystąpić przed przydawką *entrada*. Zachłanne dekodowanie jest co prawda szybkie — w każdym kroku czasowym dokonywany jest tylko jeden wybór — ale niestety, model nie dysponuje mechanizmem umożliwiającym korygowanie popełnionych błędów.

Moglibyśmy spróbować ulepszyć mechanizm uwagi tak, by zawsze zwracał uwagę na właściwe słowo i za każdym razem poprawnie je odgadywał. To jest jednak niemożliwe w przypadku niektórych zdań, bo nie sposób prawidłowo odgadnąć początkowe wyrazy zdania bez wiedzy o tym, co znajduje się na jego końcu.

Lepszym podejściem jest poszukiwanie optymalnego (lub przynajmniej — dobrego) dekodowania przy użyciu jednego z algorytmów wyszukiwania opisywanych w rozdziałach 3. i 4. Często wykorzystuje się w tej roli **lokalne wyszukiwanie skupione** (patrz sekcja 4.1.3). W kontekście dekodowania MT, przeszukiwanie to utrzymuje zazwyczaj k najlepszych hipotez na każdym etapie, rozszerzając każdą o jedno słowo, na k sposobów przy użyciu k najlepszych wyborów słów, a następnie wybiera najlepsze k z wynikowych k^2 nowych hipotez. Gdy każda z hipotez w wiązce wygeneruje specjalny token `<end>`, algorytm wyprowadza hipotezę o najwyższej punktacji.

Rysunek 24.7 przedstawia wizualizację przykładowego wyszukiwania skupionego. W miarę jak model głębokiego uczenia staje się bardziej dokładny, możemy przeważnie ograniczyć rozmiar wiązki; w nowoczesnych modelach neuronowych MT rozmiar ten waha się między 4 a 8, podczas gdy starsze modele statystycznego MT posługiwały się wiązkami o rozmiarach 100 i więcej.



RYSUNEK 24.7. Wyszukiwanie skupione z rozmiarem wiązki $b = 2$. Punktacja każdego słowa równa jest logarytmicznemu prawdopodobieństwu wynikającemu z rozkładu `softmax` generowanego przez sieć RNN, a punktacja każdej hipotezy jest sumą punktacji poszczególnych słów. W kroku czasowym $t = 3$ hipoteza o najwyższej punktacji *La entrada* może generować jedynie kontynuacje o niskim prawdopodobieństwie, więc plasuje się poza wiązką.

24.4. Architektura transformerów

Vaswani i in. w swym wpływowym artykule z 2018 roku wprowadzają architekturę **transformera** (ang. *tramsformer*), wykorzystującą mechanizm **samouwagi** (ang. *self-attention*) zdolny do modelowania długodystansowego kontekstu, bez zależności sekwencyjnych.

24.4.1. Samouwaga

W modelach „sekwencja na sekwencję" uwaga zwrócona była ze strony docelowej RNN na źródłową RNN. Mechanizm **samouwagi** rozszerza tę funkcjonalność w ten sposób, że każda sekwencja ukrytych stanów (źródłowa i docelowa) zwraca uwagę również na samą siebie. Pozwala to modelowi dodatkowo uchwycić kontekst zarówno długodystansowy, jak i pobliski w każdej sekwencji.

Najprostszym wariantem zastosowania samouwagi jest sytuacja, w której macierz uwagi tworzona jest bezpośrednio przez iloczyn skalarny wektorów wejściowych. Przypadek taki jest jednak o tyle problematyczny, że iloczyn skalarny wektora przez samego siebie zawsze jest duży, więc każdy ukryty stan będzie preferował zwracanie uwagi na samego siebie. Transformer rozwiązuje ten problem, rozpoczynając od rzutowania danych wejściowych na trzy różne reprezentacje, przy użyciu trzech różnych macierzy wagowych[5]:

- **wektor zapytania** $\mathbf{q}_i = \mathbf{W}_q\mathbf{x}_i$ jest tym *obserwującym*, podobnie jak strona docelowa w standardowym mechanizmie uwagi;
- **wektor klucza** $\mathbf{k}_i = \mathbf{W}_k\mathbf{x}_i$ jest tym *obserwowanym*, podobnie jak strona źródłowa w standardowym mechanizmie uwagi;
- **wektor wartości** $\mathbf{v}_i = \mathbf{W}_v\mathbf{x}_i$ jest kontekstem generowania.

[5] Patrz także *https://ichi.pro/pl/google-deepmind-s-rfa-approximating-softmax-attention-mechanism-in-transformers-235867597421592* — *przyp. tłum.*

W standardowym mechanizmie uwagi sieci odpowiadające kluczowi i wartości są identyczne, ale intuicyjnie ma sens ich oddzielnie reprezentowanie. Rezultat kodowania *i*-tego słowa — $\mathbf{c}_i$ — można obliczyć, stosując mechanizm uwagi do rzutowanych wektorów.

$$r_{ij} = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d}}$$

$$a_{ij} = \frac{e^{r_{ij}}}{\sum_m e^{r_{im}}}$$

$$\mathbf{c}_j = \sum_j a_{ij} \cdot \mathbf{v}_j$$

gdzie d jest wymiarowością wektorów $\mathbf{k}$ i $\mathbf{q}$. Zauważmy, że i oraz j są indeksami słów w tym samym zdaniu, ponieważ kodujemy kontekst w konwencji samouwagi. W każdej warstwie transformera samouwaga wykorzystuje ukryte wektory poprzedniej warstwy — którą początkowo jest warstwa kontekstowa.

Kilka szczegółów w powyższych formułach wartych jest szczególnej uwagi. Przede wszystkim mechanizm samouwagi jest *asymetryczny*, ponieważ $r_{ij} \neq r_{ji}$. Po drugie, rolą czynnika skalującego $\sqrt{d}$ jest poprawienie stabilności numerycznej algorytmu. Po trzecie, kodowanie poszczególnych słów w zdaniu może być prowadzone *równolegle* — fundamentem algorytmu są operacje macierzowe, dla których współczesne procesory oferują wydajne implementacje sprzętowe.

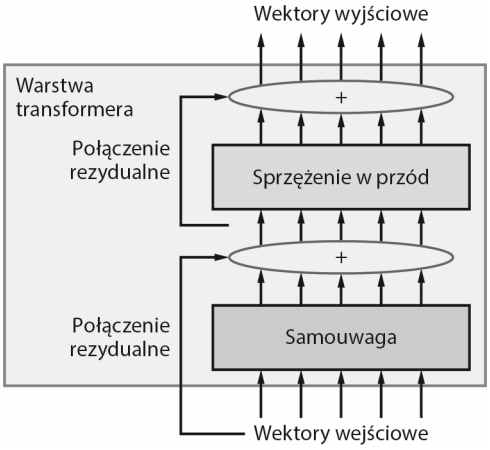
Wybór kontekstu do wykorzystania nie jest z góry określony, lecz całkowicie wyuczony na przykładach treningowych. Podsumowanie kontekstowe — $\mathbf{c}_i$ — to suma wszystkich poprzednich pozycji w zdaniu. Teoretycznie wszelkie informacje zawarte w zdaniu mogą pojawić się w $\mathbf{c}_i$. Jednym ze sposobów zapobiegania temu zjawisku jest **wielouwaga** (ang. *multiheaded attention*): dzielimy zdanie na m równych części i do każdej z nich stosujemy model uwagi; każda część ma swój własny zbiór wag. Konkatenujemy następnie wyniki, otrzymując $\mathbf{c}_i$; konkatenacja, w przeciwieństwie do sumowania, nie stwarza ryzyka utraty informacji cząstkowych.

