UNIWERSYTET IM. ADAMA MICKIEWICZA

WYDZIAŁ MATEMATYKI I INFORMATYKI

Laura Guździoł

Nr albumu: 429410

Tłumaczenie automatyczne tekstów języka japońskiego

Machine translation of Japanese texts

Praca magisterska na kierunku:

INFORMATYKA

Promotor:

prof. UAM dr hab. Krzysztof Jassem

Poznań 2018

Poznań, dnia .....

OŚWIADCZENIE

Ja, niżej podpisana Laura Guździoł, studentka Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu oświadczam, że przedkładaną pracę dyplomową pt: *Tłumaczenie automatyczne tekstów języka japońskiego* napisałam samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałam z pomocy innych osób, a w szczególności nie zlecałam opracowania rozprawy lub jej części innym osobom, ani nie odpisywałam tej rozprawy lub jej części od innych osób. Oświadczam również, że egzemplarz pracy dyplomowej w wersji drukowanej jest całkowicie zgodny z egzemplarzem pracy dyplomowej w wersji elektronicznej. Jednocześnie przyjmuję do wiadomości, że przypisanie sobie, w pracy dyplomowej, autorstwa istotnego fragmentu lub innych elementów cudzego utworu lub ustalenia naukowego stanowi podstawę stwierdzenia nieważności postępowania w sprawie nadania tytułu zawodowego.

[TAK]* - wyrażam zgodę na udostępnianie mojej pracy w czytelni Archiwum UAM

[TAK]* - wyrażam zgodę na udostępnianie mojej pracy w zakresie koniecznym do ochrony mojego prawa do autorstwa lub praw osób trzecich

*Należy wpisać TAK w przypadku wyrażenia zgody na udostępnianie pracy w czytelni Archiwum UAM, NIE w przypadku braku zgody. Niewypełnienie pola oznacza brak zgody na udostępnianie pracy.

.....
(czytelny podpis studenta)

Streszczenie

W niniejszej pracy magisterskiej opisano proces tłumaczenia automatycznego z języka japońskiego na język polski z wykorzystaniem dwóch narzędzi do tworzenia systemów tłumaczących: Moses (tłumaczenie statystyczne) oraz Marian NMT (tłumaczenie neuronowe). Celem projektu magisterskiego było sprawdzenie, jakie czynniki poprawiają jakość tłumaczenia automatycznego dla tej pary języków, a także porównanie jakości obu metod tłumaczenia.

W pracy zawarto podstawowe informacje na temat języka japońskiego, statystycznego i neuronowego tłumaczenia automatycznego oraz opisano działanie narzędzi do tłumaczenia automatycznego – Moses i Marian NMT.

Słowa kluczowe

język japoński, program Marian NMT, program Moses, tłumaczenie automatyczne, tłumaczenie neuronowe, tłumaczenie statystyczne

Abstract

This thesis discusses the process of a machine translation from the Japanese language to the Polish language using tools for building translation systems: Moses (a statistical machine translation system) and Marian NMT (a neural machine translation system). The purpose of the project was to analyze factors that influence the process of the machine translation for languages mentioned above as well as to compare translation qualities of the two methods.

The thesis includes basic information about the Japanese language, statistical and neural machine translations and the description of the following tools: Moses and Marian NMT.

Key words

Japanese language, machine translation, Marian NMT tool, Moses tool, neural translation, statistical translation

Spis treści

1. Wstęp

W dzisiejszych czasach korzystanie z automatycznych systemów tłumaczących staje się coraz powszechniejsze. Dostęp do tekstów w obcych językach jest bardzo szeroki, stąd potrzeba szybkiego i prostego ich tłumaczenia. Dotyczy to zwłaszcza stron internetowych, które dostępne są dla każdego, w każdej chwili. Wraz ze wzrostem popularności kultury japońskiej w Polsce, a także większym zainteresowaniem podróżami do Japonii, rośnie również zapotrzebowanie na automatyczne tłumaczenie języka japońskiego na język polski. Jednak zadanie to nie należy do łatwych. Języki te bardzo się od siebie różnią, a w dodatku nie stanowią one priorytetu wśród badań nad tłumaczeniem automatycznym. Dlatego też jakość systemów tłumaczących z języka japońskiego na język polski jest stosunkowo niska.

Celem niniejszej pracy jest przybliżenie tematu tłumaczenia statystycznego i neuronowego w aspekcie tłumaczenia z języka japońskiego na język polski. Wspomniane dwie metody tłumaczenia automatycznego poddane zostały testom sprawdzającym wpływ różnych czynników na jakość tłumaczenia. Ważnym elementem jest również przeanalizowanie, która metoda tłumaczenia okaże się lepsza dla wybranej pary języków.

Praca została podzielona na 6 rozdziałów. Rozdział 2. przybliża podstawowe informacje o języku japońskim oraz o trudnościach związanych z tłumaczeniem automatycznym tego języka. W rozdziale 3. przedstawiono, na czym polega tłumaczenie automatyczne i wyróżniono jego dwa rodzaje: statystyczne i neuronowe, które zostały przetestowane w projekcie. W 4. rozdziale opisano narzędzia zastosowane w projekcie: program Moses oraz program Marian NMT. Pierwszy z nich służy do budowania systemów tłumaczenia statystycznego, a drugi do tworzenia systemów tłumaczenia neuronowego. Rozdział 5. zawiera opis projektu, w tym sposób pozyskania zasobów, zastosowanie programów, ewaluację wyników oraz wnioski z przeprowadzonego eksperymentu. W rozdziale 6. znajduje się podsumowanie pracy.

2. Podstawowe informacje o języku japońskim

Język japoński jest jednym z głównych języków świata, gdyż posługuje się nim prawie 130 milionów osób. Jego pochodzenie do dziś nie zostało jednoznacznie określone, ale istnieje na ten temat wiele teorii. Między innymi, część badaczy łączy język japoński z różnymi azjatyckimi językami, najczęściej z koreańskim, a inni z kolei uważają, że jest to całkowicie odmienna rodzina językowa. Język japoński znacznie różni się od indoeuropejskich, w tym również od polskiego. Największą różnicę prawdopodobnie stanowi pismo. Również gramatyka z charakterystyczną składnią wyróżnia japoński na tle innych języków.

Interesujący w języku japońskim jest fakt, że to jedyny język na świecie, który naturalnie może wykorzystywać w jednym zdaniu trzy, a czasem aż cztery, systemy pisma.

2.1. System pisma

W języku japońskim stosuje się najczęściej trzy systemy pisma: dwa sylabiczne, zwane *kana* (*hiragana* i *katakana*) oraz logograficzne o nazwie *kanji*.

あ a	い i	う u	え e	お o
か ka	き ki	く ku	け ke	こ ko
さ sa	し shi	す su	せ se	そ so
た ta	ち chi	つ tsu	て te	と to
な na	に ni	ぬ nu	ね ne	の no
は ha	ひ hi	ふ fu	へ he	ほ ho
ま ma	み mi	む mu	め me	も mo
や ya		ゆ yu		よ yo
ら ra	り ri	る ru	れ re	ろ ro
わ wa				を wo
				ん n
が ga	ぎ gi	ぐ gu	げ ge	ご go
ざ za	じ ji	ず zu	ぜ ze	ぞ zo
だ da	ぢ (ji)	づ (zu)	で de	ど do
ば ba	び bi	ぶ bu	べ be	ぼ bo
ぱ pa	ぴ pi	ぷ pu	ぺ pe	ぽ po

Tabela 1. System *hiragana*; https://pl.wikipedia.org/wiki/Hiragana; dostęp z dnia: 20.02.2018

Znaki w systemach *hiragana* (tabela 1.) oraz *katakana* (tabela 2.) przedstawiają te same sylaby, z tym, że pierwsze z nich są w zapisie bardziej zaokrąglone, a drugie składają się głównie z prostych linii.

ア a	イ i	ウ u	エ e	オ o
カ ka	キ ki	ク ku	ケ ke	コ ko
サ sa	シ shi	ス su	セ se	ソ so
タ ta	チ chi	ツ tsu	テ te	ト to
ナ na	ニ ni	ヌ nu	ネ ne	ノ no
ハ ha	ヒ hi	フ fu	ヘ he	ホ ho
マ ma	ミ mi	ム mu	メ me	モ mo
ヤ ya		ユ yu		ヨ yo
ラ ra	リ ri	ル ru	レ re	ロ ro
ワ wa	ヰ wi		ヱ we	ヲ wo
				ン n
ガ ga	ギ gi	グ gu	ゲ ge	ゴ go
ザ za	ジ ji	ズ zu	ゼ ze	ゾ zo
ダ da	ヂ (ji)	ヅ (zu)	デ de	ド do
バ ba	ビ bi	ブ bu	ベ be	ボ bo
パ pa	ピ pi	プ pu	ペ pe	ポ po

Tabela 2. System *katakana* (na czerwono zaznaczone są nieużywane już znaki); https://pl.wikipedia.org/wiki/Katakana; dostęp z dnia: 20.02.2018

Poszczególne kolumny tabel 1. i 2. prezentują: w pierwszym rzędzie samogłoski „a”, „i”, „u”, „e” oraz „o” i odpowiadające im znaki, a w kolejnych rzędach – sylaby kończące się na wspomniane samogłoski z odpowiadającymi im znakami. Wyjątek w obu systemach pisma wśród spółgłosek stanowi litera „n”, która występuje zarówno w sylabach, jak i samodzielnie. Ponadto, w japońskim piśmie nie istnieją odpowiedniki sylab „yi” (czyt. „ji”), „ye” (czyt. „je”), „wu”, „wi” i „we” (dwie ostanie mają zapis w systemie *katakana*, jednak znaki te nie są już używane) – stąd puste pola w tabelach 1. i 2.

Hiragana służy głównie do zapisywania słów japońskiego pochodzenia, często w towarzystwie znaków *kanji*. Systemem *katakana* natomiast zapisuje się zapożyczenia z obcych języków i niektóre onomatopeje.

Przykładowe słowa zapisywane w systemie *hiragana* podane są w tabeli 3.

Hiragana	Wymowa	Znaczenie
ありがとう	arigatō	dziękuję
あの	ano	ten/ta/to

Tabela 3. Przykładowe słowa w systemie *hiragana*

Przykładowe słowa zapisywane w systemie *katakana* podane są w tabeli 4.

Katakana	Wymowa	Znaczenie
パソコン	pasokon	komputer
ジュース	jūsu	sok

Tabela 4. Przykładowe słowa w systemie *katakana*

Poza sylabami znajdującymi się w tabelach 1. i 2. występują też inne, które powstają poprzez połączenie dużego znaku np. き „ki" oraz mniejszego や „ya" tworząc napis きゃ „kya". Analogicznie tworzy się łączenia w systemie *katakana*: łącząc znak キ ze znakiem ヤ otrzymuje się napis キャ.

Z kolei *kanji* to znaki chińskie, którymi zapisuje się większość japońskich rzeczowników, przymiotników i czasowników, w wielu przypadkach w połączeniu ze znakami *hiragana*. Takich znaków jest około 50 tysięcy, jednak każdy dorosły Japończyk powinien znać co najmniej 2136 podstawowych, zwanych *jōyō kanji* (znaki powszechnego użycia wybrane przez Japońskie Ministerstwo Edukacji w 2010 roku).

Kanji/hiragana	Wymowa	Znaczenie
日本	nihon	Japonia
読む	yomu	czytać
便利	benri	wygodny
面白い	omoshiroi	zabawny

Tabela 5. Przykłady słów pisanych znakami *kanji*

Znaki *kanji*, których przykłady można znaleźć w tabeli 5., mają dwa typy wymowy: rdzennie japońską i sinojapońską, a każdy z nich może mieć po kilka czytań obu rodzajów (obrazuje to tabela 6.), które często zależą od tego, z jakimi innymi znakami się łączą.

Znak *kanji*	Czytanie japońskie	Czytanie sinojapońskie
私	watashi/watakushi	shi
愛	ito/kana/me/o/mu/mana	ai

Tabela 6. Przykłady znaków *kanji* i ich czytania

Każde słowo zapisywane z użyciem *kanji*, można zapisywać samymi znakami *hiragana*, lecz nie zawsze będzie to łatwe do zrozumienia, gdyż wiele słów o różnym znaczeniu wymawia się w ten sam sposób.

Zapis z użyciem znaków *kanji* i *hiragana*	Zapis z użyciem znaków *hiragana*	Wymowa
庭には二羽鶏がいる	にわにはにわにわとりがいる	niwa ni wa niwa niwatori ga iru

Tabela 7. Przykład zdania w języku japońskim zapisanego na dwa sposoby

W tabeli 7. pokazano przykład zdania w języku japońskim zapisanego na dwa sposoby. Zapis w systemie *hiragana* nie informuje jednoznacznie o podziale zdania na wyrazy oraz o znaczeniu danego słowa, gdyż kilka fragmentów zapisano takim samym układem znaków.

Poza wspomnianymi trzema systemami pisma, w Japonii występuje jeszcze jeden, zwany *rōmaji*, czyli wykorzystujący alfabet łaciński, jednak w życiu codziennym stosowany jest tylko sporadycznie i głównie jako ułatwienie dla obcokrajowców na przykład w odczytywaniu nazw stacji metra.

2.2. Trudności w automatyzacji tłumaczenia

Tłumaczenie automatyczne języka japońskiego stanowi duże wyzwanie ze względu na wiele, mniej lub bardziej istotnych, czynników. Pierwszym i najbardziej oczywistym jest system pisma, a właściwie połączenie czterech różnych systemów, których zastosowanie często zależy od samego autora tekstu. Ponadto, mogą występować problemy z kodowaniem, gdyż znaki japońskie kodowane są na wiele standardów, między innymi na UTF-16, UTF-8 lub ISO-2022-JP. Istotnym problemem w automatyzacji tłumaczenia japońskiego jest również brak spacji między wyrazami. Z tego powodu wymagane są dedykowane programy do tokenizacji tekstów, które pozwalają na rozdzielenie zdań

na mniejsze części, jak najdokładniej pokrywające się z rzeczywiście występującymi tam słowami. Nie jest to łatwy proces, ponieważ niektóre zdania można podzielić na kilka sposobów nadających różne znaczenie tekstowi.

Kolejną przeciwnością w automatycznym tłumaczeniu języka japońskiego są różnice w zależności od płci, wieku i statusu społecznego osoby, która się wypowiada lub pisze dany tekst. Wypowiedź osoby o niższym statusie wobec osoby wyżej w hierarchii może mieć zupełnie inną postać niż w odwrotnym przypadku. Co więcej, w Japonii występuje wiele dialektów, które różnią się między sobą w bardzo dużym stopniu.

Jeżeli chodzi o tłumaczenie języka japońskiego na język polski, to taka para tworzy kolejne trudności. Po pierwsze, języki te różnią się składnią, jak chociażby umiejscowieniem czasownika, co obrazuje tabela 8. W języku japońskim czasownik znajduje się na końcu zdania.

Zdanie japońskie	Zdanie polskie
映画館に行きました (czyt. *eigakan ni ikimashita*) (*eigakan* – kino, *ni* – do, *ikimashita* – poszłam)	Poszłam do kina

Tabela 8. Przykład zdania japońskiego i jego polski odpowiednik

Co więcej, w przeciwieństwie do języka polskiego, w języku japońskim czasowniki nie zmieniają formy zależnie od podmiotu. W dodatku podmiot często nie jest określony dosłownie i wynika jedynie z kontekstu. Jednak automatyczny system tłumaczący nie działa tak jak ludzki mózg, więc wybór odpowiedniej odmiany czasownika, która powinna znaleźć się w zdaniu docelowym, nie zawsze będzie poprawny. Trudność tę obrazuje przykład z tabeli 8. Zdanie *eigakan ni ikimashita*, można przetłumaczyć na wiele sposobów, między innymi: „poszedł do kina”, „poszli do kina”, czy „poszliśmy do kina”.

3. Tłumaczenie automatyczne

Tłumaczenie automatyczne to zautomatyzowany proces odczytywania zdania w jednym naturalnym języku – źródłowym (ang. *source*) i przekształcania go w zdanie o takim samym znaczeniu w języku docelowym (ang. *target*). Takie zadanie przysparza wiele problemów wynikających między innymi z różnic w gramatyce języków.

