Wrocławska Wyższa Szkoła

Informatyki Stosowanej

Wydział Informatyki

Tatsiana Tsyhankova

Nr albumu: 6783

Porównanie metod projektowania chatbotów opartych na uczeniu maszynowym

Praca: magisterska

Kierunek: Informatyka

Specjalność: Programowanie

Praca wykonana pod kierunkiem:

dr inż. Piotr Ciskowski

Wrocław 2023

Spis treści

Skróty i symbole

Skrót	Objaśnienie
AI	Artificial Intelligence (sztuczna inteligencja)
mld	miliard
AIML	Artificial Intelligence Markup Language (język Znaczników Sztucznej Inteligencji)
CLIR	Cross-lingual information retrieval (wielojęzyczne wyszukiwanie informacji)
DNN	Deep Neural Network (głęboka sieć neuronowa)
CNN	Convolutional Neural Networks (splotowe sieci neuronowe)
RNN	Recurrent Neural Network (rekurencyjna sieć neuronowa)
seq2seq	Sequence to Sequence (sekwencja do sekwencji)
LSTM	Long Short Term Memory (długa pamięć krótkoterminowa)
GRU	Gated Recurrent Unit
biseq2seq	bidirectional seq2seq (dwukierunkowa sekwencja do sekwencji)
NLP	Natural Language Processing (przetwarzanie języka naturalnego)
NLU	Natural Language Understanding (rozumienia języka naturalnego)
DM	Dialogue Manager (menedżer dialogu)
NLG	Natural Language Generation (generowanie języka naturalnego)
CRF	Conditional Random Field (warunkowe pola losowe)
BOS, EOS	znacznikiem początkowym (znacznik końcowy)
LSA	Latent Semantic Analysis (ukryta analiza semantyczna)
POS	part-of-speech tagging (tagowanie części mowy)
NER	Named Entity Recognition (nazwane rozpoznawanie jednostek)

CFGs, DGs	Context-Free Grammar (gramatyki bez kontekstu), Dependency Grammar (gramatyki zależności)
SVM	Support Vector Machine (metoda wektorów nośnych)
SS, SE	węzeł początkowy send_start, węzeł końcowy send_end
CRF	Conditional Random Field Models (warunkowe pola losowe)
HVS	Hidden Vector State (ukryty stan wektorowy)
BPTT	Backpropagation Through Time (propagacja wsteczna w czasie)
TBPTT	Truncated Backpropagation Through Time (obcięta wsteczna propagacja w czasie)
LRCN	long-term Recurrent convolutional Network (długoterminowa, powtarzająca się sieć konwolucyjna)
TF-IDF	TF – term frequency, IDF – inverse document frequency (ważenie częstością termów – odwrotna częstość w dokumentach)
CBOW	continuous bag-of-words (metoda zapisu znaczenia słów w postaci wektorów)
Word2Vec	zapis znaczenia słów w postaci wektorów
BiLSTM	Bidirectional Long Short Term Memory (dwukierunkowa pamięć długoterminowa)
OHE	One Hot Encoding (kodowanie na gorąco)
TPU, GPU	Tensor Processing Units (Jednostka przetwarzania tensora), Graphics processing unit (Procesor graficzny)
API	Application Programming Interface (interfejs aplikacji do programowania)

1. Wstęp

Obecnie firmy oraz marki, działające w wielu różnych obszarach, dynamicznie się rozwijają i chcą polepszyć interakcje z własnymi klientami. Jednoczesne oferowanie usług i komunikowanie się z nimi w różnych kwestiach nabiera masowego charakteru. W warunkach zwiększonego obciążenia pracowników, bardziej niż kiedykolwiek, istotna jest automatyzacja rutynowych działań, a także wykorzystanie narzędzi, które pozwalają dotrzeć do jeszcze większej grupy odbiorców. Jednym ze sposobów osiągnięcia takiego celu jest zastosowanie chatbotów.

Chatboty i czat „na żywo" tworzą systemy wiadomości błyskawicznych, które umożliwiają bezpośrednie dostarczanie użytkownikom wszelkich potrzebnych informacji. W ostatnim czasie popularność chatbotów stale rośnie oraz wciąż się rozwija. Obecne chatboty bardzo różnią się od prostych, wcześniej utworzonych botów, których możliwości bywały ograniczone.

W obecnych czasach, poprzez czerpanie korzyści z używania technologii sztucznej inteligencji, takich jak sieci neuronowe i uczenie maszynowe (coraz częściej stosowanych w rozwoju botów), chatboty stają się coraz bardziej inteligentne, wydajne i konfigurowalne.

W pracy omówiono cechy aplikacji chatbotów i ich typy, zbadano możliwości ich wykorzystania podczas interakcji z klientami w imieniu firmy lub marki. Rozważono niezbędne komponenty AI chatbota. Zbadano również cechy kilku architektur sieci neuronowych, które rozwiązują problemy z prognozowaniem sekwencji i mają zastosowanie przy opracowywaniu prototypu konwersacyjnego chatbota AI.