24.4.2. Od samouwagi do transformera

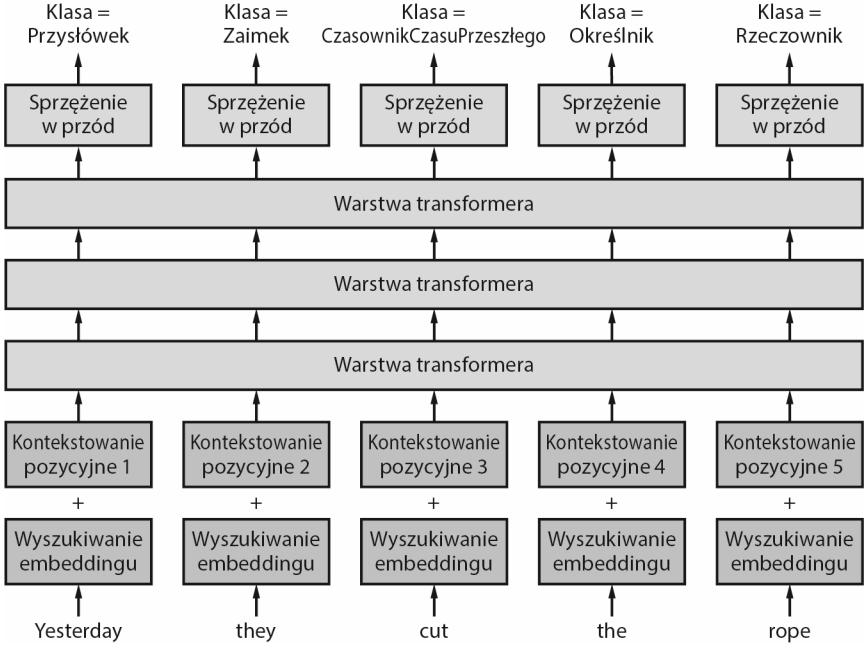
Samouwaga to tylko jeden z elementów modelu transformera. Każda warstwa transformera składa się z kilku podwarstw. Do każdej warstwy transformera najpierw stosowana jest samouwaga, następnie wyjście modułu uwagi kierowane jest na warstwy sprzężenia w przód, gdzie te same macierze wagowe sprzężenia stosowane są niezależnie do każdej pozycji. Nieliniowa funkcja aktywacyjna — zazwyczaj `ReLU` — stosowana jest po pierwszej warstwie sprzężenia. Aby rozwiązać potencjalny problem zanikającego gradientu, do warstwy transformera dodawane są dwa połączenia rezydualne. Jednowarstwowy transformer widoczny jest na rysunku 24.8. W praktyce modele transformerów zawierają zwykle sześć lub więcej warstw. Podobnie jak w przypadku innych wcześniej opisywanych modeli, wyjście warstwy i jest podawane na wejście warstwy i+1.

Architektura transformera nie utrzymuje jawnej informacji o kolejności słów w sekwencji, ponieważ kontekst modelowany jest wyłącznie poprzez samouwagę, która kolejności tej nie zna. By jednak informację o tej kolejności utrzymywać, transformer wykorzystuje technikę **embedowania pozycyjnego** (ang. *positional embedding*) Jeśli wejściowa sekwencja ma maksymalną długość n, model uczy się n nowych embeddingów, po jednym dla każdej pozycji słowa. Wejściem do pierwszej warstwy transformera jest suma embeddingów słowa na pozycji t oraz embeddingu pozycyjnego odpowiadającego pozycji t.

Rysunek 24.9 przedstawia architekturę transformera dla tagowania części mowy (POS), w zastosowaniu do tego samego zdania, które widoczne jest u dołu rysunku 24.2. W dolnej części rysunku 24.9 embeddingi słów sumowane są z embeddingami pozycyjnymi, co daje w rezultacie wejście do trójwarstwowego transformera. Transformer produkuje jeden wektor dla każdego słowa, tak jak w tagowaniu POS opartym na RNN. Każdy taki wektor podawany jest na końcową warstw wyjściową i na warstwę `softmax` w celu obliczenia rozkładu prawdopodobieństwa poszczególnych tagów.



RYSUNEK 24.8. Jednowarstwowy transformer składający się z samouwagi, sieci ze sprzężeniem w przód i połączeń rezydualnych



RYSUNEK 24.9. Wykorzystywanie architektury transformera do tagowania części mowy.

To, co dotychczas napisaliśmy o transformerach, to tylko część ich historii — ta sekcja ogranicza się do modelu zwanego **koderem transformera** (ang. *transformer encoder*), użytecznego w zadaniach klasyfikacji tekstu. Pełna architektura transformera została oryginalnie zaprojektowana jako model „sekwencja na sekwencję" na potrzeby tłumaczenia maszynowego. Oprócz kodera transformera istnieje więc także **dekoder transformera** (ang. *transformer decoder*). Koder i dekoder są niemal identyczne, z tą różnicą, że dekoder wykorzystuje wersję samouwagi, w której każde słowo może zwracać uwagę wyłącznie na słowa poprzedzające, ponieważ tekst jest generowany od strony lewej do prawej. Dekoder posiada też drugi moduł uwagi w każdej warstwie transformera, który to moduł zwraca uwagę na koder transformera.

24.5. Trenowanie wstępne i uczenie transferowe

Pozyskanie danych w ilości wystarczającej do zbudowania solidnego modelu może być prawdziwym wyzwaniem. W przypadku widzenia komputerowego (patrz rozdział 25.) wyzwanie to rozwiązano poprzez zgromadzenie dużych kolekcji obrazów (takich jak ImageNet) i ręczne ich zaetykietowanie.

Przetwarzanie języka naturalnego to przeważnie praca z nieetykietowanym tekstem — i znacznie trudniejsze jego etykietowanie: niewykształcony obserwator z łatwością może stwierdzić, że obraz przedstawia „kota" albo „zachód słońca", ale adnotowanie tekstu za pomocą tagów części mowy lub drzew rozbioru syntaktycznego wymaga solidnego treningu. Specyfika przetwarzania tekstu wynika także z jego obfitości: każdego dnia w internecie pojawia się ponad 100 miliardów słów tekstu, w tym zdigitalizowane książki, wyselekcjonowane zasoby, takie jak Wikipedia oraz nieredagowane posty w mediach społecznościowych.

Łatwy dostęp do takich danych zapewniają projekty takie jak Common Crawl. Do budowania modeli *n*-gramowych lub embeddingowych można użyć dowolnych strumieni tekstu, a struktura niektórych może być pomocna w rozmaitych zadaniach — istnieje na przykład wiele witryn z często zadawanymi pytaniami (ang. FAQ — *Frequently Asked Questions*), oferujących gotowe pary „pytanie-odpowiedź"; można je wykorzystywać do treningu systemów odpowiadających na pytania. Podobnie na wielu stronach sieci Web znaleźć można publikowane obok siebie warianty tekstu w różnych wersjach językowych, co oczywiście może stanowić cenny materiał treningowy dla systemów tłumaczenia maszynowego. Można też spotkać wiele tekstów etykietowanych w specyficzny sposób: na przykład recenzje, które autorzy opatrują ratingiem w skali (powiedzmy) od 0 do 5.

Wolelibyśmy oszczędzić sobie trudu tworzenia nowego zbioru danych dla każdego nowego modelu NLP. W tej sekcji przedstawiamy ideę **treningu wstępnego** (ang. *pretraining*) — formę **uczenia transferowego** (patrz sekcja 21.7.2), w której używamy dużej ilości współdzielonych danych ogólnej domeny językowej do trenowania początkowej wersji modelu NLP. Po takim wstępnym treningu możemy użyć mniejszej ilości danych specyficznych dla konkretnej domeny (być może danych częściowo etykietowanych) dla doskonalenia modelu — który następnie może uczyć się słownictwa, idiomów, struktur składniowych i innych osobliwości lingwistycznych specyficznych dla nowej domeny.

24.5.1. Wstępny trening embeddingów

W podrozdziale 24.1 przedstawiliśmy pokrótce koncepcję kontekstowania słów, w szczególności pokazując, jak podobieństwo słów — na przykład „jabłko" i „banan" — przekłada się na podobieństwo ich embeddingów i jak dzięki temu można rozwiązywać problemy z analogią słów, posługując się jedynie dodawaniem i odejmowaniem wektorów. Oznacza to, że embeddingi słów wychwytują istotne informacje na ich temat.

W tej sekcji zagłębimy się w szczegóły tworzenia embeddingów słów przy użyciu całkowicie nienadzorowanego procesu zastosowanego do dużego korpusu tekstu. Różni się to od kontekstowania opisywanego w podrozdziale 24.1, stanowiącego nadzorowany element tagowania części mowy i tym samym wymagającego tagów POS, które są wynikiem pracochłonnego ręcznego adnotowania.