Próby tworzenia zautomatyzowanych translatorów sięgają ubiegłego wieku. Pierwsze maszyny służące do tego celu pojawiły w I połowie XX wieku. Z biegiem lat powstawały coraz to nowsze i lepiej działające rozwiązania. Do lat osiemdziesiątych dominowały systemy wykorzystujące różnego rodzaju reguły, na przykład morfologiczne, czy składniowe. Zostały one jednak wyparte przez nowe rozwiązanie polegające na korzystaniu z korpusów równoległych, czyli dużych zbiorów tekstów w dwóch różnych językach, gdzie każdy fragment w języku źródłowym odpowiada konkretnemu fragmentowi w języku docelowym.

Przykład korpusu równoległego dla pary języków, japońskiego i polskiego, obrazuje tabela 9.

Zdanie w języku źródłowym - japońskim	Zdanie w języku docelowym - polskim
おなまえはなんですか。	Jak masz na imię?
わたしはポーランドじんです。	Jestem Polką.
公園を散歩します。	Spaceruję po parku.

Tabela 9. Przykład korpusu równoległego

Aktualnie w zagadnieniu tłumaczenia automatycznego można wyróżnić dwie główne jego metody: tłumaczenie statystyczne oraz neuronowe. Oba wspomniane typy korzystają z korpusów równoległych.

3.1. Tłumaczenie statystyczne

Metody statystyczne zdominowały dziedzinę tłumaczenia automatycznego pod koniec dwudziestego wieku (Junczys-Dowmunt, 2008). Jeszcze do niedawna można było uznać je za najbardziej wydajne i wymagające stosunkowo niewiele czasu na stworzenie sensownego systemu do tłumaczenia.

Statystyczne metody tłumaczenia automatycznego niewiele mają wspólnego z procesem, któremu poddawany jest tekst, gdy tłumaczy go człowiek. System tłumaczący nie bierze pod uwagę takich elementów jak chociażby składnia i nie analizuje jej. Tłumaczenie automatyczne nastawione jest na uzyskanie wyniku, czyli zdania które będzie w jak największym stopniu odpowiadało zdaniu wejściowemu. Tak więc statystyczne tłumaczenie w dużej mierze bazuje na prawdopodobieństwie:

$$e_{best} = argmax_{pl}\, P(Pl = pl | Ja = ja)$$

Wzór 1.

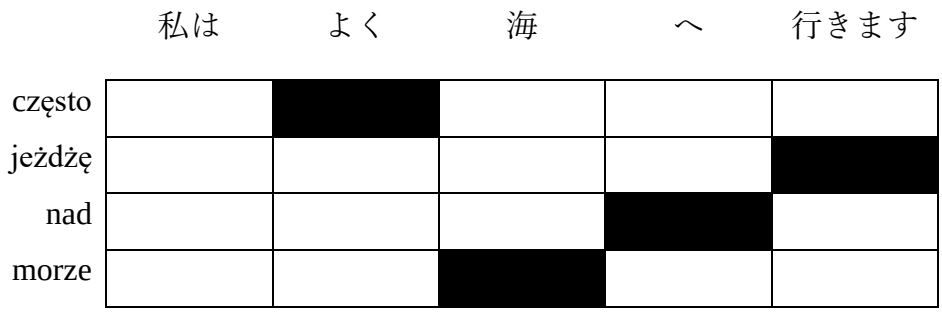
We wzorze 1. *Pl* i *Ja* oznaczają zmienne losowe, które przebiegają po wszystkich możliwych zdaniach polskich *pl* i japońskich *ja*. Zdanie e_{best} maksymalizuje powyższą funkcję dla danego zdania japońskiego *ja* i odpowiada jego najbardziej prawdopodobnemu tłumaczeniu (Junczys-Dowmunt, 2008). W celu wyliczenia prawdopodobieństwa stosuje się, wspomniane we wstępie do rozdziału 3., korpusy równoległe.

W tłumaczeniu statystycznym można wydzielić dwa najważniejsze elementy, na którym ono bazuje – model tłumaczenia oraz model języka. Pierwszy z nich wymaga istnienia tekstów równoległych, a drugi jedynie zbioru zdań w języku docelowym.

3.1.1. Model tłumaczenia

Modele tłumaczenia dzielą się na dwa główne typy: „oparte na wyrazach” (ang. *word-based*) oraz „oparte na frazach” (ang. *phrase-based*).

Początkowe modele statystyczne uznawały słowa jako swego rodzaju jednostki, na których można wykonywać różne działania, jak tłumaczenie, zmiana kolejności, dodawanie, usuwanie. Budowanie modelu tłumaczenia polegało na dopasowywaniu słowa z języka źródłowego do słowa z języka docelowego za pomocą funkcji dopasowującej (ang. *alignment function*) (Koehn, 2010). Takie właśnie podejście określa się „opartym na wyrazach”.



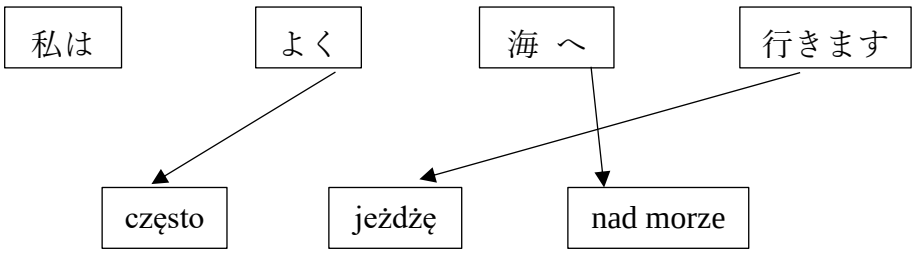
Rysunek 1. Przykład dopasowania w metodzie „opartej na wyrazach”

Na rysunku 1. zobrazowano dopasowanie słów zdania w języku japońskim do słów zdania w języku polskim (z założeniem, że zdanie japońskie zostało uprzednio poddane tokenizacji). Każdemu tokenowi zdania japońskiego przyporządkowano wyraz ze zdania polskiego o takim samym znaczeniu. W przypadku tokenu 私は nie udało się znaleźć odpowiednika polskiego, gdyż oznacza on „ja”, a takie słowo nie wystąpiło w zdaniu w języku polskim.

Przykład z rysunku 1. przedstawia idealne dopasowanie, a w rzeczywistości każdy token ze zdania japońskiego mógłby zostać dopasowany do każdego z wyrazów polskiego zdania. Jednak tak się nie dzieje, gdyż jako dopasowanie słowa w języku źródłowym przyjmuje się słowo w języku docelowym o najwyższym prawdopodobieństwie tłumaczenia.

Opisane wyżej modele „oparte na wyrazach” zostały jednak wyparte przez podejście „oparte na frazach”. W tej metodzie również wylicza się prawdopodobieństwo tłumaczenia, lecz zdanie źródłowe jest najczęściej dzielone na sekwencje słów, które są dopasowywane do sekwencji słów zdania wyjściowego. Fraza może składać się z jednego lub kilku słów, a w niektórych przypadkach nawet z całego zdania.

Przykład dla pary japońsko-polskiej ukazuje rysunek 2. Wykorzystano w nim te same zdania, które użyto w przykładzie z rysunku 1. W przypadku metody „opartej na frazach”, znaki 海 i へ zostały połączone w jedną frazę i przyporządkowano jej dwa, również połączone we frazę, polskie słowa „nad” oraz „morze”. Pozostałe frazy utworzone zostały na bazie pojedynczych wyrazów i tokenów.



Rysunek 2. Zobrazowanie działania funkcji dopasowującej w metodzie „opartej na frazach”

Istotną zaletą metody „opartej na frazach” (w przeciwieństwie do metody „opartej na wyrazach”) jest to, że umożliwia ona uniknięcie błędów w przypadku, gdy tłumaczenie frazy nie jest złożeniem tłumaczenia jej składowych.

W latach dziewięćdziesiątych XX wieku badacze amerykańskiej firmy IBM (International Business Machines Corporation) opracowali pięć statystycznych modeli tłumaczenia zwanych **modelami IBM**. Nie są to oczywiście jedyne tego typu modele stosowane w statystycznym tłumaczeniu automatycznym, jednak zyskały one największą popularność i wykorzystywane są na szeroką skalę. Wszystkie modle IBM działają niezależnie od siebie i uwzględniają różne aspekty tłumaczenia oraz zależności między wyrazami. Model 1 jest najprostszym modelem. Generuje on dopasowania słów języka źródłowego do słów języka docelowego z informacją o prawdopodobieństwie danego dopasowania. Model 2 dodaje informację o szyku wyrazów. Model 3 określa, ile wyrazów języka docelowego odpowiada jednemu wyrazowi języka źródłowego. Model 4 decyduje o położeniu tłumaczenia danego słowa na podstawie położenia wcześniejszych słów z tego samego źródła. Natomiast Model 5 jest odpowiedzialny za to, aby słowa docelowe nie zostały umieszczone w miejscu zajętym już przez inne słowo. Nie jest konieczne stosowanie wszystkich pięciu modeli, jednakże razem budują one całą bazę informacji potrzebnych do uzyskania dobrego tłumaczenia.

3.1.2. Model języka

Model języka powstaje na bazie korpusu języka docelowego. Celem zastosowania modelu języka jest nadanie płynności i poprawności zdaniu wyjściowemu. Formalnie, model języka jest funkcją (P_{LM}), która na wejściu dostaje ciąg wyrazów w danym języku i zwraca prawdopodobieństwo tego, że właśnie taki ciąg wyrazów zostałby użyty przez człowieka. Na przykład w przypadku języka polskiego większe prawdopodobieństwo ma wyrażenie

„lubię czytać książki” niż „czytać książki lubię” i to właśnie pierwszemu z nich dobry model języka powinien przypisać wyższe prawdopodobieństwo:

$$P_{LM}(lubię\ czytać\ książki) > P_{LM}(czytać\ książki\ lubię)$$

Preferencja modelu języka oparta na wyższym prawdopodobieństwie wspiera system tłumaczenia statystycznego w ustalaniu poprawnej kolejności słów w zdaniach oraz w wyborze odpowiedniego tłumaczenia danego słowa w przypadku, gdy ma ono wiele różnych tłumaczeń zależnych od kontekstu.

Najczęściej stosowaną metodą tworzenia modeli języka jest modelowanie z użyciem n-gramów. **N-gramowe modele języka** opierają się na statystycznym prawdopodobieństwie wystąpienia danego słowa po konkretnym ciągu słów. W procesie modelowania języka oblicza się prawdopodobieństwo ciągu $W = w_1, w_2, \ldots, w_n$. Jak nie trudno się domyślić, zbyt długie sekwencje słów rzadko będą się powielać nawet w wielkich korpusach, dlatego też obliczanie prawdopodobieństwa *P(W)* trzeba podzielić na mniejsze kroki. Pozwalają one na zebranie odpowiedniej ilości danych statystycznych oraz oszacowanie rozkładów prawdopodobieństwa. W n-gramowym modelowaniu języka, proces przewidywania całej sekwencji słów zamienia się na przewidywanie wystąpienia jednego słowa po ciągu słów (Koehn, 2010).

W pierwszym etapie tworzenia n-gramowego modelu należy uzyskać rozkład prawdopodobieństwa korzystając z reguły łańcuchowej (Koehn, 2010):

$$P(w_1, w_2, w_3, \ldots, w_n) = P(w_1)p(w_2|w_1)p(w_3|w_1, w_2)\ldots P(w_n|w_1, \ldots, w_{n-1})$$

Wzór 2.

Poniżej znajduje się przykład zastosowania wzoru 2. dla języka polskiego, gdzie *n=3*:

$$P(lubię, czytać, książki) = P(lubię)P(czytać|lubię)P(książki|lubię, czytać)$$

Prawdopodobieństwo całego modelu języka uzyskuje się na podstawie prawdopodobieństw poszczególnych słów, biorąc pod uwagę historię poprzedzających je wyrazów (Koehn, 2010). Dla przewidywanego słowa *n* wszystkie wyrazy znajdujące się przed nim, czyli od *1* do *n−1,* stanowią jego historię. Ze względów obliczeniowych, długość historii ogranicza się do *m* wyrazów:

$$P(w_n|w_1, w_2, \ldots, w_{n-1}) \approx P(w_n|w_{n-m}, \ldots, w_{n-2}, w_{n-1})$$

Wzór 3.

Poniżej znajduje się przykład zastosowania wzoru 3. dla języka polskiego, gdzie *n=4* i *m=3* (całe zdanie: *bardzo lubię czytać książki*):

$$P(książki|bardzo, lubię, czytać) \approx P(książki|lubię, czytać)$$

W przykładzie pomijamy pierwsze słowo, czyli „bardzo", ponieważ historia ma długość 3, a całe zdanie 4.

Jeśli chodzi o historię słów, to często stosowane są n-gramy długości 3, czyli **trigramy**. W przypadku trigramów, w celu przewidzenia trzeciego słowa wykorzystywana jest historia dwóch wyrazów poprzedzających. Inne częste n-gramy to **unigramy** (brak historii) lub **bigramy** (historia długości 1). Do obliczeń w modelach n-gramowych stosuje się **metodę największej wiarygodności** (ang. *maximum likelihood estimation*).
Dla n-gramów wylicza się, jak często w korpusie treningowym sekwencja słów $w_1, w_2, \ldots, w_{n-1}$ poprzedza wyraz w_n i dzieli się tę liczbę przez sumę wystąpień innych słów po sekwencji $w_1, w_2, \ldots, w_{n-1}$ (Koehn, 2010). Wzór 4. przedstawia wzór metody największej wiarygodności dla trigramów:

$$P(w_3|w_1, w_2) = \frac{count(w_1, w_2, w_3)}{\sum_w count(w_1, w_2, w)}$$

Wzór 4.

3.1.3. Podstawowe równanie tłumaczenia statystycznego

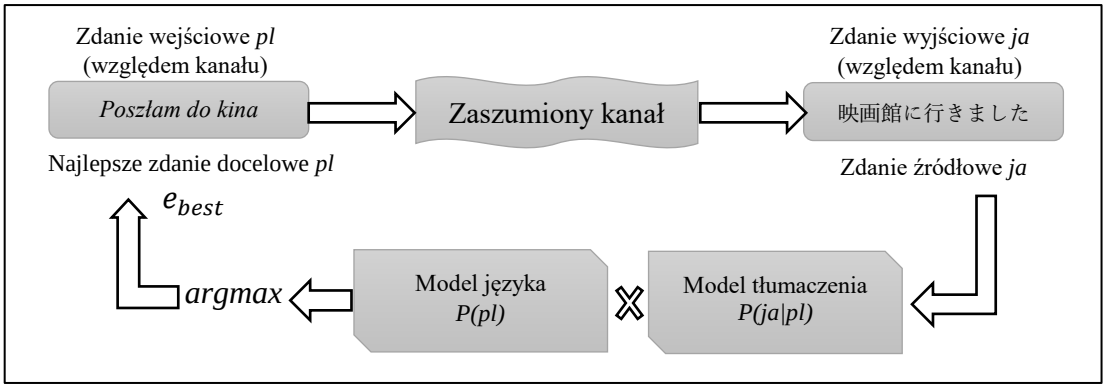
Połączenie modelu języka z modelem tłumaczenia tworzy podstawowe równanie tłumaczenia statystycznego (Junczys-Dowmunt, 2008):

$$e_{best} = argmax_{pl} P(ja|pl) P_{LM}(pl)$$

Wzór 5.

We wzorze 5. *pl* i *ja* oznaczają kolejno zdania polskie i japońskie. Zdanie w języku polskim e_{best} maksymalizuje powyższą funkcję dla danego zdania japońskiego *ja* bazując na prawdopodobieństwie uzyskanym dzięki modelom tłumaczenia ($P(ja|pl)$) i języka ($P_{LM}(pl)$).

Takie połączenie modeli oparte na wzorze 5. tworzy tak zwany **model kanału zaszumionego** (ang. *noisy-channel model*). Schemat jego działania dla tłumaczenia statystycznego przedstawiono na rysunku 3.



Rysunek 3. Model zaszumionego kanału; na podst. Jurafsky i Martin, 2000

W modelu zaszumionego kanału zdaniem wejściowym względem kanału jest zdanie w języku polskim, który dla systemu tłumaczącego jest językiem docelowym. Natomiast zdanie wyjściowe z kanału to zdanie w języku japońskim i stanowi ono także zdanie źródłowe dla systemu tłumaczącego.

Połączenie modelu tłumaczenia z modelem języka pozwala uzyskać zarówno wierne, jak i płynne tłumaczenie tekstów z języka źródłowego na język docelowy.

3.2. Tłumaczenie neuronowe

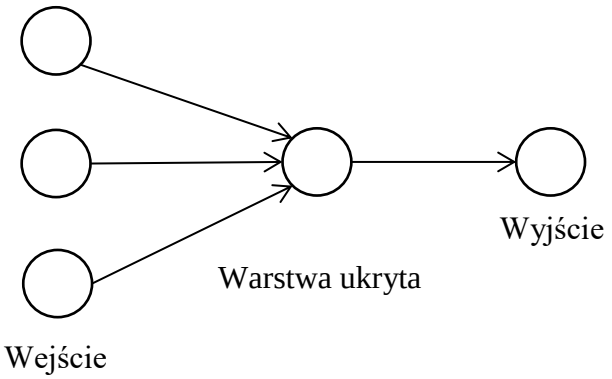
Neuronowe tłumaczenie automatyczne w ostatnich latach stanowi najważniejszą i w dodatku bardzo szybko rozwijającą się metodę tłumaczenia automatycznego (Koehn, 2017). Metoda ta polega na zastosowaniu sztucznych sieci neuronowych w procesie tworzenia systemu tłumaczącego.

3.2.1. Sztuczne sieci neuronowe

Sztuczne sieci neuronowe stanowią jedną z technik uczenia maszynowego. Inspiracją do ich stworzenia była idea działania układu nerwowego istot żywych. W przeciwieństwie do innych metod uczenia maszynowego, wyróżnia je zastosowanie dodatkowych warstw, tak zwanych warstw ukrytych, pomiędzy wejściem a wyjściem.

Najprostsza sztuczna sieć neuronowa składa się tylko z warstwy wejściowej (ang. *input layer*), jednej warstwy ukrytej (ang. *hidden layer*) i warstwy wyjściowej (ang. *output layer*). Rysunek 4. przedstawia uproszczoną wersję sieci neuronowej. Pierwsze trzy węzły

stanowią warstwę wejściową, kolejny węzeł to warstwa ukryta, a ostatni z nich warstwa wyjściowa.



Rysunek 4. Prosta sztuczna sieć neuronowa

Z matematycznego punktu widzenia, prosta sieć neuronowa z jedną warstwą ukrytą składa się z następujących elementów:

- wektora węzłów wejściowych z wartościami $\vec{x} = (x_1, x_2, x_3, \ldots, x_n)^T$
- wektora węzłów ukrytych z wartościami $\vec{h} = (h_1, h_2, h_3, \ldots, h_m)^T$
- wektora węzłów wyjściowych z wartościami $\vec{y} = (y_1, y_2, y_3, \ldots, y_l)^T$
- macierzy wag $W = \{w_{ij}\}$ łączących węzły wejściowe z ukrytymi
- macierzy wag $U = \{u_{ij}\}$ łączących węzły ukryte z wyjściowymi

Sieć neuronowa jest przetwarzana w dwóch krokach:

- najpierw kombinacja liniowa węzłów wejściowych jest przetwarzana w celu uzyskania wartości ukrytych węzłów zgodnie ze wzorem 6.:

$$h_j = \sum_i x_i w_{ji}$$

Wzór 6.

- następnie kombinacja liniowa powstałych wcześniej węzłów ukrytych jest przetwarzana w celu otrzymania wartości każdego węzła wyjściowego zgodnie ze wzorem 7.:

$$y_k = \sum_j h_j u_{kj}$$

Wzór 7.

Aby powiązać ze sobą warstwę wejściową z wyjściową należy dodatkowo zastosować funkcję aktywacji (ang. *activation function*). Taka funkcja służy do obliczania wyjścia neuronów w danej sieci (Koehn, 2017).

Gdy pewna wartość liczbowa dociera do węzła, zostaje ona podstawiona do jednej z funkcji aktywacji, np.:

- funkcji logistycznej (ang. *logistic function*) (wzór 8.)

$$f(x) = \frac{1}{1 + e^{-x}}$$

Wzór 8.

- funkcji tangens hiperboliczny (ang. *hyperbolic tangent*) (wzór 9.)

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Wzór 9.

- funkcji ReLU (ang. *rectified linear unit*) (wzór 10.)

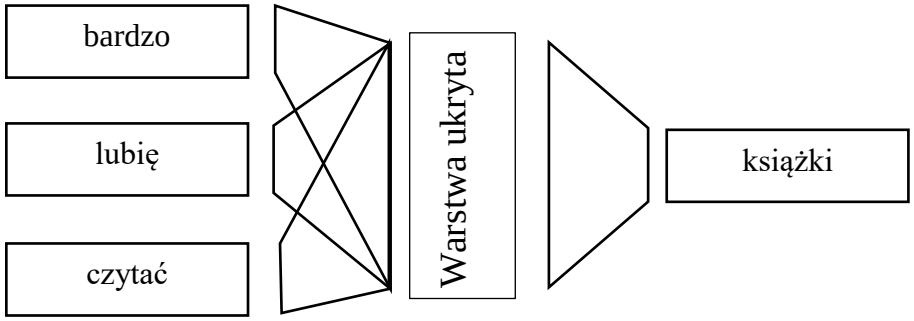
$$f(x) = max(0, x)$$

Wzór 10.

3.2.2. Neuronowe modele języka

Na początku sieci neuronowe zostały zastosowane w tłumaczeniu automatycznym w celu ulepszenia modeli języka. Nazwano je neuronowymi modelami języka. **Neuronowe modele języka** (NLM) to klasa modeli językowych, które zaprojektowano szczególnie w celu rozwiązania problemu zwanego przekleństwem wielowymiarowości (ang. *curse of dimensionality*) w modelowaniu sekwencji słów w języku naturalnym (Bengio i in., 2003). Przekleństwo wielowymiarowości jest związane z tym, że istnieje spore prawdopodobieństwo, iż dana sekwencja słów, na której model będzie testowany, okaże się inna niż wszystkie sekwencje słów, które pojawiły się podczas trenowania modelu języka. W przypadku neuronowych modeli zastosowano rozproszoną reprezentację słów. W przeciwieństwie do modeli n-gramowych stosowanych w tłumaczeniu statystycznym, modele neuronowe traktują w podobny sposób słowa mające wspólne cechy.

Na rysunku 5. przedstawiono prosty schemat 4-gramowego neuronowego modelu języka składającego się ze słów wejściowych, warstwy ukrytej oraz przewidywanego słowa:



Rysunek 5. Prosty schemat neuronowego modelu języka, na podst. Koehn, 2017

W sieci neuronowej słowa reprezentowane są z wykorzystaniem wektorów wielowymiarowych. Każde słowo zajmuje jeden wymiar w słowniku i ma przypisaną wartość 1 w miejscu, które odpowiada jego wymiarowi. Tego typu wektory nazywane są *one-hot vector*. Na przykład słowa *panda* i *koala* mogą mieć następującą reprezentację wektorową:

$$panda = (0,0,0,1,0,0,0,\dots)^T$$

$$koala = (0,0,0,0,0,0,1,\dots)^T$$

Takie wektory zazwyczaj są bardzo obszerne w związku z ogromną ilością słów w korpusach.

Kolejną warstwą w sieci neuronowej znajdującą się między wejściem, a warstwą ukrytą jest warstwa *word embeddings.* W tej warstwie słowa zostają rzutowane na przestrzeń o mniejszej liczbie wymiarów, niż łączna liczba słów.

Word embeddings to specjalna reprezentacja słów o wspólnych cechach. Nazwa ta w języku polskim tłumaczona jest czasem jako „zanurzenia słów".

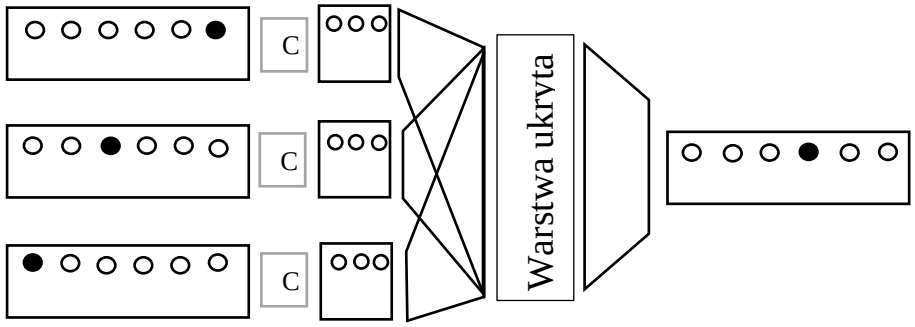
Przykładowo, jeśli słowa „pandy" i „koale" wystąpiły w podobnym kontekście, np.:

Ludzie lubią oglądać pandy w zoo
Ludzie lubią oglądać koale w zoo

i zostały zmapowane do reprezentacji o wspólnych atrybutach, wtedy zdania ze słowem „pandy" mogą dostarczać informacje o kolejnych słowach podczas przewidywania zdań ze słowem „koale". Dzięki temu, że istnieje wiele takich atrybutów, które można

współdzielić, jest dużo możliwości stosowania generalizacji dla grup słów. Dlatego też model można trenować na mniejszej ilości danych uczących, niż ma to miejsce w przypadku modeli statystycznych.

Rysunek 6. przedstawia pełną architekturę neuronowego modelu języka. Na wejściu znajdują się słowa zakodowane w postaci *one-hot vector*. Są one przetwarzane z użyciem macierzy C służącej do tworzenia wektorów *word embeddings*. Kolejną warstwę stanowi warstwa ukryta, a po niej otrzymuje się słowo wyjściowe (także w formie wektora).

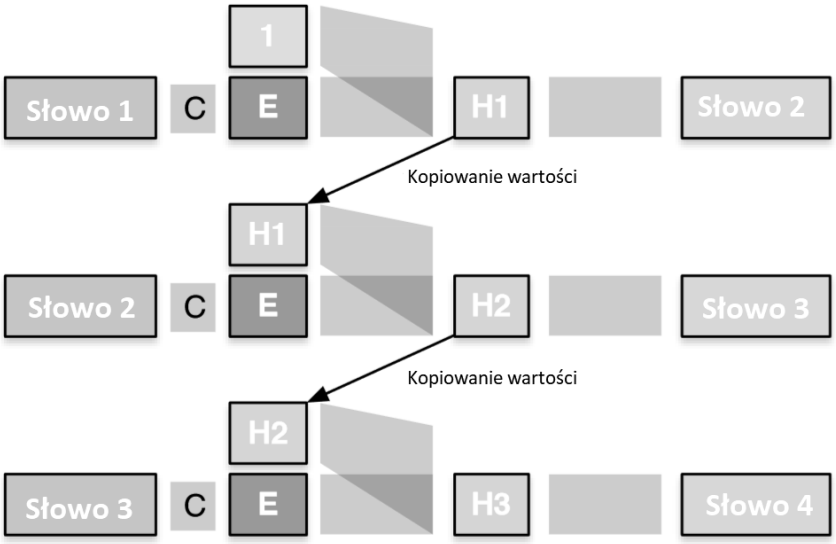


Rysunek 6. Pełna architektura neuronowego modelu języka, na podst. Koehn, 2017

Parametry modelu języka trenowane są poprzez przetwarzanie wszystkich n-gramów w korpusie wejściowym. W przypadku każdego z n-gramów, sieć neuronowa zasilana jest słowami danej sekwencji, a każde słowo wyjściowe sieci dopasowywane jest do wektora *one-hot vector* odpowiadającego poprawnemu przewidywanemu słowu. Wagi sieci neuronowej aktualizowane są za pomocą algorytmu wstecznej propagacji w czasie (ang. *back-propagation through time*). Taka procedura „odwija" sieć neuronową w głąb do pewnego określonego stopnia, na przykład cofa się o trzy przewidywania słów (Koehn, 2017).

Neuronowe modele języka nie muszą jednak działać tylko w jedną stronę, jak to dzieje się w powyżej opisanym modelu. Sieci neuronowe typu *feed-forward* (działające w przód) nie mają możliwości zapamiętania przetworzonych wcześniej informacji i wykorzystania ich ponownie. Jako rozwiązanie tego problemu można zastosować **rekurencyjne sieci neuronowe** (ang. *recurrent neural networks*, RNN). **Rekurencyjne neuronowe modele języka** (ang. *recurrent neural language models*) cechują się tym, że do przewidywania słowa w_n stosuje się dodatkowo warstwy ukryte, które powstały podczas przewidywania słowa w_{n-1} (Koehn, 2017).

Rysunek 7. prezentuje schemat rekurencyjnego neuronowego modelu języka. Dane wejściowe w tym modelu to pierwsze słowo (*Słowo 1*) oraz drugi zestaw neuronów, który w tym wypadku oznacza początek zdania (*1*). Wektor *word embeddings* pierwszego słowa wraz z neuronami początku zdania mapowane są do warstwy ukrytej H_1, która potem zostaje wykorzystana do przewidywania słowa wyjściowego (*Słowo 2*). Następnie te same etapy przechodzi *Słowo 2* wraz z powstałą wcześniej warstwą H_1.



Rysunek 7. Schemat rekurencyjnego neuronowego modelu języka, na podst. Koehn, 2017

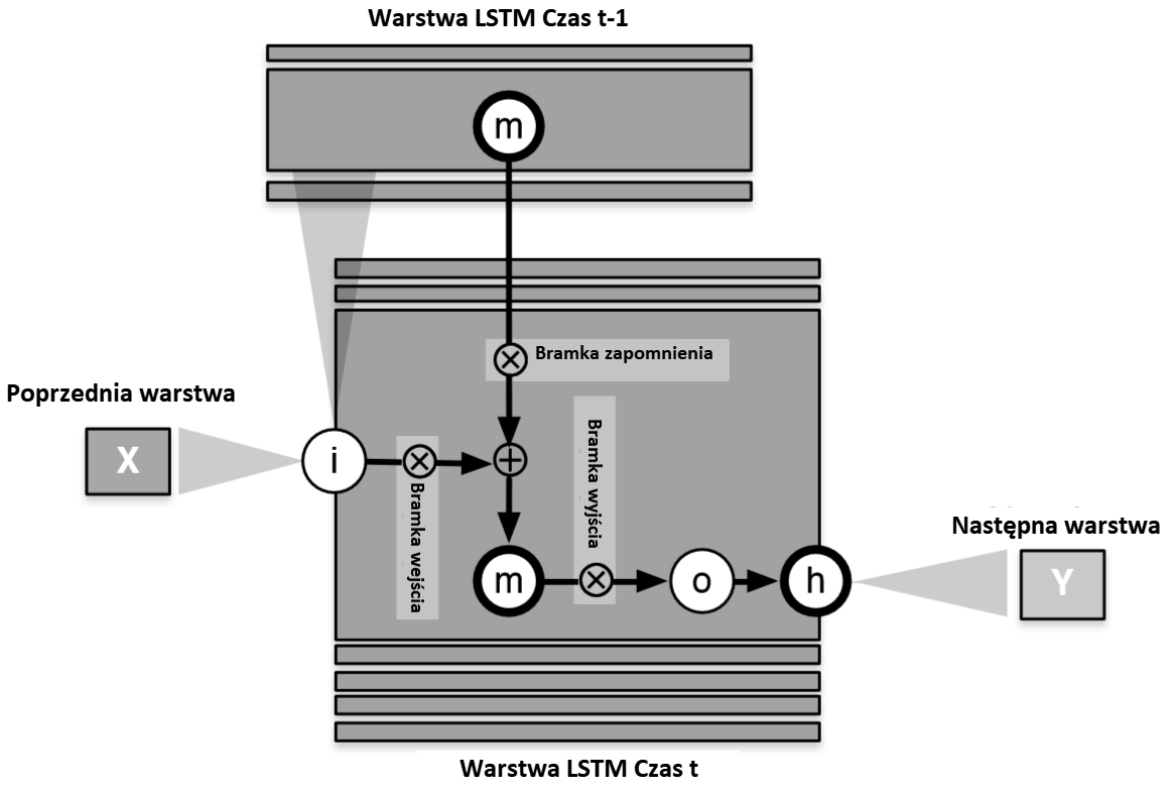
W rekurencyjnych neuronowych modelach języka stosuje się również wsteczną propagację do aktualizacji wag sieci neuronowej (Koehn, 2017).