Skoncentrujemy się na jednym konkretnym modelu embeddingowym — GloVe (ang. *Global Vectors*). Model rozpoczyna swe działanie od zliczenia, ile razy każde ze słów pojawia się w oknie innego słowa, podobnie jak w modelach skipgramowych (patrz sekcja 23.1.3). Najpierw wybieramy rozmiar okna (powiedzmy 5 słów). Niech X_{ij} oznacza liczbę wystąpień słów i oraz j w tym samym oknie, a X_i niech oznacza liczbę wystąpień słowa i we wspólnym oknie z jakimkolwiek innym słowem; wówczas $P_{ij} = X_{ij}/X_i$ jest prawdopodobieństwem wystąpienia słowa j w kontekście słowa i. Tak jak poprzednio, oznaczmy przez $\mathbf{E}_i$ embedding słowa i.

Intuicyjny charakter modelu GloVe bierze się w dużej mierze stąd, że związek między dwoma słowami można najlepiej uchwycić przez porównywanie ich z innymi słowami. Rozważmy na przykład słowa *ice* („lód") i *steam* („para") i stosunek prawdopodobieństw ich współwystępowania z pewnym słowem *w*, czyli

$$P_{w,\,ice}/P_{w,\,steam}$$

Ponieważ lód jest ciałem stałym (ang. *solid*), a para wodna jest gazem (ang. *gas*), więc $P_{solid,\,ice}$ będzie wyraźnie większe od $P_{solid,\,steam}$, a $P_{gas,\,ice}$ będzie znacznie mniejsze od $P_{gas,\,steam}$ — w obu przypadkach stosunek wymienionych prawdopodobieństw będzie wyraźnie odbiegał od 1. Gdy natomiast *w* będzie słowem pozbawionym treści, na przykład *the*, stosunek $P_{w,\,ice}/P_{w,\,steam}$ będzie bliski 1; tak samo będzie w przypadku, gdy *w* będzie słowem jednakowo relewantnym dla *ice* i *steam* (na przykład *water* — „woda") albo jednakowo dla nich nierelewantnym (na przykład *fashion* — „moda").

Model GloVe, opierając się na tym intuicyjnym spostrzeżeniu, wykonuje pewne operacje matematyczne (Pennington i in., 2014), które przekształcają stosunki prawdopodobieństw na różnice wektorowe i iloczyny skalarne, co ostatecznie prowadzi do zależności

$$\mathbf{E}_i \cdot \mathbf{E}'_j = \log\left(P_{ij}\right)$$

Innymi słowy, iloczyn skalarny dwóch embeddingów jest równy logarytmicznemu prawdopodobieństwu ich współwystępowania. Ma to intuicyjny sens: dwa prawie prostopadłe wektory mają iloczyn skalarny bliski zeru, a dwa prawie identyczne znormalizowane[6] wektory mają iloczyn skalarny bliski 1. Istnieje pewna techniczna komplikacja polegająca na tym, że model GloVe tworzy dwa embeddingi $\mathbf{E}_i$ i $\mathbf{E}'_i$ dla każdego słowa i na końcu dodaje je do siebie; intencją tego zabiegu jest zmniejszenie ryzyka przeuczenia modelu.

Uczenie modelu takiego jak GloVe jest zazwyczaj znacznie tańsze niż uczenie standardowej sieci neuronowej: nowy model można wytrenować na korpusie zawierającym miliardy słów w ciągu kilku godzin przy użyciu standardowego procesora domowego komputera.

Zdarza się, że trenowanie modelu w kontekstowaniu słów z określonej domeny doprowadza do odkrycia wiedzy głęboko tkwiącej w tej domenie, ale jeszcze powszechnie nieuświadomionej. Tshitoyan i in. (2019) wykorzystali 3,3 miliona abstraktów publikacji naukowych z dziedziny materiałoznawstwa jako korpus treningowy. Widzieliśmy wcześniej, jak model embeddingów potrafi przewidywać właściwe słowa na zasadzie analogii („Ateny mają się go Grecji tak, jak Oslo do…"); na tej samej zasadzie wspomniany zespół postanowił uzyskać odpowiedź na pytanie „NiFe tak się ma do ferromagnetyków, jak IrMn do…" — i GloVe udzielił odpowiedzi „antyferromagnetyków".

Wydaje się, że ich model nie polegał wyłącznie na statystyce współwystępowania słów, lecz dysponował dodatkowo pewną wiedzą domenową. Zapytany o to, jakie związki chemiczne są „termoelektrykami" i „izolatorami topologicznymi", był w stanie poprawnie odpowiedzieć, mimo iż wzór chemiczny $CsAgGa_2Se_4$ nigdzie w korpusie nie pojawia się w pobliżu słowa „termoelektryk", ale występuje wraz ze słowami „chalkogenki", „przerwa energetyczna" i „optoelektryki", które są wskazówkami pozwalającymi na zaklasyfikowanie go jako podobnego do „termoelektryka".

Co więcej, po wytrenowaniu wyłącznie na bazie abstraktów nieprzekraczających 2008 roku i zażądaniu wybrania związków, które są „termoelektryczne", wśród pięciu czołowych zwróconych przez model znalazły się trzy, które za termoelektryki uznane zostały dopiero w literaturze opublikowanej w latach 2009 – 2019. Tak oto model lingwistyczny antycypował o wiele lat oficjalne odkrycie na gruncie doświadczalnym.

[6] „Wektor znormalizowany" to wektor o długości 1. Iloczyn skalarny dwóch wektorów równy jest iloczynowi ich długości oraz kosinusa kąta między nimi. Dla wektorów prawie równoległych kąt ten jest bliski 0, a jego kosinus jest bliski 1 — *przyp. tłum.*

24.5.2. Wstępny trening reprezentacji kontekstu zdaniowego

Embeddingi w opisywanej dotychczas formie są *embeddingami absolutnymi* w tym sensie, że nie odzwierciedlają *znaczenia* słów wynikającego z *kontekstu zdań*, w których zostały użyte, co staje się problemem w przypadku słów wieloznacznych (polisemicznych). Przykładowo, słowo *rose* może być rzeczownikiem „róża", ale może też być formą czasu przeszłego czasownika *rise* („pojawić się", „wzrastać", „wschodzić"). Spodziewamy się zatem odnaleźć co najmniej dwa całkowicie różne klastery kontekstów zdaniowych dla *rose*: jeden skupiający wyrazy „kwiatopodobne" — *dahlia* („georginia"), *freesia* („frezja"), *geranium* („pelargonia") — i drugi skupiający wyrazy bliskoznaczne do *rise* (na przykład *upsurge* — „ożywienie, rozkwit"); nie jest możliwe odzwierciedlenie obu tych znaczeń w formie pojedynczego embeddingu. *Rose* to tylko jeden przykład słowa o (co najmniej) dwóch całkowicie różnych znaczeniach, lecz język angielski zawiera wiele bardziej subtelnych wieloznaczności, silnie zależnych od kontekstu znaczeniowego: słowo *need*, kojarzące się z „potrzebą", ma nieco odmienne znaczenie w zdaniach *you need to see this movie* („powinieneś zobaczyć ten film") i *humans need oxygen to survive* („ludzie potrzebują tlenu do przeżycia"). Jeszcze ciekawszym przykładem wieloznaczności są konstrukcje idiomatyczne — wyrażenie *break the bank* („rozbić bank") ma znaczenie kompletnie inne niż sugerowane przez każde z jego słów składowych.

Zamiast więc trenować model w oparciu o tablicę odwzorowań „słowo → embedding", powinniśmy prowadzić trening oparty na reprezentacji odzwierciedlającej **kontekst znaczeniowy** poszczególnych słów tworzących zdanie. Taka reprezentacja danego słowa, zwana jego **embeddingiem znaczeniowym**, to embedding stanowiący wynik mapowania słowa *wraz z otaczającym je kontekstem znaczeniowym*; innymi słowy, jeśli przekażemy modelowi słowo *rose* i kontekst *the gardener planted a rose bush* („ogrodnik zasadził krzew róży"), model ten powinien wyprodukować embedding znaczeniowy podobny do tego odpowiadającego kontekstowi *the cabbage rose had an unusual fragrance* („róża stulistna pachniała niezwykle"), ale zupełnie inny niż dla kontekstu zdania *the river rose five feet* („poziom wody w rzece podniósł się o pięć stóp").