Innymi często stosowanymi neuronowymi modelami języka są **modele LSTM** (ang. *long short-term memory models*). Od modeli rekurencyjnych różnią się głównie tym, że w sieci LSTM podstawowa konstrukcja, czyli komórka (ang. *cell*), zawiera wyodrębniony stan pamięci. Taka komórka przypomina komórkę pamięci w komputerach, jednak z tą różnicą, że przechowuje liczby rzeczywiste, a nie bity. Komórka LSTM pozwala na operacje odczytu, zapisu oraz resetu, które regulują specjalne parametry liczbowe zwane bramkami (ang. *gates*). Wyróżnia się następujące bramki:

- wejścia (ang. *input gate*) – służące do regulacji stopnia zmiany stanu pamięci poprzez nowe dane wejściowe,
- zapomnienia (ang. *forget gate*) – służące do regulacji tego, jaka część wcześniejszego stanu pamięci zostanie zachowana (lub zapomniana),
- wyjścia (ang. *output gate*) – służące do regulacji stopnia w jakim stan pamięci przekazywany będzie do kolejnej warstwy.

Modele LSTM trenowane są tak samo, jak w przypadku rekurencyjnych neuronowych modeli języka z zastosowaniem propagacji wstecznej. Jednak wszystkie operacje w komórkach LSTM są bardziej złożone, niż te zachodzące w sieciach rekurencyjnych (Koehn, 2017).

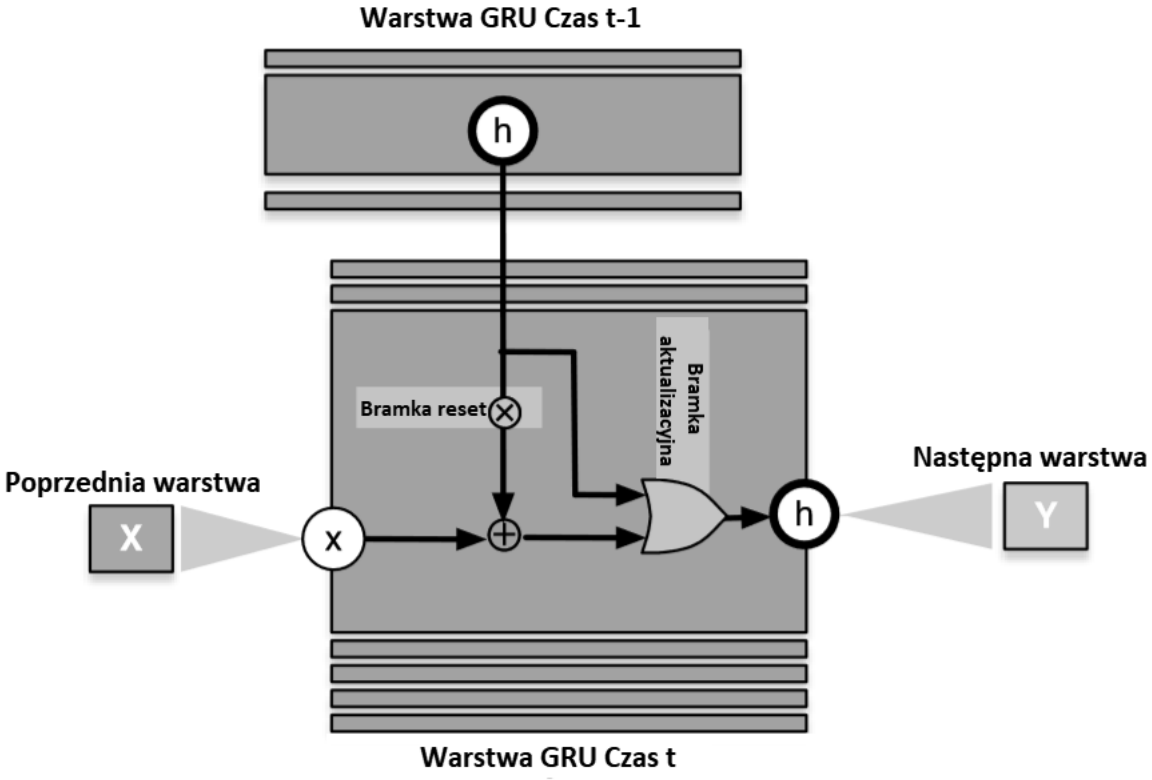
Na rysunku 8. przedstawiony jest schemat działania komórki LSTM. Komórka otrzymuje poprzednią warstwę (*x*) na wejście oraz wartości z warstwy ukrytej z poprzedniego kroku czasu (*t-1*). Stan pamięci *m* aktualizowany jest na podstawie stanu wejściowego *i* oraz wartości stanu pamięci z poprzedniego kroku czasu. Informacje w komórce przepływają przez bramki aż do wartości wyjściowej (*o*).



Rysunek 8. Schemat działania komórki LSTM, na podst. Koehn, 2017

W związku z tym że komórki LSTM dodają bardzo wiele różnego rodzaju parametrów, między innymi macierzy wag dla bramek, rośnie czas trenowania modeli. Jako uproszczoną wersję komórek LSTM zaproponowano **bramkowane rekurencyjne jednostki** (ang. *gated recurrent units*, GRU). Od komórek LSTM w szczególności odróżnia je to, że pojedyncza jednostka bramkowa równocześnie kontroluje czynnik zapomnienia oraz aktualizację stanu jednostki. W komórkach GRU znajdują się tylko dwie bramki: aktualizacyjna (ang. *update*) i reset (ang. *reset*). Parametry tych bramek przewidywane są na podstawie danych wejściowych oraz poprzedniego stanu (Koehn,

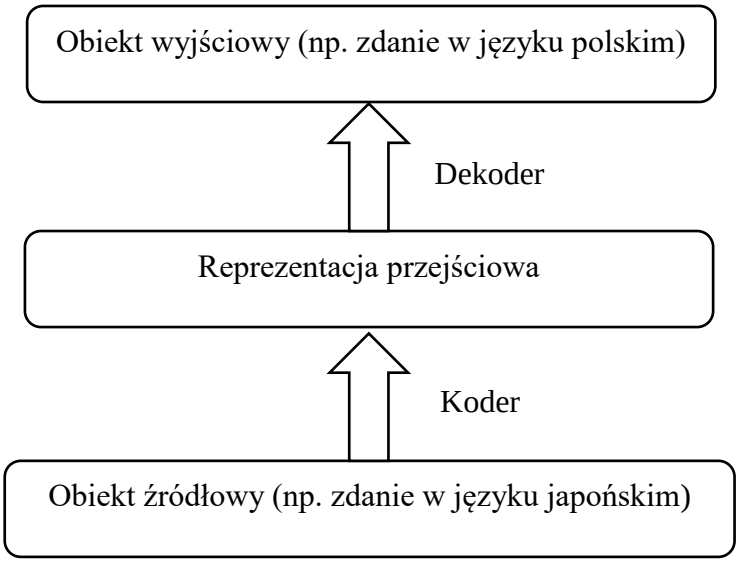
2017). Na rysunku 9. przedstawiony jest schemat działania komórki GRU. Proces wygląda bardzo podobnie jak w przypadku schematu komórki LSTM, lecz jest nieco uproszczony, gdyż występują tylko dwie bramki, a nie trzy.



Rysunek 9. Schemat działania komórki GRU, na podst. Koehn, 2017

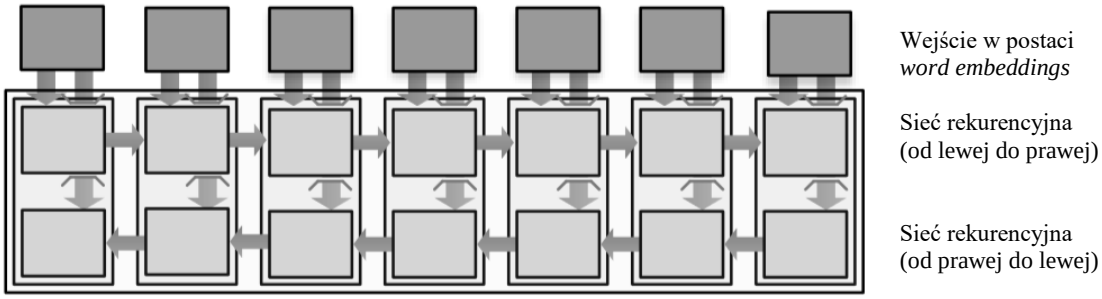
3.2.3. Neuronowy model tłumaczenia

Model neuronowego tłumaczenia automatycznego z podejściem zwanym **koder-dekoder** (ang. *encoder-decoder*) polega na zastosowaniu co najmniej dwóch sieci neuronowych. Zarówno koder, jak i dekoder to najczęściej rekurencyjna sieć neuronowa, czyli taka sieć, która korzysta na wejściu z warstw ukrytych, które powstają w czasie jej działania. Koder jest modelem, który koduje korpus języka źródłowego na strukturę danych np. wektory. Z kolei te dane przekazywane są na wejście dekodera, który odczytuje je i generuje na ich podstawie zdanie w języku docelowym. Rysunek 10. przedstawia prosty schemat podejścia koder-dekoder.



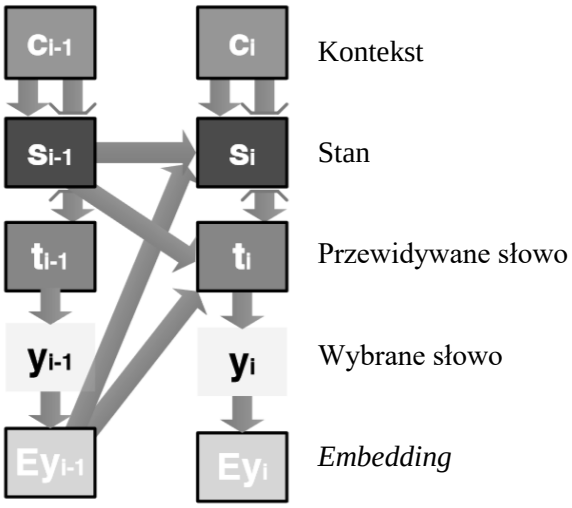
Rysunek 10. Schemat struktury koder-dekoder, na podst. Goodfellow, Bengio i Courville, 2016

Zadaniem kodera jest przekonwertowanie ciągu wejściowego na pewną reprezentację przejściową. Wyrazy ciągu przetwarzane są z pomocą neuronowych sieci rekurencyjnych. Powstają w ten sposób ukryte stany kodujące każde słowo z ich lewym kontekstem, czyli wszystkimi poprzedzającymi słowami. Buduje się także drugą sieć rekurencyjną działającą na odwrót – od końca ciągu do jego początku. Rysunek 11. przedstawia schemat kodera składającego się z dwóch sieci rekurencyjnych:



Rysunek 11. Koder, na podst. Koehn, 2017

Dekoder również stanowi sieć rekurencyjną. Na wejściu wymaga on reprezentacji kontekstu zdania wejściowego, przewidywanego słowa docelowego oraz poprzedniego stanu ukrytego i na ich bazie generuje nowy ukryty stan oraz nowe przewidywane słowo docelowe, co przedstawione jest na rysunku 12.



Rysunek 12. Dekoder, na podst. Koehn, 2017

Aby połączyć reprezentację słów pochodzącą z kodera z siecią neuronową dekodera oczekującego na kontekst c_i w każdym kroku *i*, stosuje się mechanizm atencji (ang. *attention mechanism*). Mechanizm atencji otrzymuje informację o wszystkich reprezentacjach słów wejściowych oraz o poprzednim ukrytym stanie dekodera, tworząc kontekst c_i. Obliczanie powiązania pomiędzy stanem dekodera a każdym słowem wejściowym zwraca informację o wpływie danego słowa wejściowego na następne słowo wyjściowe (Koehn, 2017).

W celu zakończenia procesu trenowania sieci neuronowych wprowadza się kryterium stopu. Aby móc określić, kiedy przerwać proces trenowania stosuje się metryki walidacyjne. Przykładem takiej metryki jest entropia krzyżowa (ang. *cross-entropy*), czyli średnia z ujemnego logarytmu z prawdopodobieństw słów. W przypadku stosowania tej metryki kryterium stopu to brak poprawy funkcji kosztu. Entropię krzyżową definiuje się wzorem 11, gdzie *w* oznacza słowo, a *LM* to model języka.

$$H(p_{LM}) = -\frac{1}{n} log\ P_{LM}(w_1, w_2, \ldots, w_n) = -\frac{1}{n} \sum_{i=1}^{n} log\ P_{LM}(w_i | w_1, \ldots, w_{i-1})$$

Wzór 11.

4. Narzędzia zastosowane w projekcie magisterskim

W projekcie magisterskim opisanym w poniższej pracy zastosowano dwa narzędzia do tworzenia systemów tłumaczących. Pierwszy z nich to program Moses wykorzystywany do trenowania statystycznych systemów tłumaczenia automatycznego. Natomiast drugi program to Marian NMT służący do budowania neuronowych systemów tłumaczenia automatycznego.

4.1. Program Moses

Program Moses służy do statystycznego tłumaczenia automatycznego. Pozwala na wytrenowanie systemu tłumaczącego z języka źródłowego na język docelowy na podstawie korpusów równoległych. Program Moses składa się z dwóch podstawowych komponentów: potoku treningowego i dekodera. W pierwszym z nich stosuje się zestaw narzędzi napisanych głównie w językach programowania Perl i C++, które budują model tłumaczenia, korzystając z surowych tekstów. Dekoder to program w języku C++ tłumaczący tekst źródłowy na język docelowy z wykorzystaniem modelu tłumaczenia.

Program Moses to darmowe narzędzie typu *open-source* na licencji LGPL (*Lesser General Public License*) dostępne w serwisie Github (https://github.com/moses-smt/mosesdecoder.git).

4.1.1. Instalacja

Z programu Moses można korzystać pod takimi systemami operacyjnymi jak Linux, Windows oraz MacOS. Wszystkie kroki instalacji wraz z odpowiednimi komendami do uruchomienia z poziomu konsoli systemowej, opisane są w podręczniku programu Moses dostępnym na stronie internetowej http://www.statmt.org/moses/.

W przypadku poniższej pracy program Moses został zainstalowany pod systemem Ubuntu 16.04.

4.1.2. Przygotowanie korpusów

Przed wykorzystaniem korpusu do trenowania systemu tłumaczącego należy go najpierw odpowiednio przygotować. Większość narzędzi potrzebnych do tego celu instaluje się wraz z programem Moses. Najważniejsze z nich to:

- narzędzie do tokenizacji (*tokenizer*),
- narzędzia do zmiany wielkości liter (*truecaser* lub *lowercaser*),
- narzędzie do czyszczenia korpusu ze zdań zbyt krótkich lub zbyt długich.

Tokenizacja służy do odseparowania słów i znaków interpunkcyjnych spacjami. W przypadku języka polskiego jest to stosunkowo proste zadanie, gdyż każde słowo w zdaniu jest naturalnie od siebie oddzielone. Jednak należy jeszcze oddzielić znaki interpunkcyjne od słów, do czego stosuje się skrypt *tokenizer.perl* wywoływany następującym poleceniem:

```
~/mosesdecoder/scripts/tokenizer/tokenizer.perl -l pl \
                < korpus.pl > korpus.token.pl
```

W przypadku języka japońskiego tokenizacja nie jest już takim łatwym procesem, gdyż między znakami zazwyczaj nie występują spacje. W tym celu trzeba znaleźć inny sposób na utworzenie tokenów, gdyż program Moses nie dostarcza narzędzi dla tego języka. Najprostszym sposobem jest rozdzielenie zdania znak po znaku, jednak nie odpowiada to zazwyczaj rzeczywistym słowom, które mogą składać się z wielu znaków. Istnieją również specjalne programy przeznaczone dla języka japońskiego, które rozdzielają zdania na słowa, choć żadne z nich nie osiąga stuprocentowej dokładności, więc część zdań może zostać rozdzielona w niepoprawny sposób. Przykładem narzędzia specjalizującego się w tokenizacji języka japońskiego jest biblioteka języka Python zwana TinySegmenter (https://pypi.python.org/pypi/tinysegmenter).