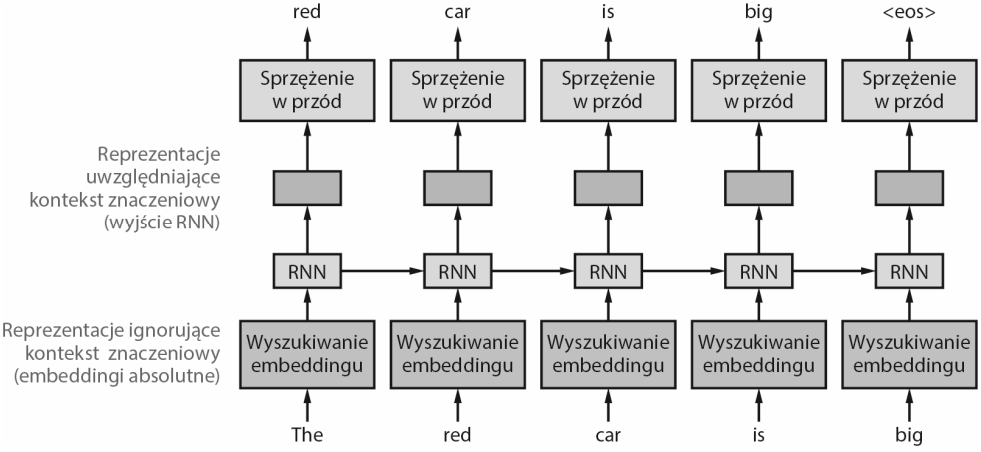
Rysunek 24.10 przedstawia sieć RNN, która tworzy embeddingi znaczeniowe słów zdania *The red car is big* („czerwony samochód jest duży") — embeddingi te przedstawione są w postaci nieetykietowanych prostokątów. Zakładamy, że zbudowaliśmy już kolekcję embeddingów absolutnych. Podajemy jedno słowo na raz i żądamy od modelu, aby przewidział następne słowo. W przykładzie na rysunku w punkcie, w którym dotarliśmy do słowa *car*, węzeł RNN w tym kroku czasowym otrzyma dwie dane wejściowe: embedding absolutny słowa *car* oraz kontekst, który koduje informacje z poprzednich słów: *The red*. Węzeł RNN wygeneruje następnie kontekstową reprezentację dla *car*, a sieć jako całość wyprowadza przewidywanie dla następnego słowa: *is*. Potem aktualizujemy wagi sieci, tak by zminimalizować błąd między przewidywaniem a faktycznie występującym następnym słowem.

Ten model jest podobny do modelu tagowania części mowy (POS) z rysunku 24.4, z dwiema istotnymi różnicami. Po pierwsze, model ten jest jednokierunkowy (od lewej do prawej), natomiast model POS jest dwukierunkowy. Po drugie, zamiast przewidywać tagi POS dla bieżącego słowa, model ten przewiduje następne słowo przy użyciu poprzedniego kontekstu. Po zbudowaniu modelu możemy go używać do wyszukiwania reprezentacji słów i przekazywania ich do innych zadań; nie musimy już przewidywać następnego słowa. Zauważmy, że obliczanie reprezentacji kontekstu zdaniowego zawsze wymaga dwóch danych wejściowych — bieżącego słowa i kontekstu jego występowania.

24.5.3. Maskowane modele językowe

Słabością standardowych modeli językowych, takich jak modele *n*-gramowe, jest to, że kontekstualizacja każdego słowa opiera się tylko na słowach poprzedzających — przewidywania tworzone są od lewej do prawej. Niekiedy jednak kontekst danego słowa staje się zrozumiały dopiero po wyjaśnieniach dostarczanych przez późniejsze zdania, na przykład w zdaniu „Należy pozamykać okna na pulpicie" wyjaśnienia takiego dostarcza słowo „pulpit".

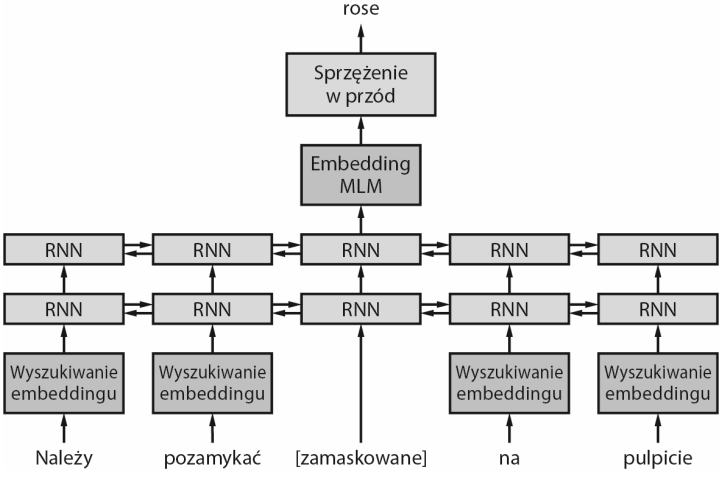
Jednym z prostych sposobów obejścia tego problemu jest wytrenowanie oddzielnego modelu języka, zapisywanego od strony prawej do lewej, który to model kontekstualizuje każde słowo na podstawie następnych słów w zdaniu; potem można połączyć oba modele w celu otrzymania połączonej reprezentacji dla obu kierunków. Nie da się jednak w ten sposób zintegrować materiałów dowodowych wnoszonych przez każdy z kierunków.



RYSUNEK 24.10. Trenowanie reprezentacji kontekstu zdaniowego w kierunku od lewej do prawej (*The red car is big* — „czerwony samochód jest duży").

Zamiast tego możemy użyć **modelu języka maskowanego** (ang. MLM — *Masked Language Model*). Modele MLM trenowane są przez maskowanie (ukrywanie) wybranych słów w danych wejściowych, a następnie żądanie od modelu, by owe zamaskowane słowa odgadywał. Do realizacji tego zadania można użyć głębokiej dwukierunkowej sieci RNN lub transformera w odniesieniu do zdania wejściowego. Na przykład w zdaniu „Należy pozamykać okna na pulpicie" można zamaskować „okna", uzyskując „należy pozamykać ___ na pulpicie", i zażądać od modelu wypełnienia powstałej luki.

Przykład zastosowania modelu języka maskowanego przedstawia rysunek 24.11.



RYSUNEK 24.11. Modelowanie języka maskowanego: wstępny trening modelu dwukierunkowego — na przykład wielowarstwowej sieci RNN — przez maskowanie niektórych słów i następnie ich odgadywanie

Finalne ukryte wektory odpowiadające zamaskowanym tokenom są następnie używane do przewidywania zamaskowanych słów — w tym przykładzie „okna". W procesie trenowania jedno zdanie może być użyte wiele razy, z różnymi pozycjami maskowania. Piękno tego podejścia polega na tym, że nie wymaga ono uprzedniego etykietowania danych — samo zdanie stanowi etykietę dla zamaskowanego słowa. Jeśli przeprowadza się taki trening na dużym korpusie tekstowym, można otrzymać zbiór wstępnie wytrenowanych reprezentacji, użytecznych w wielu zadaniach NLP — tłumaczeniu maszynowym, odpowiadaniu na pytania, podsumowaniach treści, ocenie poprawności gramatycznej i wielu innych.

24.6. Obecny stan sztuki

Głębokie uczenie i uczenie transferowe wydatnie przyczyniły się do podniesienia wiedzy w zakresie NLP na wyższy poziom — do tego stopnia, że jeden z komentatorów w 2018 roku oświadczył: „Nadszedł moment ImageNet NLP" (Ruder, 2018). Stwierdzenie to stanowi zapewne wspomnienie punktu zwrotnego w historii sztucznej inteligencji, gdy w 2012 roku zastosowanie głębokiego uczenia w systemach widzenia komputerowego przyniosło tym systemom zaskakująco dobre osiągnięcia w konkursie ImageNet. Podobny przełom dało się zauważyć właśnie w 2018 roku, gdy odkryto zadziwiającą użyteczność uczenia transferowego w zadaniach NLP. Dostępny jest bowiem gotowy ogólny model języka, który można pobrać i dostosować do specyfiki konkretnego zadania.