Kolejnym ważnym elementem przygotowania tekstu jest konwersja dużych liter na małe, aby nie miały one wpływu na działanie programu Moses, dla którego mała i duża litera to zupełnie inny znak. Proces o nazwie *lowercasing* konwertuje wszystkie duże litery do małych, natomiast proces *truecasing* konwertuje głównie duże litery występujące na początku zdania. W przypadku procesu *truecasing* najpierw należy wytrenować specjalny model zawierający informację o tym, które słowa powinny zaczynać się wielką literą. Do trenowania można zastosować ten sam korpus, który zostanie później poddany procesowi *truecasing*. W przypadku poniższego polecenia zastosowano korpus po tokenizacji – *korpus.token.pl*.

```
~/mosesdecoder/scripts/recaser/train-truecaser.perl \
  –model  truecase-model.pl --corpus korpus.token.pl
```

Proces *truecasingu* uruchamia się komendą, stosując skrypt truecase.perl:

```
~/mosesdecoder/scripts/recaser/truecase.perl \
–model truecase-model.pl < korpus.token.pl >
korpus.true.pl
```

Ostatnim etapem jest usunięcie nadmiarowych spacji, pustych linii i zdań przekraczających długość ustaloną przez użytkownika. Skrypt clean-corpus-n.perl usuwa zdania z obu korpusów za jednym razem, dlatego nazwy plików zawierających korpusy muszą być takie same (np. *korpus.true*) i różnić się tylko oznaczeniem języka (w przypadku japońskiego będzie to *ja*, a polskiego: *pl*). W poniższym przykładowym poleceniu usunięte zostaną puste linie oraz dłuższe niż 80:

```
~/mosesdecoder/scripts/training/clean-corpus-n.perl \
korpus.true ja pl korpus.clean 1 80
```

4.1.3. Trenowanie modelu języka

Gdy korpusy w języku źródłowym i docelowym zostaną już odpowiednio przygotowane, można rozpocząć trenowanie modelu języka docelowego, w przypadku niniejszej pracy – języka polskiego. Model języka poprawia płynność zdań wynikowych, więc stanowi bardzo istotny element w tworzeniu systemu tłumaczącego. Można tworzyć modele o wybranej długości historii słów, na przykład model 3-gramowy. Taki model buduje się programem *lmplz* z flagą *-o* ustawioną na 3:

```
~/mosesdecoder/bin/lmplz -o 3 < korpus.true.pl \
> korpus.arpa.pl
```

Aby utworzony model ładował się szybciej podczas tworzenia systemu tłumaczącego, należy go poddać binaryzacji:

```
~/mosesdecoder/bin/build_binary \
korpus.arpa.pl korpus.blm.pl
```

4.1.4. Trenowanie modelu tłumaczenia

Mając gotowe korpusy i model języka można przejść do najważniejszego punktu procesu tworzenia systemu tłumaczącego, czyli **trenowania modelu tłumaczenia**. Składa się on z następujących elementów:

- dopasowania słów z zastosowaniem programu GIZA++ i algorytmów heurystycznych,
- ekstrakcji fraz i ich oceny,
- stworzenia tabel leksykalnej zmiany kolejności wyrazów,
- zbudowania modelu generacji,
- utworzenia pliku konfiguracyjnego moses.ini.

Program GIZA++ to ogólnodostępna implementacja modeli IBM szerzej opisanych w podrozdziale 3.1.1. Jest on potrzebny do uzyskania dopasowania słów języka źródłowego do słów języka docelowego. Program GIZA++ tworzy dopasowania w dwie strony: dopasowanie słowa w języku A do słowa w języku B oraz odwrotnie – dopasowanie słowa w języku B do słowa w języku A. W następnym kroku ustala się kombinację tych dwóch wersji za pomocą algorytmów heurystycznych. Domyślną heurystyką jest algorytm *grow-diag-final*, który działa w taki sposób, że zaczyna od części wspólnej dwóch dopasowań (z języka A do języka B oraz na odwrót), a następnie uzupełnia je o dodatkowe miejsca dopasowania. Gdy wszystkie dopasowania słów zostaną już ustalone, na ich podstawie powstaje tablica maksymalnego prawdopodobieństwa tłumaczenia leksykalnego (tłumaczenia słowa w języku źródłowym na słowo w języku docelowym).

Kolejnym krokiem jest ekstrakcja fraz i przypisanie im punktów dopasowania, czyli oszacowania prawdopodobieństwa tłumaczenia danej frazy w języku źródłowym na frazę w języku docelowym. Następnie powstaje model zmiany kolejności wyrazów (ang. *reordering model*), który zawiera informację o koszcie pominięcia lub zamiany słów w ciągu wyrazów podczas ich tłumaczenia. Na przykład z pominięciem dwóch słów wiąże się większy koszt, niż w przypadku pominięcia jednego słowa.

W przedostatnim kroku trenowania powstaje ostateczny model tłumaczenia. Ostatni krok to stworzenie pliku konfiguracyjnego o nazwie *moses.ini* zawierającego ścieżki do elementów wygenerowanego modelu i domyślne wartości parametrów tłumaczenia.

Wszystkie wymienione wcześniej kroki wykonują się po uruchomieniu jednego polecenia:

```
~/mosesdecoder/scripts/training/train-model.perl \
-root-dir train -corpus korpus.clean -f ja -e pl \
   -alignment grow-diag-final-and -reordering \
            msd-bidirectional-fe -lm \
      0:3:$HOME/lang-model/korpus.blm.pl:8 \
-external-bin-dir ~/mosesdecoder/tools >& training.out &
```

Następnie tabelę fraz (ang. *phrase-table*), która powstaje podczas procesu trenowania modelu, poddaje się binaryzacji:

```
~/mosesdecoder/bin/processPhraseTableMin \
-in train/model/phrase-table.gz -nscores 4 \
     -out binarised-model/phrase-table
```

4.1.5. Tuning

Tuning to proces w tworzeniu systemu tłumaczącego, który ma za zadanie poprawić jakość tłumaczenia. Wymaga on niewielkiego korpusu równoległego, na bazie którego ustalane są wartości parametrów w pliku konfiguracyjnym *moses.ini*, na przykład kary za nieznane słowo (ang. *unknown word penalty*). Tekst nie może powielać się z tekstem zastosowanym do trenowania systemu tłumaczącego. Polecenie uruchamiające tuning ma następującą postać:

```
~/mosesdecoder/scripts/training/mert-moses.pl \
 korpus_tuning.true.ja korpus_tuning.true.pl \
~/mosesdecoder/bin/moses train/model/moses.ini --mertdir \
      ~/mosesdecoder/bin/ &> mert.out &
```

4.1.6. Testowanie

Gdy proces budowania systemu tłumaczącego dobiegnie końca, można go uruchomić następującym poleceniem z wykorzystaniem pliku konfiguracyjnego *moses.ini*:

```
~/mosesdecoder/bin/moses -f moses.ini
```

Następnie należy wpisać zdanie w języku źródłowym i poczekać na wynik tłumaczenia. Więcej informacji na temat trenowania systemu tłumaczącego i jego testowania można przeczytać w podręczniku programu Moses (http://www.statmt.org/moses/).

4.2. Program Marian NMT

Program Marian NMT to narzędzie napisane w języku programowania C++ służące do neuronowego tłumaczenia automatycznego. Zostało ono opracowane na Uniwersytecie im. Adama Mickiewicza w Poznaniu oraz na Uniwersytecie Edynburskim. Można z niego korzystać za darmo, dzięki licencji MIT. Dostępne jest do pobrania w serwisie GitHub (https://github.com/marian-nmt/marian). Tak jak w przypadku programu Moses, program Marian NMT wymaga korpusów równoległych do wytrenowania systemu tłumaczącego.

4.2.1. Instalacja

Z programu Marian NMT można korzystać tylko pod systemem operacyjnym Linux. Sama instalacja narzędzia nie jest skomplikowana, gdyż należy jedynie wykonać kilka poleceń, które znajdują się w dokumentacji na stronie https://marian-nmt.github.io. Jednakże program wymaga instalacji dodatkowych narzędzi. W przypadku przetestowanego i zalecanego przez twórców systemu Ubuntu 16.04 LTS należy zainstalować następujące narzędzia:

- CMake 3.5.1
- GCC/G++ 5.4
- Boost 1.58
- CUDA 8.0

Głównym ograniczeniem w korzystaniu z programu Marian NMT jest to, że do trenowania systemu tłumaczącego wymagana jest karta graficzna obsługująca platformę NVIDIA CUDA, czyli równoległą architekturę obliczeniową opracowaną przez firmę NVIDIA zapewniającą duży wzrost wydajności obliczeniowej. Platforma NVIDIA CUDA jest dostępna dla takich kart graficznych jak GeForce, ION Quadro i Tesla. W niniejszej pracy wykorzystano kartę NVIDIA GeForce 940M.

W przypadku poniższej pracy program Marian NMT został zainstalowany pod systemem Manjaro 17.1.7, który stanowi dystrybucję systemu Linux opierającą się na dystrybucji Arch Linux. Na początku podjęto próbę instalacji programu Marian na systemie Ubuntu 16.04, jednak w związku z problemem z wykrywaniem dodatkowej karty graficznej (wspomnianej karty NVIDIA GeForce 940M), a co za tym idzie, z instalacją platformy NVIDIA CUDA, zdecydowano o instalacji sytemu Manjaro, na którym można było

rozwiązać te kwestie w prostszy sposób. Należało zainstalować odpowiednie sterowniki dla grafiki hybrydowej (czyli zestawu dwóch kart graficznych w jednym komputerze, w tym wypadku Intel i NVIDIA) poprzez polecenie:

```
mhwd -f -i pci video-hybrid-intel-nvidia-bumblebee
```

A następnie zainstalować platformę NVIDIA CUDA poleceniem:

```
pacman -S cuda
```

Na koniec należało również zainstalować narzędzia: CMake i Boost.

4.2.2. Zastosowanie

W przypadku programu Marian NMT korpusy równoległe należy przygotować za pomocą zewnętrznych narzędzi, na przykład stosując narzędzia programu Moses do obróbki tekstów. Proces przygotowywania tekstów został opisany w podrozdziale 4.1.2.

Program Marian NMT stanowi implementację architektury koder-dekoder opisanej w podrozdziale 3.2.3. Bazuje on w dużej mierze na narzędziu Nematus (Sennrich i in., 2017), które zastało napisane w języku programowania Python. Narzędzie Nematus jest implementacją architektury koder-dekoder podobnej do architektury opisanej w (Bahdanau i in., 2015). Jedną z istotnych różnic między nimi jest zmieniona kolejność etapów w działaniu dekodera. W przypadku systemu z 2015 roku wyróżniano następujące etapy: *Zobacz, Wygeneruj, Zaktualizuj* (ang. *Look, Generate, Update*), natomiast w narzędziu Nematus odwrócono kolejność etapów *Wygeneruj* i *Zaktualizuj*, dzięki czemu bardzo uproszczono implementację dekodera. Do systemu Nematus dodano również nowego rodzaju komórki GRU i zmieniono sposób inicjalizacji warstwy ukrytej dekodera. Wspomniane różnice stanowią tylko część zmian, które zastosowano w systemie Nematus, aby usprawnić budowanie neuronowych systemów tłumaczących. Więcej informacji na ten temat można przeczytać w (Sennrich i in., 2017).

Program Marian NMT umożliwia budowanie modeli różnych typów, między innymi:

- **s2s**: model koder-dekoder oparty na rekurencyjnych sieciach neuronowych z zastosowaniem mechanizmu atencji,
- **transformer**: model zaprojektowany przez Google (Vaswani i in., 2017) oparty wyłącznie na mechanizmach atencji,

- **amun**: odpowiednik modelu Nematus w przypadku, gdy nie stosuje się normalizacji warstwy (ang. *layer normalization*; jest to technika poprawiająca zbieżność rekurencyjnych sieci neuronowych),
- **nematus**: odpowiednik modeli stworzonych z zastosowaniem narzędzia Nematus (Sennrich i in., 2017) przez grupę Edinburgh MT na Konferencję Tłumaczenia Automatycznego (WMT) w roku 2017,
- **lm**: model języka stosujący rekurencyjne sieci neuronowe.

Domyślnym modelem programu Marian NMT, a zarazem zastosowanym w projekcie opisywanym w niniejszej pracy, jest model Amun.

Program Marian NMT składa się z kilku narzędzi, w tym narzędzia *marian*, które służy do budowania kodera oraz narzędzia *amun* do dekodowania tekstu z wykorzystaniem wcześniej zbudowanego modelu typu Amun lub Nematus.

W celu zbudowania systemu tłumaczącego, należy w konsoli wykonać polecenie uruchamiające narzędzie *marian*. Najprostsze polecenie wykorzystuje ustawienia domyślne ma postać:

```
./build/marian \
--train-sets korpus.ja korpus.pl \
--vocabs slownik.ja slownik.pl \
--model model.npz
```

W powyższym poleceniu należy podać nazwę wcześniej przygotowanych korpusów języka źródłowego i docelowego oraz wybrane nazwy dla słowników dla obu języków i nazwę dla modelu. Jeśli słowniki lub model o podanej nazwie zostały już wcześniej utworzone, to zostaną one ponownie wykorzystane w dalszym procesie trenowania systemu tłumaczącego.

Program *marian* działa w taki sposób, że dzieli dane wejściowe na mniejsze partie (ang. *batches*), na których trenuje sieć neuronową. Wielkość takiej partii użytkownik może podać samodzielnie w opcji *--mini-batch* lub pozwolić programowi na dobieranie odpowiedniej wielkości w zależności od zarezerwowanej pamięci (opcja *--mini-batch-fit*). Całość danych wejściowych przetwarzana jest w jednej epoce (ang. *epoch*), czyli jeśli program wykorzysta wszystkie dane wejściowe, to skończy się wtedy dana epoka i zacznie

następna. Użytkownik ma możliwość ustawienia po ilu epokach uczenie się zakończy. Pozwala na to opcja *--after-epochs*.

Narzędzie *marian* ma wiele opcji, które można ustawiać w zależności od preferencji. Ich pełna lista wraz z opisami znajduje się na stronie internetowej https://marian-nmt.github.io/docs/cmd/marian/.

Podczas procesu uczenia systemu tłumaczącego, istnieje możliwość zapisywania modelu tłumaczenia co pewien czas – po określonej liczbie aktualizacji modelu, którą podaje się poprzez opcję *--save-freq*. Dzięki temu można kontrolować, czy model ulepsza się wraz z dłuższym czasem uczenia sieci neuronowej. Automatyczne sprawdzanie jakości modelu można ustawić poprzez opcje walidacji. W opcji *--valid-sets* należy podać zestawy tekstów równoległych w językach źródłowym i docelowym, w opcji *--valid-freq* podaje się po ilu aktualizacjach ma zostać przeprowadzona walidacja, a w przypadku opcji *--valid-metrics* należy podać metodę walidacji, np. entropię krzyżową. Pozostałe opcje walidacji opisane są na stronie internetowej https://marian-nmt.github.io/docs/cmd/marian/.

Narzędzie *amun*, które służy do tłumaczenia tekstu, również zawiera sporo opcji do ustawienia. Są one opisane na stronie internetowej https://marian-nmt.github.io/docs/cmd/amun/. Ważną opcją jest możliwość wyboru, czy proces tłumaczenia ma się odbywać na procesorach CPU czy GPU, dzięki czemu narzędzie to jest mniej wymagające niż narzędzie *marian* i można je uruchomić na większości komputerów. Najprostsze polecenie do uruchomienia procesu tłumaczenia z domyślnymi opcjami jest następujące:

```
./marian/build/amun -m model.npz -s slownik.ja \
                    -t slownik.pl < test.ja
```

W powyższym poleceniu podaje się nazwę wytrenowanego modelu, nazwy słowników oraz plik z tekstem testowym w języku źródłowym.

4.3. Automatyczna ocena tłumaczenia

Manualna ocena tłumaczenia zajmuje sporo czasu, dlatego nie jest to wydajna metoda przy dużej ilości tekstu. W takiej sytuacji z pomocą przychodzi automatyczna ocena (ang. *automatic evaluation*), z której można korzystać w każdym momencie i w dodatku jest zdecydowania tańsza.

Ogólny sposób działania automatycznych ocen tłumaczenia polega na tym, że wynik działania systemu tłumaczenia automatycznego porównywany jest z tłumaczeniami wykonanymi przez człowieka, zwanymi tłumaczeniami referencyjnymi (ang. *reference translation*). Im bardziej zdanie przetłumaczone z użyciem systemu tłumaczenia automatycznego jest podobne do zdania referencyjnego, tym wyżej oceniana jest jego poprawność.

Istnieje wiele metryk do automatycznej oceny, jednak najbardziej powszechnie stosowaną jest BLEU (*Bilingual Evaluation Understudy*). Ocena BLEU opiera się na dopasowywaniu n-gramów występujących w tłumaczeniu automatycznym do n-gramów tłumaczenia referencyjnego.

Metrykę BLEU definiuje się za pomocą wzoru 12.:

$$BLEU = kara\_za\_zwiezlosc \exp \sum_{i=1}^{n} \lambda_i \log precyzja_i$$

Wzór 12.

Precyzja ze wzoru 12. to stosunek liczby poprawnie przetłumaczonych słów w n-gramie do liczby wyrazów w całym przetłumaczonym n-gramie, co obrazuje wzór 13.

$$precyzja = \frac{liczba\_poprawnie\_przetlumaczonych\_slow}{liczba\_slow\_w\_n\_gramie}$$

Wzór 13.

Natomiast kara za zwięzłość (ang. *brevity penalty*) definiowana jest wzorem 14.:

$$kara\_za\_zwiezlosc = min\left(1, \frac{dlugosc\_zdania\_wyjsciowego}{dlugosc\_zdania\_referencyjnego}\right)$$

Wzór 14.

Kara za zwięzłość jest karą za opuszczanie słów podczas tłumaczenia, czyli za zbyt krótkie zdanie wyjściowe. Poniższy przykład obrazuje zastosowanie kary za zwięzłość:

Zdanie referencyjne: *To oczywiste , że każdy lubi inne książki*

Zdanie przetłumaczone przez system A: *To normalne , że każdy lubi książki*

Zdanie przetłumaczone przez system B: *Oczywiście każdy lubi inne książki*

	System A	System B
Kara za zwięzłość	7/8	5/8

Tabela 10. Przykład kary za zwięzłość

W tabeli 10. przedstawiono wartości kary za zwięzłość dla zdań przetłumaczonych przez dwa różne systemy tłumaczące. Kara dla systemu A wyniosła 7/8, ponieważ zdanie referencyjne ma długość 8, a zdanie wyjściowe ma długość 7. Natomiast w przypadku systemu B kara wyniosła 5/8, gdyż zdanie wyjściowe ma długość 5. Niska wartość kary za zwięzłość działa niekorzystnie na ocenę BLEU systemu tłumaczącego.

5. Projekt magisterski

5.1. Cel projektu

Głównym celem projektu magisterskiego opisanego w niniejszej pracy było porównanie efektywności statystycznego i neuronowego tłumaczenia automatycznego w tłumaczeniu tekstu z języka japońskiego na język polski. Istotnym elementem projektu była również analiza wpływu różnych czynników na jakość działania systemów tłumaczących.

Zagadnienie tłumaczenia automatycznego między językiem japońskim a polskim na dzień dzisiejszy nie należy do często analizowanych. Prawdopodobnie jest to pierwsza praca poświęcona tej parze języków, w której przeprowadzono rzeczywiste badania. Co prawda istnieje artykuł opisujący trudności w automatycznym tłumaczeniu języka japońskiego na język polski oraz proponujący sposób na utworzenie systemu tłumaczącego dla tej pary języków (Świeczkowska, 2017), jednak nie wspomniano w nim o przeprowadzeniu jakichkolwiek prób zbudowania takiego systemu. W dodatku zaproponowana metoda pozyskania korpusu równoległego poprzez skorzystanie z profesjonalnych polskich tłumaczeń japońskich komiksów (zwanych *manga*) nie jest łatwa do zrealizowania, gdyż trudno znaleźć ogólnodostępne „surowe" teksty z takich komiksów w obu językach. Większość japońskich komiksów występuje w postaci graficznej, więc automatyczne pobranie z nich tekstów oraz dopasowanie do siebie odpowiednich fragmentów jest czasochłonne i skomplikowane. Jednak metoda ta może okazać się przydatna dla przyszłych badaczy dysponujących dostępem do dużej ilości komiksów *manga*.

5.2. Pozyskanie zasobów

Jedno z najważniejszych źródeł korpusów równoległych dla języka japońskiego w parze z językiem polskim stanowi strona internetowa http://opus.nlpl.eu/. W przypadku języka japońskiego w połączeniu z polskim ze strony OPUS można pobrać za darmo korpus napisów filmowych stworzony na bazie napisów pochodzących ze strony http://www.opensubtitles.org/. Najnowszy korpus japońsko-polski, OpenSubtitles2018, zawiera ponad 1,5 miliona par zdań.

Przykład danych wejściowych przed ich przetworzeniem znajduje się w tabeli 11.

Korpus języka japońskiego	Korpus języka polskiego
カリガリ博士の小屋 幕 1 僕の婚約者だ	Gabinet doktora Caligari Akt pierwszy To moja narzeczona.

Tabela 11. „Surowe” dane

Natomiast w tabeli 12. pokazane są dane z tabeli 11. po przetworzeniu. Oba korpusy poddano tokenizacji, a w przypadku języka polskiego również procesowi *truecasingu*.

Korpus języka japońskiego	Korpus języka polskiego
カリガリ 博士 の 小屋 幕 1 僕 の 婚約 者 だ	gabinet doktora Caligari akt pierwszy to moja narzeczona .

Tabela 12. Przetworzone dane

Dodatkowo, w ramach projektu magisterskiego przygotowano słownik japońsko-polski składający się z haseł z Wikipedii (https://www.wikipedia.org/) pobranych ze strony https://dumps.wikimedia.org/backup-index.html z wykorzystaniem narzędzia *wikipedia-parallel-titles* dostępnego w serwisie GitHub (https://github.com/clab/wikipedia-parallel-titles). Otrzymano prawie 278 tysięcy par haseł japońsko-polskich.

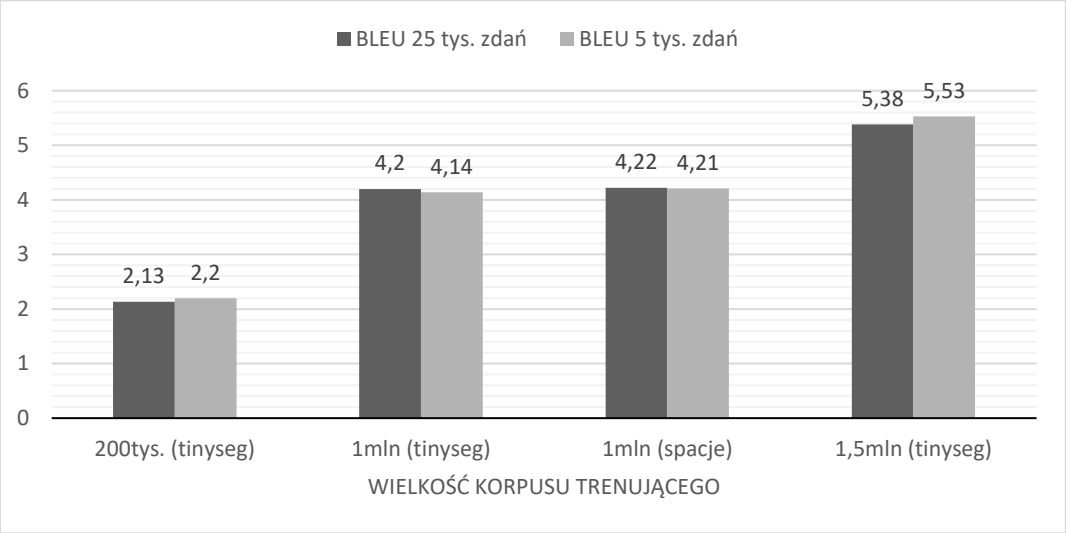
5.3. Zastosowanie programu Moses

W celu zbudowania statystycznego systemu tłumaczącego za pomocą programu Moses, zebrany uprzednio korpus równoległy podzielono na kilka części. Wydzielono 1,5 miliona par zdań z korpusu napisów do wytrenowania systemu tłumaczącego. 25 tysięcy z pozostałego korpusu wydzielono do testów, a do tuningu – 20 tysięcy par zdań.

5.3.1. Trenowanie i testowanie

Aby przedstawić wpływ różnych czynników na jakość tłumaczenia automatycznego z wykorzystaniem programu Moses przygotowano kilka wersji systemów tłumaczących i każdy z nich oceniono stosując metrykę BLEU. Z początku wytrenowane modele tłumaczenia testowane były na korpusie zawierającym 25 tysięcy zdań, jednak był

to proces bardzo czasochłonny, więc przetestowano je również na 5 tysiącach zdań. Różnica w ocenie BLEU była niewielka, więc kolejne modele testowano już tylko na 5 tysiącach. Wykres 1. przedstawia porównanie ocen BLEU dla czterech różnych modeli przetestowanych na 25 tysiącach i 5 tysiącach zdań.



Wykres 1. Porównanie wyników testów na 5 tysiącach i 25 tysiącach zdań

1. Zwiększanie korpusu i dwa sposoby tokenizacji

Badanie działania statystycznego tłumaczenia automatycznego rozpoczęto od systemu trenowanego na 200 tysiącach par zdań, gdzie wynik BLEU wyniósł zaledwie nieco ponad **2**. Następnie zwiększano korpus do 1 miliona par zdań oraz 1,5 miliona par zdań. W przypadku tych trzech systemów korpus w języku japońskim został poddany tokenizacji narzędziem TinySegmenter. Następnie, dla porównania, wytrenowano system na bazie 1 miliona oraz 1,5 miliona par zdań, gdzie w korpusie języka japońskiego każdy poszczególny znak oddzielono spacjami. W przypadku korpusu milionowego nieco lepszy wynik uzyskano przy tokenizacji spacjami (BLEU = **4,21**), jednak już przy wersji o pół miliona większej, narzędzie TinySegmenter okazało się lepsze (BLEU = **5,53**). W związku z tym kolejne systemy trenowano już na korpusie tokenizowanym przez program TinySegmenter.

2. Dodanie haseł z Wikipedii

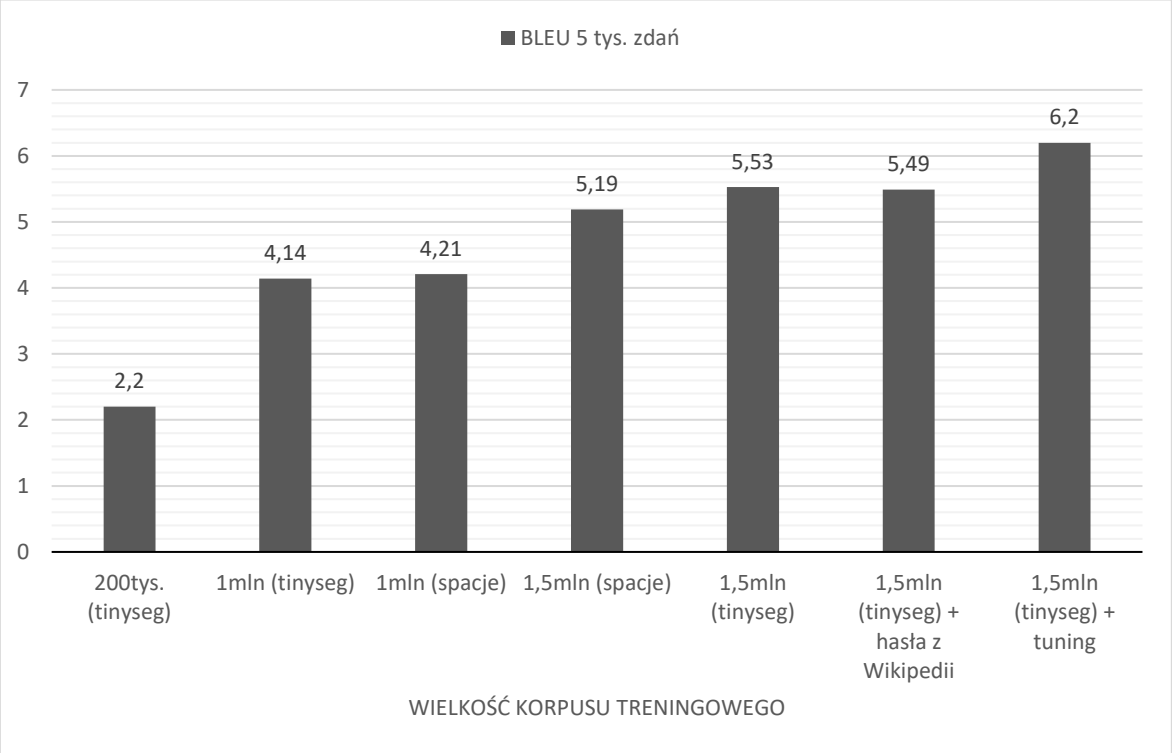
W następnej próbie poprawienia działania systemu tłumaczącego do korpusu trenującego dodano słownik składający się z haseł z Wikipedii, który z założenia miał uzupełnić korpus o wyrazy, które mogły nie znaleźć się w napisach do filmów. Jednak według oceny BLEU nie okazało się to dobrą praktyką, gdyż ocena BLEU spadła do **5,49**. Dla pewności

przeprowadzono dodatkowy test na korpusie 25-tysięcznym i w tym wypadku ocena BLEU wyniosła **5,53**. Dla przypomnienia, ocena BLEU tłumaczenia systemu wytrenowanego na korpusie bez haseł i przetestowanego na 25 tysiącach zdań była równa **5,38** (wykres 1.). W zawiązku z powyższym, nie jest jednoznaczne czy dodanie haseł z Wikipedii miało pozytywny czy negatywny wpływ na tłumaczenie automatyczne.

3. Tuning

Na koniec przeprowadzono proces tuningu na bazie 20 tysięcy par zdań. Jako że jest to najwolniejszy krok, można do niego wykorzystać nawet nieco mniejszy korpus niż miało to miejsce w niniejszym projekcie. Zastosowanie tuningu do poprzednio wytrenowanego na 1,5-milionowym korpusie systemu tłumaczącego zdecydowanie poprawiło ocenę jakości tłumaczenia. Wzrosła ona do **6,2**.

Wykres 2. przedstawia porównanie wytrenowanych systemów tłumaczących według oceny BLEU. Każdy z systemów został przetestowany na korpusie składającym się z 5 tysięcy zdań w języku japońskim.



Wykres 2. Porównanie ocen BLEU dla systemów tłumaczenia trenowanych na różnych danych z użyciem programu Moses

Jak widać na wykresie 2., jakość tłumaczenia automatycznego zdecydowanie wzrastała wraz ze zwiększającą się wielkością korpusów równoległych, jednak dodanie haseł z Wikipedii nieco zaburzyło ten proces.

5.3.2. Przykłady

System tłumaczenia statystycznego wytrenowany na potrzeby niniejszej pracy, pomimo zaledwie 1,5-milionowego korpusu uczącego, w wielu przypadkach zdań testowych okazał się działać co najmniej poprawnie. Przetłumaczonym zdaniom często daleko jest do ideału, lecz są wystarczająco zrozumiałe i oddają sens zdania źródłowego. Kilka przykładów takich zdań znajduje się w tabeli 13. Są to przykłady ze zbioru testowego przetłumaczonego za pomocą modelu, który dał najwyższy wynik BLEU.

Zdanie źródłowe	Zdanie referencyjne	Tłumaczenie zwracane przez system
ほう 情報 を 感謝 し ます 大尉	Dziękuję za informację, komisarzu.	Tak, dziękuję, kapitanie.
もし 、 よろしければ 車 に 乗せ て もらえ ませ ん か	Może by mnie pani gdzieś podrzuciła?	Jeśli, jeśli mógłbyś mnie podwieźć?
聞こえ てる の か ？	Słyszysz mnie w ogóle?	Słyszysz mnie?
命 を 救っ た 、 お前 は 何 を し た ？	Uratował jej życie. Co zrobiłeś?	Uratowałeś mi życie, a ty?
私 を 見ろ	Spójrz na mnie.	Spójrz na mnie.

Tabela 13. Przykłady tłumaczeń zdań w języku japońskim z wykorzystaniem statystycznego systemu tłumaczącego

Część przykładów z tabeli 13. (szczególnie przykład pierwszy) pokazuje przypadki, w których tłumaczenie jest zrozumiałe, jednak niezbyt podobne do zdania referencyjnego. Takie zdania niestety nie podwyższają oceny BLEU, gdyż nie zgadzają się ze zdaniem referencyjnym.