Zaczęło się od prostych systemów embeddingowych — WORD2VEC w 2013 roku i GloVe w 2014 roku. Badacze mogą pobierać gotowe modele tego typu bądź szybko trenować własne, i to bez konieczności wykorzystywania superkomputerów. Z drugiej strony, wstępnie wytrenowane reprezentacje kontekstowe są o rząd wielkości bardziej pracochłonne w treningu.

Modele te stały się realizowalne dopiero po szerokim upowszechnieniu nowych rozwiązań sprzętowych, głównie procesorów GPU i TPU, dzięki którym badacze mogli pobierać gotowe modele zamiast zużywać zasoby na trenowanie własnych. Model transformera pozwolił na efektywne trenowanie znacznie większych i znacznie głębszych niż wcześniej sieci neuronowych — tym razem dzięki rozwojowi raczej oprogramowania niż sprzętu. Od 2018 roku nowe projekty NLP zwykle rozpoczynają się od wstępnie wytrenowanego modelu transformera.

Chociaż te modele transformera zostały wytrenowane pod kątem przewidywania następnego słowa w tekście, sprawują się zaskakująco dobrze w innych zadaniach językowych. Model ROBERTA, po pewnym dostrojeniu, osiąga znakomite rezultaty w testach odpowiadania na pytania i czytania ze zrozumieniem (Liu i in., 2019b). Model językowy GPT-2, podobny do transformera, posiadający 1,5 miliarda parametrów, wytrenowany z użyciem tekstu internetowego o objętości ok. 40 gigabajtów, osiąga dobre wyniki w tak różnorodnych zadaniach, jak tłumaczenie maszynowe z francuskiego na angielski i odwrotnie, wyszukiwanie odwołań w zależnościach długodystansowych i odpowiadanie na pytania z zakresu wiedzy ogólnej, wszystko bez dostrajania do specyfiki konkretnego zadania. Jak ilustruje tabela 24.2, GPT-2 może generować dość przekonujący tekst na podstawie zaledwie kilku słów promptu.

Nowoczesny system NLP — ARISTO (Clark i in., 2019) — uzyskał wynik 91,6% w teście na poziomie ośmioklasisty (patrz tabela 24.3). System ten składa się z zespołu solverów, z których część wyszukuje informację (podobnie do wyszukiwania w sieci Web), część realizuje wynikanie tekstowe i wnioskowanie jakościowe, niektóre natomiast używają dużych modeli transformerowych. Okazało się, że sam model ROBERTA osiągnął w tym teście wynik 88,2%. W treści bardziej zaawansowanej, na poziomie maturzysty, ARISTO osiąga 83%; wynik 65% oznacza „spełnia normy" (*meeting the standards*), a wynik 85% „spełnia normy z wyróżnieniem" (*meeting the standards with distinction*).

TABELA 24.2. Przykładowe „dokończenia" tekstu generowane przez model GPT-2 w reakcji na różne prompty (wyróżnione przez **pogrubienie**). Większość generowanego tekstu jest płynną narracją w języku angielskim (przynajmniej lokalnie), a sensowność zdań umożliwia niemal wierne ich przetłumaczenie na język polski. Ostatni przykład pokazuje, że w niektórych przypadkach model może się załamywać

It is a truth universally acknowledged *that the earth is not the center of the universe. There are those who assert there is. I do not accept them, but others I consider to be of the same opinion. The truth is, however, that if there are other than the center, and if there are any other living things in the universe and if they are not human, then we do not yet have our answers. We have to go on. This page gives a simplified, simplified answer to the problem. We don't have all the answers. The truth is, however, that the truth is out there.*
Jest powszechnie uznaną prawdą, że Ziemia nie jest środkiem wszechświata. Są tacy, którzy twierdzą, że jest. Nie zgadzam się z nimi, ale uważam, że inni podzielają tę opinię. Prawda jest jednak taka, że jeśli istnieją inne centra i jeśli istnieją we wszechświecie jakieś żywe istoty niebędące ludźmi, to nie znamy jeszcze [prawidłowej] odpowiedzi, musimy [jej] dalej poszukiwać. Ta strona daje bardzo uproszczoną odpowiedź na problem. Nie znamy wszystkich odpowiedzi, lecz zapewne prawda leży gdzie indziej.
When Gregor Samsa woke up one morning*, he did not notice anything strange. "When my wife is looking at me, I feel like she is looking at a piece of art," he said. "I think she is admiring something I have created." The idea is that by looking at your own life, you learn something important and become a better person. It is a theory that emerged from psychologist Daniel Goleman's work, in which he asked "How do you know you're not a loser?"*
Kiedy Gregor Samsa obudził się pewnego ranka, nie zauważył niczego dziwnego. „Kiedy moja żona patrzy na mnie, mam wrażenie, że patrzy na dzieło sztuki" — powiedział. „Myślę, że podziwia coś, co stworzyłem". A chodzi o to, że patrząc na własne życie, uczysz się czegoś ważnego i stajesz się lepszym człowiekiem. Jest to teoria stworzona przez psychologa Daniela Golemana, który w jednej ze swych prac zapytywał: „Skąd wiesz, że nie jesteś przegrany?"
Alice was beginning to get very tired of sitting with her sister on the bank*. She sat up, yawned, and said, with a loud little scream, „I hope you don't mind if I keep on doing what I should like to do, and if someone asks me which of us will do more, don't tell them that I won't do much, my dear sister".*
Alicję zaczynało bardzo męczyć siedzenie z siostrą na skarpie. Usiadła, ziewnęła i powiedziała, niemalże krzycząc „Mam nadzieję, że nie masz nic przeciwko temu, że będę dalej robić to, co powinnam, a jeśli ktoś zapyta mnie, kto z nas zrobi więcej, nie mów im, droga siostro, że nie zrobię zbyt wiele"
All happy families are alike*; each happy family is like a garden of paradise. The only difference between happy families and unhappy families, is that the unhappy family doesn't have any flowers or trees.*
Wszystkie szczęśliwe rodziny są takie same; każda szczęśliwa rodzina jest jak rajski ogród. Jedyna różnica między rodziną szczęśliwą a nieszczęśliwą polega na tym, że w nieszczęśliwej nie rosną żadne kwiaty ani drzewa.
Tell me a story*. Tell me a story. Tell me a story. Tell me a story. Tell me a story. Tell me a story. Tell me a story. Tell me a story. Tell me a story. Tell me a story. Please fill out the following details. Thank you... Thank you for your interest in this interview. Please wait...*
Opowiedz mi historię. Opowiedz mi historię. Opowiedz mi historię. Opowiedz mi historię. Opowiedz mi historię. Opowiedz mi historię. Opowiedz mi historię. Opowiedz mi historię. Opowiedz mi historię. Opowiedz mi historię. Uzupełnij proszę poniższe szczegóły. Dziękuję... Dziękuję za zainteresowanie tym wywiadem. Poczekaj proszę....

TABELA 24.3. Przykład testu szkolnego na poziomie ośmioklasisty, który system ARISTO potrafi poprawnie rozwiązać wykorzystując zespół metod, z modelem ROBERTA na czele. Poprawne rozwiązanie takiego testu wymaga znajomości języka naturalnego, posiadania tzw. wiedzy ogólnej i elementarnej wiedzy naukowej oraz zasad konstruowania testów tego rodzaju

1. Za pomocą czego najlepiej rozdzielić mieszaninę opiłków żelaza i czarnego pieprzu?
(a) magnesu (b) papierowego filtru (c) wagi laboratoryjnej[7] (d) woltomierza
2. Jaką formę energii wytwarza drgająca gumka-recepturka?
(a) chemiczną (b) świetlną (c) elektryczną (d) akustyczną
3. Miedź jest metalem, zatem
(a) jest cieczą w temperaturze pokojowej (b) nie reaguje z innymi substancjami
(c) jest kiepskim przewodnikiem prądu (d) jest dobrym przewodnikiem ciepła
4. Który proces zachodzący w jabłoni jest najważniejszym skutkiem podziału komórkowego?
(a) wzrost (b) fotosynteza (c) wymiana gazowa (b) wydalanie produktów przemiany materii

ARISTO ma jednak ograniczenia: radzi sobie tylko z testami wyboru spośród gotowych odpowiedzi, nie z zadaniami „otwartymi" (opisowymi), nie potrafi też czytać ani generować diagramów[8].