5.4. Zastosowanie programu Marian NMT

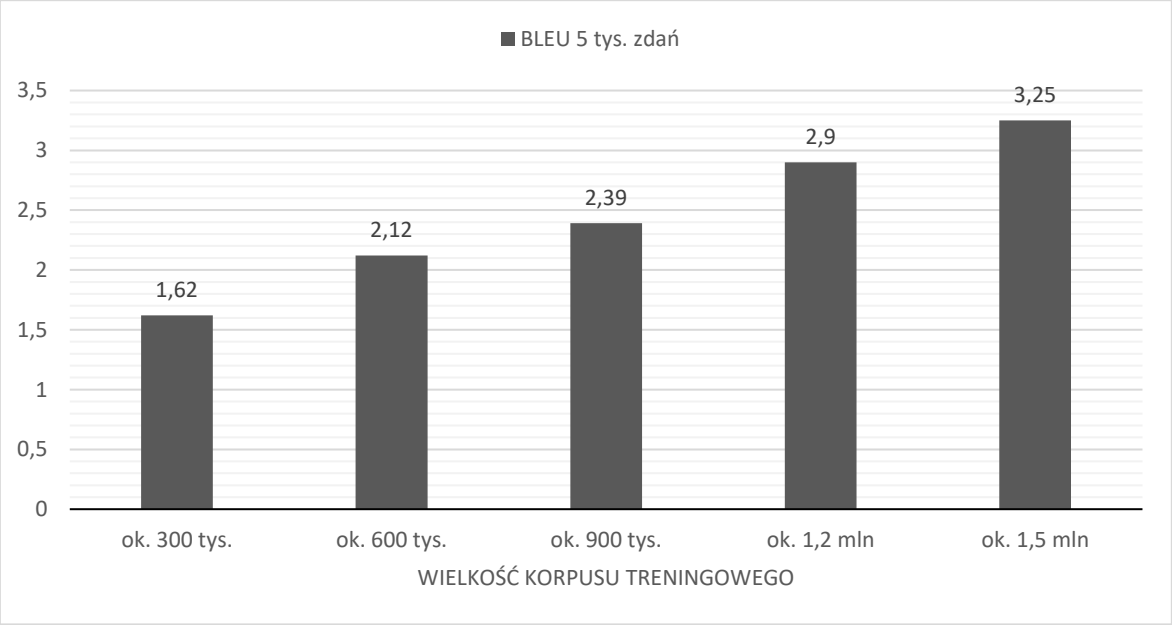
Podobnie jak w przypadku programu Moses, do zbudowania systemu tłumaczącego z użyciem programu Marian NMT z zebranego korpusu równoległego wydzielono 1,5 miliona par zdań. Natomiast 25 tysięcy z pozostałego korpusu wyodrębniono do testów, jednak w związku z czasochłonnością tłumaczenia takiej ilości tekstu, zmniejszono korpus testowy do 5 tysięcy par zdań. Nawet taki pomniejszony zestaw system tłumaczący przetwarzał dane od 15 do 20 minut.

5.4.1. Trenowanie i testowanie

W celu przedstawienia wpływu różnych czynników na jakość tłumaczenia automatycznego z wykorzystaniem programu Marian NMT wytrenowano kilka wersji neuronowych systemów tłumaczących. Następnie przeprowadzono ich ewaluację stosując metrykę BLEU.

1. **Zwiększanie korpusu**

W pierwszej kolejności zbadano wpływ liczby par zdań w korpusie treningowym na ocenę BLEU w pierwszej epoce trenowania, co przedstawia wykres 3. Wraz z rosnącą liczbą przetworzonych zdań, rośnie ocena BLEU. Jednak nie jest to wzrost proporcjonalny, gdyż istnieje wiele czynników, które dodatkowo wpływają na proces trenowania, między innymi jakość zdań w języku źródłowym i docelowym.



Wykres 3. Porównanie ocen BLEU dla neuronowych modeli tłumaczenia trenowanych na różnej liczbie par zdań w pierwszej epoce trenowania

Następnie skupiono się na wytrenowaniu systemu tłumaczącego na 1 milionie par zdań przy zastosowaniu większej liczby epok. Po 11. epoce, czyli po prawie 48 godzinach trenowania, udało się uzyskać ocenę BLEU równą **5,39**. Dalsze próby uczenia sieci neuronowej nie poprawiły już tego wyniku, a wręcz go pogorszyły, więc zakończono proces trenowania.

Kolejny system tłumaczący powstał na 1,5 milionowym korpusie. Po 10 epokach wynik BLEU wynosił **5,88**. Nie jest to jednak diametralny wzrost oceny w stosunku do modelu wytrenowanego na milionie par zdań.

2. **Dodanie haseł z Wikipedii**

Dodanie do korpusu haseł z Wikipedii nie podwyższyło oceny BLEU. Po 10 epokach udało się uzyskać ocenę wynoszącą zaledwie **5,68**. W związku z brakiem poprawy w działaniu systemu, przeprowadzono ponowne trenowanie modelu na korpusie z hasłami. Przy drugim podejściu zmieniono wartości opcji *--dim-vocabs*. Pierwszą wartość stanowiła maksymalna liczba elementów słownika języka źródłowego, a drugą – maksymalna liczba elementów słownika języka docelowego. W poprzednich modelach były to wartości odpowiednio 6600 i 5000, a zmieniono je na 7500 i 7500. Było to maksimum, jakie mogła przetworzyć karta graficzna, na której trenowany był model. Wspomniane słowniki budowane są na podstawie korpusu treningowego i zawierają wszystkie tokeny, jakie się w nim znajdują, z przypisanym identyfikatorem. Zazwyczaj są one zdecydowanie większe niż 7500 tokenów, lecz korzystanie ze wszystkich elementów podczas procesu trenowania zużywa bardzo dużo pamięci karty graficznej. Jednakże warto było zwiększyć ten parametr, gdyż prawdopodobnie dzięki temu uzyskano ocenę BLEU równą **6,04** po 12 epokach.

3. **Eksperyment – zautomatyzowane czyszczenie korpusów**

W ostatniej próbie wytrenowania systemu tłumaczącego przeprowadzono następujący eksperyment. Korpus składający się z 1,5 miliona par zdań został poddany dodatkowemu procesowi czyszczenia. W tym celu napisano skrypt w języku programowania Python, który odrzucał zdania równoległe różniące się od siebie co najmniej dwukrotnie liczbą tokenów. W przypadku zdań w języku polskim odejmowano zawsze jeden token, gdyż właściwie każde zdanie miało na końcu znak interpunkcyjny, w przeciwieństwie do większości zdań japońskich.

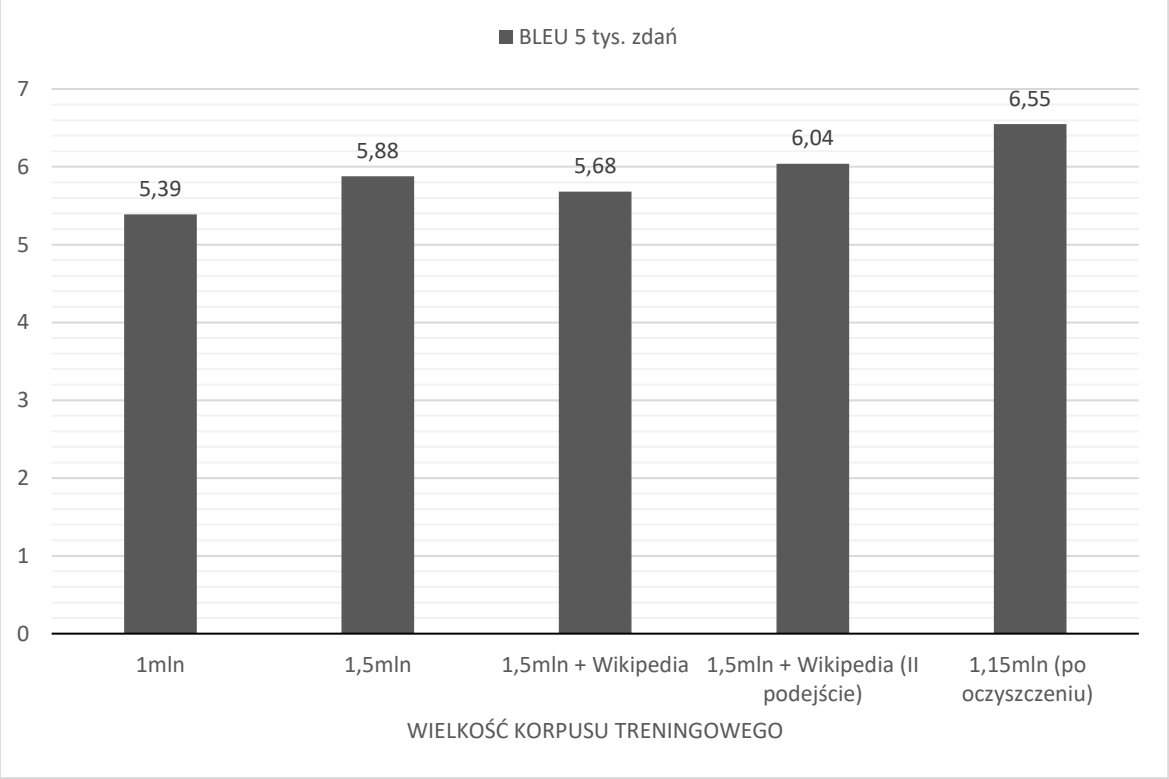
Na przykład zdania:

- **w języku japońskim:** 待て
- **w języku polskim:** *proszę na mnie poczekać !*

zostałyby usunięte, gdyż zdanie japońskie składa się tylko z 1 tokenu, natomiast zdanie polskie – z 5.

Po zastosowaniu opisanego skryptu, zestaw treningowy zmniejszył się do nieco ponad 1,15 miliona par zdań. Wytrenowany na takim oczyszczonym korpusie model tłumaczenia okazał się lepszy od wszystkich uprzednio wytrenowanych modeli. Ocena BLEU wyniosła **6,55**.

Na wykresie 4. pokazane są wartości oceny BLEU dla wytrenowanych neuronowych modeli tłumaczących przetestowanych na 5-tysięcznym korpusie testowym.



Wykres 4. Porównanie ocen BLEU dla neuronowych systemów tłumaczenia trenowanych na korpusach różnej wielkości i jakości

Podobnie jak w przypadku systemu Moses, zwiększenie korpusu treningowego poprawiało ocenę BLEU trenowanych systemów tłumaczących. Nie dotyczyło to jednak ostatniego systemu, który został wyuczony na mniejszej ilości danych, ale lepszej jakości. Można

z tego wnioskować, iż tłumaczenie neuronowe jest mocno podatne na zaszumienie danych, a nie wymaga za to wielkiej ilości danych uczących.

5.4.2. Przykłady

Tłumaczenia zdań z użyciem neuronowego systemu tłumaczenia, który został wytrenowany na oczyszczonym korpusie, choć nie zawsze oddające w stu procentach sens zdania źródłowego, sprawiają dobre wrażenie. Można wręcz powiedzieć, że są to często ładne zdania i poprawne składniowo. Kilka przykładów takich zdań zamieszczono w tabeli 14. Przykłady te pochodzą ze zbioru testowego przetłumaczonego za pomocą neuronowego modelu, który uzyskał najwyższą ocenę BLEU.

Zdanie źródłowe	Zdanie referencyjne	Tłumaczenie zwracane przez system
ここで働いている人を探してるんだけど	Szukam kogoś, kto tu pracuje.	Szukam kogoś, kto tu pracuje.
昨日のある時点でホテルの部屋から失踪しました	Wczoraj zniknął ze swojego pokoju w hotelu.	Zniknął wczoraj z pokoju w hotelu.
命を救ってくれた	Ocaliłaś mi życie.	Uratował mi życie.
我々がここを通っているのは偶然じゃない	Mówiłem, że nie bez powodu jesteśmy w tych okolicach.	To nie przypadek, że tu jesteśmy.
彼女はいなかったすべて跡形もなくなくなってました	Nie było jej tam. - Wszystko zniknęło bez śladu.	Nie było jej. Wszystko zniknęło.
お前の意見が聞きたい	Proszę cię o radę.	Chcę usłyszeć twoją opinię.

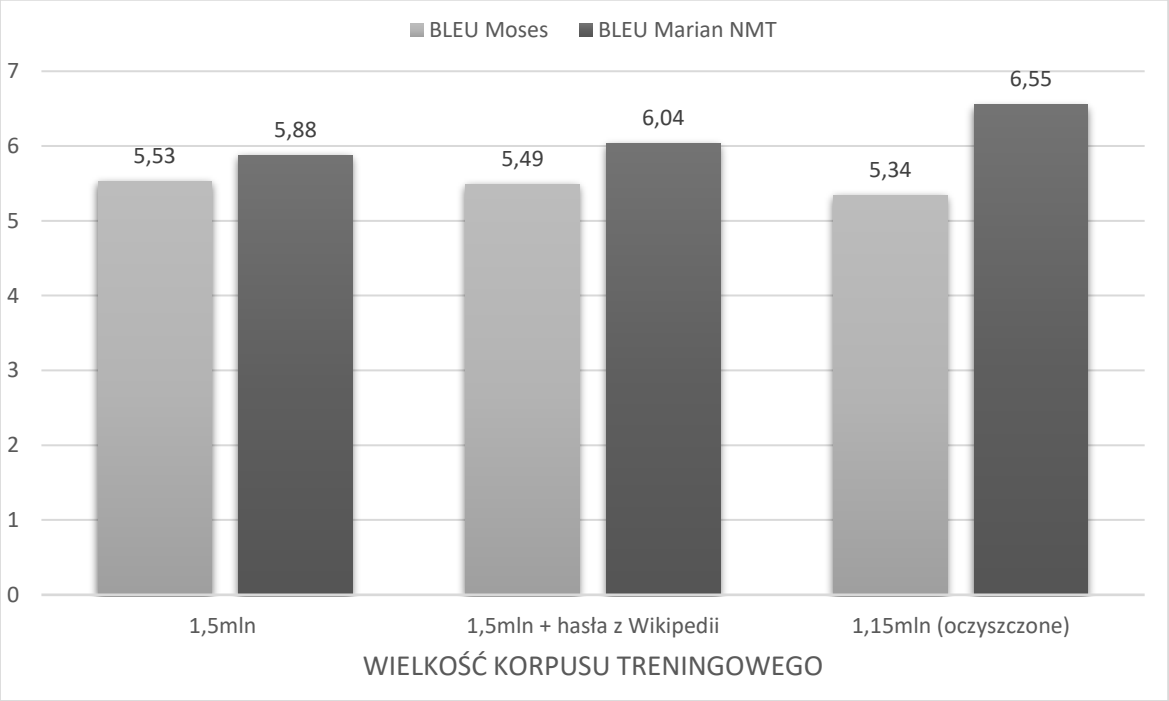
Tabela 14. Przykłady tłumaczeń zdań w języku japońskim z wykorzystaniem neuronowego systemu tłumaczącego

Podobnie jak w przypadku przykładów tłumaczeń uzyskanych poprzez zastosowanie modelu statystycznego, przykłady z tabeli 14. ukazują przypadki, w których tłumaczenie jest zrozumiałe i w dodatku poprawne, jednak zdecydowanie różniące się od zdania referencyjnego. Z tego powodu ocena BLEU systemu tłumaczącego jest zaniżona pomimo poprawności tłumaczeń.

5.5. Porównanie wyników działania programów

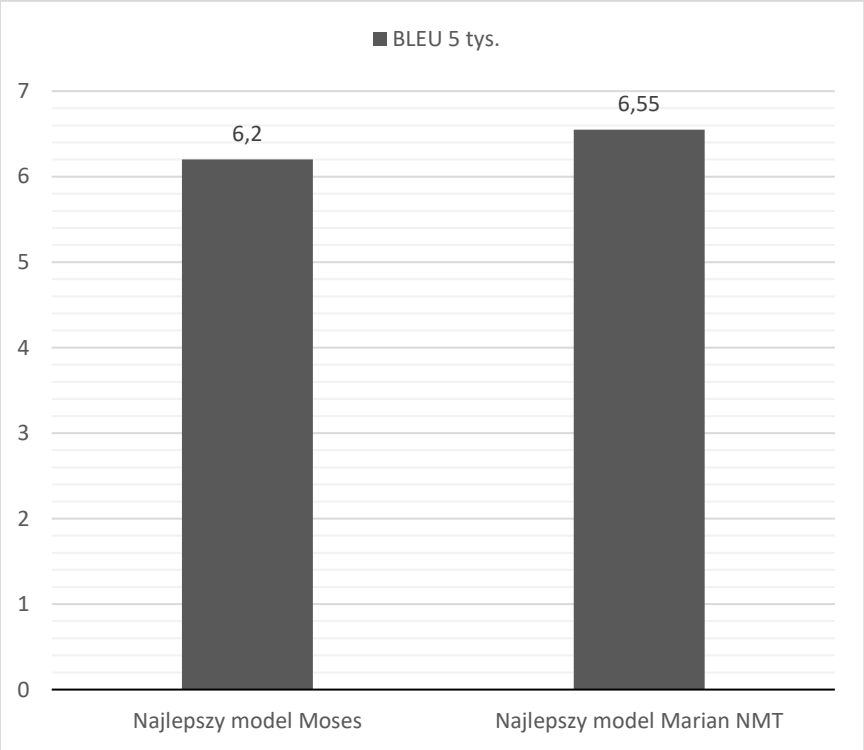
Półtoramilionowy korpus równoległy nie jest wystarczającym materiałem do uzyskania bardzo dobrej jakości systemu tłumaczącego, zwłaszcza biorąc pod uwagę jakość zdań i ograniczoną tematykę korpusu (w tym wypadku napisy filmowe). Dla bardziej popularnych par języków, jak na przykład para angielsko-polska, bez większego problemu można uzyskać korpus kilkunastokrotnie większy, a więc i jakość tłumaczenia automatycznego takich języków może być o wiele lepsza. Nie oznacza to jednak, że wytrenowane na rzecz niniejszej pracy systemy tłumaczące produkują tylko niepoprawne zdania wynikowe.