System T5 (*Text-to-Text Transfer Transformer*) przeznaczony jest do generowania odpowiedzi tekstowych na wejściowe dane tekstowe różnego rodzaju. Zawiera standardowy model kodera-dekodera transformerowego, wstępnie wytrenowany na 35 miliardach słów pochodzących z 750-gigabajtowego korpusu C4 (*Colossal Clean Crawled Corpus*). Ten nieetykietowany trening zaprojektowany został w celu wyposażenia modelu w wiedzę językową, którą można generalizować i która będzie przydatna do wielu specyficznych zadań. T5 jest następnie trenowany dla każdego zadania z użyciem danych tekstowych, których każda pozycja ma postać

nazwa zadania **:** *treść*

na przykład polecenie

translate English to German **:** *That is good*

daje w wyniku *Das ist gut.* W przypadku niektórych zadań dane wejściowe są znakowane, na przykład w tekście *Winograd Schema Challenge* zadanie polegało na określeniu podmiotu, do którego odnosi się zaimek *they* w zdaniu

referent **:** *The city councilmen refused the demonstrators a permit because* ***they*** *feared violence*

(„radni miejscy odmówili demonstrantom zezwolenia, ponieważ obawiali się przemocy"). Odpowiedzią jest *councilmen* („radni"), ale już w bardzo podobnym zdaniu

referent **:** *The city councilmen refused the demonstrators a permit because* ***they*** *advocated violence*

(„radni miejscy odmówili demonstrantom zezwolenia, ponieważ ci nawoływali do przemocy") *they* odnosi się do „demonstrantów". Różnica ta nie wynika w żaden sposób z gramatyki języka, lecz z ogólnej wiedzy o otaczającym świecie

[7] W oryginale *triple beam balance* — „waga przesuwnikowa jednoszalkowa trójramienna" — *przyp. tłum.*

[8] Zauważono, że w przypadku niektórych testów wielokrotnego wyboru możliwe jest uzyskanie dobrego wyniku nawet *bez związku z pytaniami*, bo w błędnych wariantach wyboru mogą skrywać się elementy demaskujące ich zwodniczy charakter (Gururangan i in., 2018). Wydaje się, że dotyczy to również odpowiedzi na pytania wizualne (Chao i in., 2018).

— to nie radni chcieli manifestować poparcie dla przemocy, skoro zgody na manifestację nie wydali, zresztą, jak pokazuje doświadczenie, jeśli przemoc ma miejsce, to rodzi się przeważnie ze spontanicznych manifestacji, a nie odgórnych zarządzeń władz municypalnych, a w ogóle w zdaniu jest mowa o demonstrantach. Komputerom jednak z natury obca jest wiedza tego rodzaju, dlatego wzmiankowany test postrzegany jest jako najważniejsza alternatywa dla oryginalnego testu Turinga[9].

Na polu ulepszania systemów NLP wciąż wiele pozostaje do zrobienia. Jednym z podstawowych problemów większości modeli transformerów jest ograniczoność kontekstu — opierają się one na wąskim kontekście, nieprzekraczającym kilkuset słów. Próba złagodzenia tego limitu jest przedmiotem kilku podejść eksperymentalnych, między innymi system Reformer (Kitaev i in., 2020) zdolny jest do obsługi kontekstu o długości do miliona słów.

Ostatnie wyniki eksperymentów pokazały, że użycie większej ilości danych treningowych owocuje lepszymi modelami — na przykład model ROBERTA osiągnął imponujące wyniki po wytrenowaniu na zbiorze liczącym 2,2 biliona słów. A skoro tak zachęcające wyniki uzyskuje się dzięki danym wyłącznie tekstowym, to co można by osiągnąć, sięgając po dane innego rodzaju — ustrukturyzowane bazy danych, dane liczbowe, obrazy i wideo? Niewątpliwie do efektywnego spożytkowania tego panoptikum informacji konieczny byłby kolejny przełom w możliwościach sprzętu obliczeniowego, głównie w szybkości procesorów i pojemności pamięci. Stosowny przełom musiałby się także dokonać w metodologiach sztucznej inteligencji.

Wnikliwi Czytelnicy zapewne zapytają w tym momencie, z nutką zdziwienia, dlaczego w poprzednim rozdziale tak szczegółowo omawialiśmy gramatyki, analizę syntaktyczną i interpretację semantyczną, skoro w świetle treści niniejszego rozdziału deprecjonujemy te koncepcje na rzecz modeli opartych wyłącznie na danych? Szybka odpowiedź powinna eksponować bezsprzeczne zalety systemów sterowanych danymi: łatwość budowania i rozwijania, wspaniałe osiągi w standardowych benchmarkach — pod tym względem ustępują im wyraźnie systemy budowane ręcznie, przy użyciu rozsądnego wysiłku ludzkiego, oparte na podejściach opisywanych w rozdziale 23. Może być tak dlatego, że modele transformerów i ich pochodne uczą się ukrytych reprezentacji wychwytujących te same podstawowe idee co gramatyki i informacje semantyczne. Nie można też wykluczyć, że w tych ogromnych modelach dzieje się coś zupełnie innego, o czym po prostu (jeszcze) nie wiemy. Wiemy natomiast, że system wytrenowany przy użyciu danych tekstowych jest łatwiejszy w utrzymaniu i przystosowywaniu do nowych domen i nowych języków naturalnych niż system, który opiera się na ręcznie definiowanych cechach.

Niewykluczone, że przyszłe przełomy w jawnym modelowaniu gramatycznym i semantycznym spowodują odwrócenie tej tendencji, bardziej jednak prawdopodobne jest ukształtowanie się podejść hybrydowych, integrujących najlepsze koncepcje z obu wymienionych światów. Kitaev i Klein (2018) wykorzystali mechanizm uwagi do ulepszenia tradycyjnego parsera składnikowego, osiągając najlepszy wynik, jaki zarejestrowano kiedykolwiek przy użyciu zestawu testowego Penn Treebank. Podobnie Ringgaard i in. (2017) pokazują, w jaki sposób można ulepszyć parser zależności za pomocą embeddingów i rekurencyjnej sieci neuronowej. Ich system — SLING — przeprowadza parsowanie bezpośrednio do reprezentacji w postaci ramek semantycznych, łagodząc problem kumulowania się błędów w tradycyjnym systemie potokowym.

Systemy NLP, mimo niekwestionowanych osiągnięć, wciąż ustępują człowiekowi pod względem wydajności w wielu zadaniach, chociaż każdy z nich trenowany jest przy użyciu takiej objętości tekstu, jakiej człowiek nie byłby w stanie przeczytać w ciągu całego swojego życia. Stanowi to inspirujące pole do nowych pomysłów, odkryć i wynalazków dla językoznawców, psychologów i oczywiście badaczy NLP.

[9] Patrz *https://pol.phonecardsoftware.com/s2468-winograd-schema-challenge-a-turing-test-alternative-puts-ai-programs-to-shame* — mimo nieco dziwnego (automatycznego) tłumaczenia tekst jest całkiem zrozumiały — *przyp. tłum.*

Podsumowanie

W tym rozdziale omawialiśmy następujące kluczowe aspekty związków głębokiego uczenia z przetwarzaniem języka naturalnego:

- Reprezentacje ciągłe słów, z embeddingami na czele, są bardziej użyteczne od dyskretnych reprezentacji atomowych, można je bowiem trenować przy użyciu nieetykietowanych danych tekstowych.
- Lokalny i długodystansowy kontekst można efektywnie modelować za pomocą rekurencyjnych sieci neuronowej, przechowując w jej ukrytych wektorach relewantne informacje.
- Modele „sekwencja na sekwencję" okazują się efektywne w rozwiązywaniu problemów tłumaczenia maszynowego i generowania tekstu.
- Modele transformerów wykorzystujące samouwagę mogą być wykorzystywane do modelowania kontekstu (lokalnego i długodystansowego) z dużą efektywnością dzięki możliwości wykorzystywania operacji macierzowych implementowanych w nowoczesnym sprzęcie obliczeniowym.
- Uczenie transferowe wykorzystujące wytrenowane wstępnie embeddingi słów umożliwia budowanie modeli w oparciu o ogromne nieetykietowane korpusy i wykorzystywanie tych modeli do rozwiązywania różnorodnych zadań. Modele wytrenowane wstępnie do przewidywania brakujących słów okazują się użyteczne także w realizacji innych zadań, takich jak odpowiadanie na pytania i wynikanie tekstowe, po dostrojeniu do specyfiki docelowych domen.

Bibliografia i uwagi historyczne

Rozkład słów i fraz w języku naturalnym jest zgodny z **prawem Zipfa** (Zipf, 1935, 1949), zgodnie którym słowo zajmujące *n*-tą pozycję w rankingu występowania pojawia się z częstotliwością w przybliżeniu odwrotnie proporcjonalną do *n*. Oznacza to, że mamy problem ze słowami rzadko występującymi: analizując nawet ogromne, liczone w miliardach słów dane treningowe wciąż można napotykać nowe słowa i frazy nienapotkane wcześniej w tych danych.

Generalizację na nowe słowa i frazy ułatwiają reprezentacje hołdujące zasadzie, że słowa o podobnym znaczeniu pojawiają się najczęściej w podobnych kontekstach. Deerwester i in. (1990) dokonali rzutowania słów na wektory niskowymiarowe, dekomponując macierz współwystępowania utworzoną na podstawie słów oraz dokumentów, w których słowa te występowały. Inną możliwością jest potraktowanie otaczających słów, w odległości ograniczonej oknem o szerokości (powiedzmy) pięciu słów, jako kontekstu tychże słów.

Brown i in. (1992) pogrupowali słowa w hierarchiczne klastery zgodnie z kontekstem bigramowym tych słów, co okazało się skuteczne w przypadku zadań takich jak rozpoznawanie nazwanych encji (Turian i in., 2010). System WORD2VEC (Mikolov i in., 2013) był pierwszą znaczącą demonstracją zalet embeddingów budowanych w procesie trenowania sieci neuronowych. Embeddingi systemu GloVe (Pennington i in., 2014) uzyskano, operując bezpośrednio na macierzy współwystępowania słów zbudowanej na podstawie miliardów słów tekstu. Levy i Goldberg (2014) wyjaśniają, skąd bierze się zdolność embeddingów do odzwierciedlania regularności językowych.

Bengio i in. (2003) to autorzy pionierskiego pomysłu związanego w zastosowaniem sieci neuronowych do modeli językowych, polegającego na (cyt.) „połączeniu (1) rozproszonej reprezentacji każdego słowa oraz (2) funkcji prawdopodobieństwa dla sekwencji słów, wyrażonej w kategoriach tych reprezentacji". Mikolov i in. (2010) zademonstrowali zastosowanie sieci RNN do modelowania lokalnego kontekstu w modelach językowych. Jozefowicz i in. (2016) pokazali, w jaki sposób sieć RNN wytrenowana na miliardzie słów może przewyższyć wydajnością starannie zaprojektowane „ręczne" modele *n*-gramowe. Kontekstowe reprezentacje słów zostały szczegółowo opisane przez Petersa i in. (2018), którzy opatrzyli je akronimem ELMo (***E**mbeddings from **L**anguage **Mo**dels*).

Należy w tym miejscu zaznaczyć, że niektórzy autorzy porównują modele językowe pod względem ich **powikłania** (ang. *perplexity*). Powikłanie rozkładu prawdopodobieństwa wynosi 2^H, gdzie H jest entropią rozkładu (patrz sekcja 19.3.3). Spośród dwóch modeli lepszy jest ten o mniejszym powikłaniu, pod warunkiem identyczności pozostałych parametrów. W praktyce jednak bardzo rzadko daje się zapewnić ową „identyczność", dlatego bardziej miarodajnym kryterium porównawczym jest wydajność modeli w zastosowaniu do rzeczywistych zadań.

Howard i Ruder (2018) opisują framework ULMFiT (ang. ***U**niversal **L**anguage **M**odel **Fi**ne-**t**uning*), ułatwiający dostrajanie wstępnie wytrenowanego modelu językowego, bez konieczności posługiwania się dużym zbiorem dokumentów z domeny docelowej. Artykuł Rudera i in. z 2019 roku to tutorial na temat zastosowań uczenia transferowego w zadaniach NLP.

Mikolov i in. (2010) przedstawili ideę wykorzystania sieci RNN w zadaniach NLP, a Sutskever i in. (2015) wprowadzili ideę uczenia „sekwencja na sekwencję" z wykorzystaniem głębokich sieci neuronowych. Zhu i in. (2017) oraz Liu i in. (2018b) wykazali, że podejście nienadzorowane faktycznie działa i znacznie ułatwia kolekcjonowanie danych. Bardzo szybko okazało się, że tego rodzaju modele potrafią zaskakująco dobrze radzić sobie z różnymi zadaniami, na przykład tworzeniem podpisów pod obrazami (Karpathy i Fei-Fei, 2015; Vinyals i in., 2017b).

Devlin i in. (2018) wykazali, że modele transformerów wstępnie wytrenowane pod kątem języka maskowanego mogą być bezpośrednio wykorzystywane do wielu zadań. Opisywany model opatrzyli oni akronimem BERT (ang. *Bidirectional Encoder Representations from Transformers*); wstępnie wytrenowane modele tego typu można dostosowywać do określonych domen i określonych zadań, w tym do odpowiadania na pytania, rozpoznawania nazwanych encji, klasyfikowania tekstu, analizy sentymentu i wnioskowania w języku naturalnym.

System XLNET (Yang i in., 2019) to ulepszenie modelu BERT, eliminujące rozbieżność między treningiem wstępnym a dostrajaniem. Framework ERNIE 2.0 (Sun i in., 2019) wydobywa więcej informacji z danych treningowych, dzięki uwzględnianiu kolejności zdań i obecności nazwanych encji, a nie tylko współwystępowania słów; wykazano, że przewyższa on zarówno BERT-a, jak i XLNET. Odpowiedzią twórców BERT-a było jego ponowne przeanalizowanie i ulepszenie — nowy system ROBERTA (Liu i in., 2019b) wykorzystywał więcej danych, różne hiperparametry oraz rozmaite procedury treningowe, dzięki czemu zdołał dorównać wydajnością systemowi XLNET. System Reformer (Kitaev i in., 2020) rozszerza zakres uwzględnianego kontekstu aż do miliona słów. Tymczasem ALBERT (ang. *A Lite BERT*) to krok w innym kierunku — zmniejszenia liczby parametrów ze 108 mln do 12 mln, by system zmieścił się na urządzeniu mobilnym, bez uszczerbku dla (wysokiej) dokładności.

System XLM (Lample i Conneau, 2019) to model transformera z danymi treningowymi w wielu językach. Wielojęzyczność ta jest oczywiście użyteczna w zadaniach tłumaczenia maszynowego, ale zapewnia również bardziej niezawodne reprezentacje w zadaniach jednojęzycznych. W niniejszym rozdziale opisaliśmy dwa inne ważne systemy, GPT-2 (Radford i in., 2019) oraz T5 (Raffel i in., 2019); w drugim z wymienionych artykułów opisano również korpus C4 (*Colossal Clean Crawled Corpus*) z 35 miliardami słów.

Zaproponowano różne obiecujące ulepszenia algorytmów wstępnego treningu (Yang i in., 2019; Liu i in., 2019b). Wstępnie wytrenowane modele kontekstowe opisywane są w artykułach Petersa i in. (2018) oraz Dai i Le (2016).

Benchmark GLUE (ang. *General Language Understanding Evaluation*) to zbiór wzorcowych zadań i narzędzi do ewaluacji systemów NLP; został skonstruowany przez Wanga i in. (2018a). Wspomniane zadania obejmują odpowiadanie na pytania, analizę sentymentu, wynikania tekstowego, tłumaczenie maszynowe i parsowanie. Modele transformerów zdominowały ranking systemów NLP, przeciętne ludzkie możliwości porównywalne są z dziewiątą pozycją tego rankingu. GLUE zyskał godnego następcę o nazwie SUPERGLUE (Wang i in., 2019), którego wzorcowe zadania zaprojektowane zostały jako trudniejsze dla komputerów, lecz łatwe jak dotąd dla ludzi.