Na podstawie wyników testów modeli powstałych z użyciem narzędzia Moses oraz narzędzia Marian NMT można zauważyć, że niezależnie od danych uczących, neuronowe tłumaczenie daje nieco lepsze wartości metryki BLEU. Obrazuje to wykres 5.



Wykres 5. Porównanie ocen BLEU systemu tłumaczenia statystycznego (Moses) z ocenami systemu tłumaczenia neuronowego (Marian NMT)

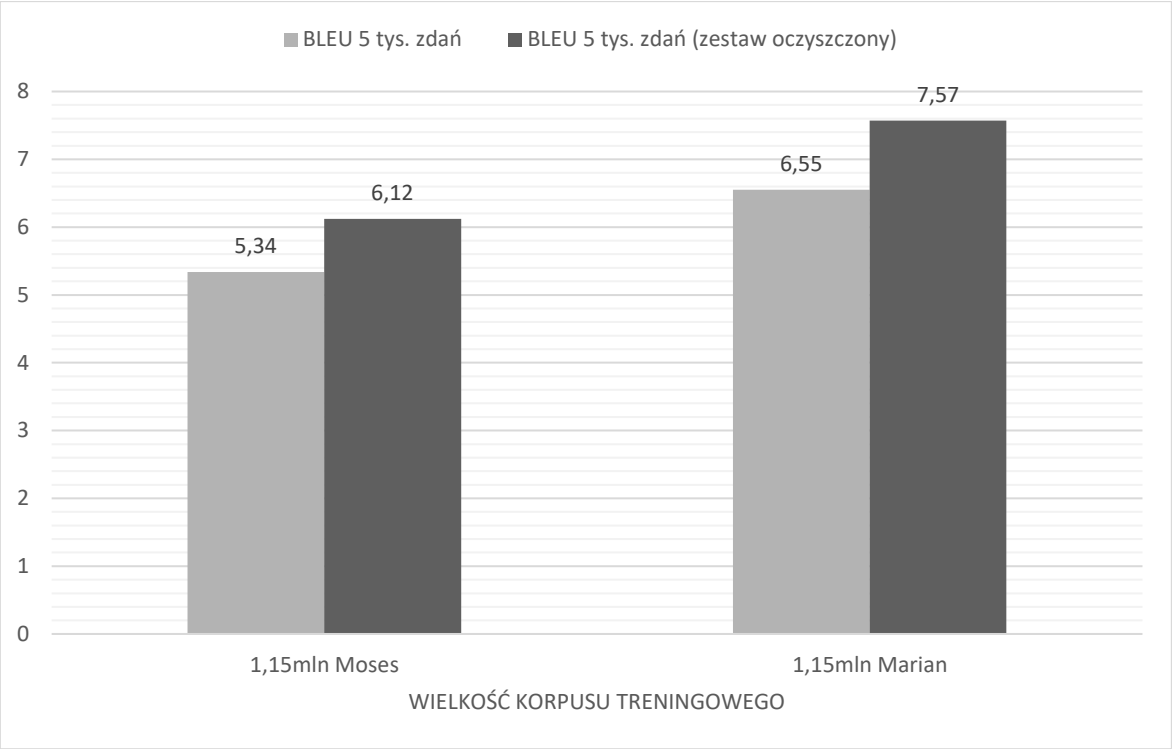
Również porównanie najwyższych ocen BLEU, jakie można było uzyskać dla systemów tłumaczących wytrenowanych z pomocą programu Moses i Marian NMT, jednoznacznie wskazuje na przewagę tłumaczenia neuronowego nad tłumaczeniem statystycznym. Można to zauważyć na wykresie 6. Istotne w tym wypadku jest to, że system statystyczny był wytrenowany na większym korpusie (1,5 miliona par zdań), niż system neuronowy (nieco ponad 1,15 miliona par zdań).



Wykres 6. Porównanie najlepszych modeli

Dla porównania, model statystyczny wytrenowany na oczyszczonym korpusie (ok. 1,15 miliona par zdań) uzyskał ocenę BLEU równą **5,34** przy teście przeprowadzonym na 5 tysiącach zdań testowych (wykres 7.).

Przeprowadzono także dodatkowy test na modelach statystycznym i neuronowym, wytrenowanych na oczyszczonych korpusach. Testowym zbiorem w tym wypadku był korpus składający się z 5 tysięcy zdań uzyskanych z większego korpusu testowego, który został poddany takiemu samemu czyszczeniu, jak korpus treningowy. Poprawiony zestaw testowy zdecydowanie polepszył ocenę BLEU, zarówno w przypadku systemu statystycznego, jak i neuronowego. W ten sposób uzyskano najwyższą ocenę modelu spośród wszystkich przeprowadzonych testów. Wyniosła ona **7,57**. Opisane wyniki zostały zobrazowane na wykresie 7.



Wykres 7. Porównanie modeli powstałych na oczyszczonych korpusach

Podsumowując uzyskane wyniki, zdecydowanie najlepszą opcją trenowania modelu tłumaczenia języka japońskiego na język polski okazało się zastosowanie oczyszczonego korpusu składającego się z ponad 1,15 miliona par zdań.

Aby potwierdzić poprawność wyników automatycznej oceny jakości tłumaczenia porównano manualnie część przetłumaczonych zdań uzyskanych za pomocą najlepszego modelu statystycznego i neuronowego oraz zdań referencyjnych. Kilka przykładów zdań można zobaczyć w tabeli 15.

Zdanie referencyjne	**Tłumaczenie z użyciem najlepszego modelu programu Moses**	**Tłumaczenie z użyciem najlepszego modelu programu Marian NMT**
Wiem, że ciężko dać temu wiarę.	Nadal nie mogę w to uwierzyć.	Nie mogę w to uwierzyć.
Zaufanie jest dla mnie bardzo ważne.	Zaufanie to dla mnie ważne. Panna Crain? Nie wiem.	Zaufanie jest dla mnie bardzo ważne, panie.
Mam tak wiele pytań	Wiele pytań.	Mam wiele pytań.
Ile lat ma już Elvis?	Elvis. Ile ma lat?	Ile ma lat?

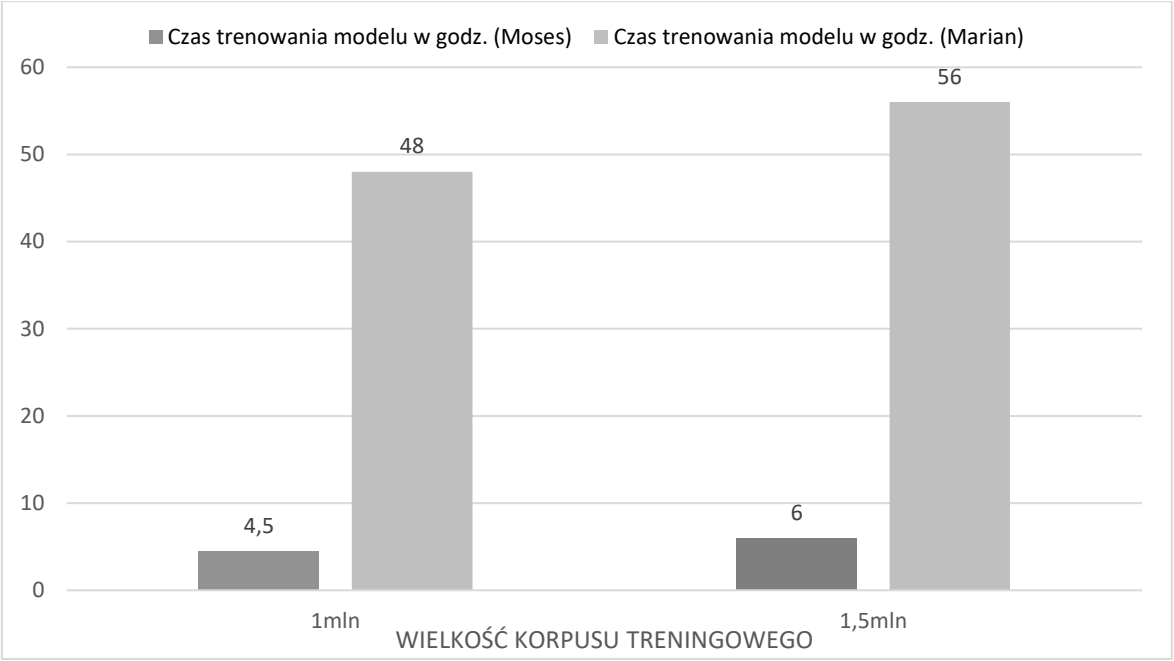
Nigdy bym ci tego nie zrobiła.	Nie, nie, nie, nie, nie.	Nie zrobię tego.
Nie wiem, czy się martwić o to dziecko, czy się go bać .	Nie wiem jak, ale wiem, że ten dziecko, boję się .	Nie wiem, co robić, ale to dziecko.

Tabela 15. Przykłady tłumaczeń zdań w języku japońskim z wykorzystaniem neuronowego i statystycznego systemu tłumaczącego

Po manualnym sprawdzeniu jakości tłumaczeń okazało się, że zdania przetłumaczone neuronowym systemem wyglądają „ładniej" i bardziej przypominają zdania, które wypowiedziałby człowiek. Tak jak pisano w podrozdziałach 5.3.2 i 5.4.2, wiele tłumaczeń z korpusu testowego nie odzwierciedlało zdań referencyjnych, więc mimo swej poprawności zaniżało ocenę BLEU. W związku z tym, gdyby przeprowadzono więcej testów manualnych można by zapewne uzyskać lepszą ocenę wytrenowanych systemów tłumaczących, zwłaszcza w przypadku tłumaczenia neuronowego.

Porównując systemy statystyczne z neuronowymi, warto też wspomnieć, iż czas potrzebny na wytrenowanie neuronowego systemu tłumaczącego jest nieporównywalnie większy od czasu trenowania systemu statystycznego. W przypadku milionowego korpusu, program Moses wytrenował model tłumaczenia w około 4,5 godziny, a przy półtoramilionowym korpusie – w około 6 godzin. Natomiast aby uzyskać w miarę dobrej jakości model neuronowy, program Marian NMT potrzebował prawie 48 godzin trenowania przy milionowym korpusie, a około 56 godzin przy półtoramilionowym. W dodatku czas trenowania statystycznych modeli jest stały i skończony, zależny jedynie od danych wejściowych i parametrów uczenia, podczas gdy trenowanie modeli neuronowych mogłoby trwać przez czas nieokreślony, a więc należy kontrolować, w którym momencie jakość tłumaczenia przestaje się poprawiać. Porównanie czasu trenowania modeli zaprezentowano na wykresie 8.

Na korzyść modeli neuronowych przemawia jednak rozmiar zajmowanej przez nie pamięci. Całość plików powstałych podczas trenowania modelu niezależnie od wielkości danych wejściowych zajmuje łącznie około 0,5 GB, a modele statystyczne zwiększają się zależnie od rosnących danych treningowych i już przy milionowym korpusie zajmują około 1,3 GB.



Wykres 8. Porównanie czasu (w godzinach) trenowania modeli statystycznych i neuronowych

6. Podsumowanie

Na podstawie wyników przeprowadzonych badań nad tłumaczeniem automatycznym języka japońskiego na język polski można stwierdzić, iż tłumaczenie neuronowe ma przewagę nad tłumaczeniem statystycznym dla tej pary języków. Potwierdza to zarówno automatyczna, jak i manualna ocena jakości tłumaczenia.

Co więcej, testy pokazały, że statystyczne systemy tłumaczące uzyskują lepsze wyniki głównie poprzez zwiększanie korpusu uczącego. W przypadku systemów neuronowych ważniejsza jest jednak jakość tekstów, niż ich ilość.

Badania udowodniły również, że istotnym elementem budowania systemów tłumaczących jest odpowiednie przygotowanie korpusów. Duże znaczenie ma tokenizacja, co dotyczy szczególnie języka japońskiego, a także dokładne oczyszczenie tekstów, między innymi ze zdań równoległych zbyt różniących się długością.

Wśród wytrenowanych systemów tłumaczących najwyższą ocenę BLEU uzyskał system, który powstał z pomocą narzędzia Marian NMT. Ocena ta wyniosła 7,57. Taki wynik nie wydaje się być bardzo wysoki w porównaniu do chociażby najlepszego rezultatu tłumaczenia języka japońskiego na język angielski opisanego w (Matsumura i in., 2017), gdzie osiągnięto ocenę BLEU równą 29,73. Trzeba wziąć jednak pod uwagę fakt, iż dane uczące w przypadku niniejszej pracy były stosunkowo małe, a także niezbyt dobrej jakości. W dodatku język polski jest dużo trudniejszy niż język angielski pod względem tłumaczenia automatycznego, między innymi w związku z różnicami w odmianie wyrazów w zależności od płci.

Niezależnie od wyników metryki BLEU, najlepszy system tłumaczący, który powstał na potrzeby niniejszej pracy, w wielu przypadkach pozwala uzyskać dobrej jakości tłumaczenie. Ponadto przeprowadzone badania nad wpływem różnego rodzaju czynników na tłumaczenie, mogą stanowić podstawę do dalszych eksperymentów w celu ulepszenia systemu tłumaczącego teksty w języku japońskim na język polski.

Bibliografia

1. Koehn P., 2010: *Statistical Machine Translation*. Wyd. Cambridge University Press, Nowy Jork
2. Junczys-Dowmunt M., 2008: *Wprowadzenie do metod statystycznych w tłumaczeniu automatycznym*. Online: http://www.inveling.amu.edu.pl/pdf/Marcin_Junczys-Dowmunt_inve16.pdf (dostęp 22.02.2018)
3. Koehn P., 2018: *Statistical Machine Translation: System User Manual and Code Guide*. Online: http://www.statmt.org/moses/manual/manual.pdf (dostęp 24.03.2018)
4. Goodfellow I., Bengio Y., Courville A., 2016: *Deep Learning*. The MIT Press
5. Koehn P., 2017: *Statistical Machine Translation Draft of Chapter 13: Neural Machine Translation*
6. Junczys-Dowmunt M., Grundkiewicz R., Dwojak T., Hoang H., Heafield K., Neckermann T., Seide F., Germann U., Aji A.F., Bogoychev N., Martins A.F.T., Birch A., 2018: *Marian: Fast Neural Machine Translation in C++*
7. Sennrich R., Firat O., Cho K., Birch A., Haddow B., Hitschler J., Junczys-Dowmunt M., Läubli S., Barone A.V.M., Mokry J., Nădejde M., 2017: *Nematus: a Toolkit for Neural Machine Translation*
8. Świeczkowska P., 2017: *Towards a direct Japanese-Polish machine translation system*
9. Bahdanau D., Cho K., Bengio Y., 2015: *Neural Machine Translation by Jointly Learning to Align and Translate. In Proceedings of the International Conference on Learning Representations (ICLR)*
10. Matsumura Y., Sato T., Komachi M., 2017: *English-Japanese Neural Machine Translation with Encoder-Decoder-Reconstructor*
11. Shibatani M., 2017: *Japanese language*. Online: https://www.britannica.com/topic/Japanese-language (dostęp 20.02.2018)
12. https://marian-nmt.github.io [Marian NMT] (dostęp 25.04.2018)
13. http://www.statmt.org/moses/ (dostęp 24.03.2018)
14. http://opus.nlpl.eu/ (dostęp 20.02.2018)
15. http://japonia-online.pl/article/91 [Japonia-Online] (dostęp 20.02.2018)

Spis rysunków