U schyłku 2019 roku T5 był ogólnym liderem wspomnianego rankingu, uzyskując wynik 89,3 — zaledwie 0,5 poniżej linii bazowej 89,8. W przypadku trzech z dziesięciu zadań T5 okazał się lepszy od człowieka. Pierwsze z tych zadań polegało na udzieleniu odpowiedzi typu „tak/nie" („Czy Francja należy do tej samej strefy czasowej co Wielka Brytania?"), dwa pozostałe sprowadzały się do czytania ze zrozumieniem, czyli przeczytania (odpowiednio) akapitu publikacji i artykułu prasowego, a następnie odpowiedzenia na pytania związane z ich treścią.

Tłumaczenie maszynowe to główne zastosowanie modeli językowych. Już w 1933 roku Petr Troyanskii otrzymał patent na „maszynę tłumaczącą", ale była to jedynie idea, której nie sposób było urzeczywistnić, z prostego powodu — nie było jeszcze komputerów. W 1947 Warren Weaver, opierając się na pracach z kryptografii i teorii informacji, napisał do Norberta Wienera:

> *Gdy patrzę na artykuł napisany w języku rosyjskim, myślę sobie: „Ten artykuł napisano w języku angielskim, ale został on zakodowany w postaci dziwacznych symboli, które muszę zdekodować".*

Owa koncepcja „tłumaczenia przez dekodowanie", sama z siebie interesująca, okazała się jednak nierealizowalna w praktyce, ze względu na brak wystarczających danych i możliwości ówczesnej infrastruktury obliczeniowej.

W latach siedemdziesiątych ubiegłego wieku ten stan rzeczy zaczął się to zmieniać, a system SYSTRAN (Toma, 1977) był pierwszym udanym komercyjnym systemem tłumaczenia maszynowego. Opierał się na regułach leksykalnych i gramatycznych opracowanych ręcznie przez lingwistów, a także na danych treningowych. W latach 80. społeczność NLP przyjęła modele czysto statystyczne, oparte na częstotliwości występowania słów i fraz (Brown i in., 1988; Koehn, 2009). Gdy zbiory treningowe osiągnęły miliardy lub biliony tokenów (Brants i in., 2007), pojawiły się systemy, które dawały wyniki zrozumiałe, ale jednak dalekie od płynności (Och i Ney, 2004; Zollmann i in., 2008). Och i Ney (2002) pokazują, jak trenowanie dyskryminatywne doprowadziło do postępów w tłumaczeniu maszynowym na początku XXI wieku.

Sutskever i in. (2015) po raz pierwszy wykazali, że jest możliwe uczenie kompleksowego modelu neuronowego „sekwencja na sekwencję" pod kątem tłumaczenia maszynowego. Bahdanau i in. (2015) wykazali zalety modelu, który uczy się dopasowywać zdania w języku źródłowym i docelowym oraz dokonywać tłumaczeń między tymi językami. Vaswani i in. (2018) wykazali, że neuronowe systemy tłumaczenia maszynowego można dalej ulepszać poprzez zastąpienie pamięci LSTM architekturą transformerów, wykorzystującą mechanizm uwagi do wychwytywania kontekstu. Te neuronowe systemy tłumaczenia rychło zdystansowały statystyczne metody oparte na frazach, a architektura transformerów wkrótce rozprzestrzeniła się na inne zadania NLP.

Wielce pomocnym w badaniach nad problemami **odpowiadania na pytania** okazało się stworzenie repozytorium SQuAD (ang. ***S**tanford **Qu**estionn **A**nswering **D**ataset*)[10] — pierwszego wielkoskalowego zestawu danych do trenowania i testowania takich systemów (Rajpurkar i in., 2016). Od tego czasu opracowano szereg modeli głębokiego uczenia dedykowanych temu zadaniu (Seo i in., 2017; Keskar i in., 2019). System ARISTO (Clark i in., 2019) wykorzystuje głębokie uczenie w połączeniu z zestawem innych taktyk. Począwszy od 2018 roku większość modeli odpowiadania na pytania korzysta z wytrenowanych wstępnie reprezentacji językowych, co prowadzi do zauważalnej poprawy w stosunku do wcześniejszych systemów tej kategorii.

Wnioskowanie na podstawie języka naturalnego polega na ocenie, czy dana hipoteza (*dogs need to eat* — „psy potrzebują pożywienia") wynika z określonej przesłanki (*all animals need to eat* — „wszystkie zwierzęta potrzebują pożywienia"). Zadanie to stało się szczególnie popularne dzięki konkursom PASCAL Challenge (Dagan i in., 2005) i obecnie dostępne są dedykowane mu zbiory danych na dużą skalę (Bowman i in., 2015; Williams i in., 2018). Szczególnie dobrą wydajność zapewniają systemy oparte na wstępnie wytrenowanych modelach, takich jak ELMo i BERT.

Konferencja CoNLL (ang. *Conference on Computational Natural Language Learning*) koncentruje się — zgodnie z nazwą — na problemach uczenia maszynowego związanych z ogólnie pojętym przetwarzaniem języka naturalnego. Także wszelkie czasopisma i konferencje wspominane w rozdziale 23. obejmują tematykę głębokiego uczenia, które obecnie jest dominującym kierunkiem badan na polu NLP.

[10] Patrz *https://rajpurkar.github.io/SQuAD-explorer/ — przyp. tłum.*

SKOROWIDZ

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

Ś

T

U

W

Z

ε

Sztuczna inteligencja: dokąd zmierzasz, technologio?

Praktycznie codziennie korzystamy z osiągnięć sztucznej inteligencji (ang. Artificial Intelligence – AI). Mimo to jej potencjał wciąż jest zagadką: nie wiemy, gdzie leżą granice jej rozwoju i jakie jeszcze technologie przyniesie nam ta relatywnie młoda dziedzina nauki. Równocześnie niektóre zastosowania sztucznej inteligencji budzą niepokój i zmuszają do zadawania trudnych pytań. Jakakolwiek próba odpowiedzi wymaga jednak wiedzy o tym, czym w istocie jest AI i jakie są jej ograniczenia.

To drugi tom klasycznego podręcznika wiedzy o sztucznej inteligencji. Podobnie jak w wypadku pierwszej części, lektura tej książki nie wymaga wybitnej znajomości tematu. Dzięki przejrzystości tekstu i umiejętnemu unikaniu nadmiernego formalizmu można w dość łatwy sposób zrozumieć kluczowe idee i koncepcje nauki o AI. Najnowsze technologiczne osiągnięcia zostały pokazane na tle rozwijającej się wiedzy, również z innych dziedzin inżynierii. Sporo miejsca poświęcono zagadnieniom, które budzą wątpliwości. Mowa tu o wyrafinowanych technikach uczenia maszynowego, modelach językowych czy widzeniu komputerowym, a także o sprawach, które już dziś wymagają najwyższej troski: o etycznych aspektach sztucznej inteligencji, bezpieczeństwie związanych z nią technologii i jej perspektywach.

W drugim tomie:

- różne modele i koncepcje uczenia maszynowego
- przetwarzanie języka naturalnego i modele językowe
- widzenie komputerowe, w tym generowanie obrazów
- roboty: percepcja, działanie, uczenie
- perspektywy sztucznej inteligencji

Stuart Russell jest znany ze swojego wkładu w naukę o sztucznej inteligencji. Jest profesorem na Uniwersytecie Kalifornijskim w Berkeley i założycielem Center for Human-Compatible AI. Laureat wielu prestiżowych nagród i autor kilkuset publikacji.

Peter Norvig jest wybitnym akademikiem, pracuje w Stanford Institute for Human-Centered AI. Był szefem Wydziału Nauk Obliczeniowych NASA Ames Research Center i dyrektorem do spraw badań w Google.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-283-7773-8
9 788328 377738
Cena: 129,00 zł
Pearson