2. Cele pracy

Głównymi celami tej pracy są:

- poznanie cech konstrukcji i zastosowania chatbotów,
- szkolenie architektury sieci neuronowej Sequence to Sequence[1] na konwersacyjnym zestawie danych, umożliwiającym botowi dopasowanie pytań wejściowych do znaczących odpowiedzi,
- osadzanie uwagi(Attention)[2] na modelu Sequence to Sequence,
- szkolenie modelu Transformer[3],
- porównanie wydajności wyszkolonych modeli za pomocą automatycznych miar oceny,
- porównanie wydajności i ocena wyszkolonych modeli pod kątem zastosowania w konkretnym procesie biznesowym.

[1] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le., *Sequence to Sequence Learning with Neural Networks.* In: CoRR. 2014. Vol. abs/1409.3215. url: http://arxiv.org/abs/1409.3215.
[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate.* In: CoRR.2014.Vol. abs/1409.0473.
[3] Ashish Vaswani et al., *Attention Is All You Need.* In: CoRR.2017.Vol. abs/1706.03762. url: http://arxiv.org/abs/1706.03762.

3. Informacje teoretyczne o chatbotach i funkcjach ich rozwoju

3.1 Pojęcie chatbota i cel tworzenia chatbotów

Przez bota (robota, bota internetowego, bota www) rozumiemy specjalny program, który wykonuje dowolne działania, za pośrednictwem interfejsów zaprojektowanych dla ludzi automatycznie lub zgodnie z pewnym harmonogramem.

Zwykle boty są używane do wykonywania monotonnej i powtarzalnej pracy, aby zmaksymalizować szybkość jej wykonywania. Przykładem takiego zastosowania są roboty wyszukiwarek. Można również użyć bota, aby zapewnić szybszą reakcję na daną sytuację, w porównaniu z reakcją człowieka. Przykładem mogą być boty do gier lub boty do aukcji internetowych. W ostatnim czasie rośnie również popularność korzystania z botów, które imitują ludzkie działania. Dostały one miano „chatbotów".

Chatbot to program, wykrywający potrzeby użytkowników, a następnie pomagający w ich spełnieniu. Takie programy nazywane są również wirtualnymi rozmówcami lub programami rozmówców, konwersacyjnymi AI-botami (Artificial Intelligence), AI-asystentami, inteligentnymi wirtualnymi asystentami, wirtualnymi asystentami klienta, asystentami cyfrowymi, agentami dialogowymi, agentami wirtualnymi, interfejsami dialogowymi, itp.

Wielkość rynku chatbotów wzrośnie z 2,9 mld USD w 2020 r. do 10,5 mld USD do 2026 r., przy złożonej rocznej stopie wzrostu (Compound Annual Growth Rate (CAGR)) wynoszącej 23,5% w okresie prognozy. Czynniki przyczyniające się do rozwoju rynku chatbotów obejmują wzrost zapotrzebowania na obsługę klienta 24/7 przy niższych kosztach operacyjnych, zwiększenie koncentracji na zaangażowanie klientów za pośrednictwem różnych kanałów oraz postęp technologiczny w połączeniu z rosnącym zapotrzebowaniem klientów na operacje samoobsługowe, oferując tym samym przewagę konkurencyjną dla firm[4].

Z reguły chatbot automatycznie komunikuje się z użytkownikiem za pomocą tekstu lub głosu w imieniu firmy lub marki, w celu uproszczenia komunikacji online, aby zapewnić użytkownikowi najbardziej aktualne informacje. Komunikacja użytkownika z chatbotem jest zatem alternatywą dla korespondencji z operatorem na żywo lub połączenia z menedżerem firmy.

[4] *Chatbot Market* (2021) https://www.marketsandmarkets.com/Market-Reports/smart-advisor-market-72302363.html (dostęp 24.06.2022 r.).

Chatbot to program, który działa wewnątrz komunikatora, a konkretniej wewnątrz systemu wiadomości błyskawicznych. Jest on w stanie odpowiadać na pytania, a także samodzielnie je zadawać. Jednocześnie chatboty mogą również działać jako asystenci cyfrowi, przebywając w komunikatorze, wykonując polecenia, udzielając rekomendacji i prowadząc wyszukiwania według parametrów, zadawanych przez użytkownika. Chatbot może stanowić również część witryny i może być, na przykład, dostępny dla użytkownika za pośrednictwem pewnego widżetu[5].

Chatbot to zatem program komputerowy przeznaczony do symulowania komunikacji z ludźmi za pośrednictwem Internetu, aby pomóc użytkownikom rozwiązać pewne sytuacje, z wykorzystaniem najbardziej naturalnej formy komunikacji – w języku zrozumiałym dla odbiorcy. Prowadzi on więc werbalną lub niewerbalną interakcję z osobą za pośrednictwem środowiska online w celu rozwiązania jej spraw. Chatbot może zautomatyzować zadania, a dodatkowo może podzielić je na inne segmenty, takie jak obsługa klienta, ankiety, planowanie spotkań i inne. Chatboty z powodzeniem pomagają, na przykład, w rozwiązywaniu spraw związanych z określonymi markami, odpowiadają na często zadawane pytania lub dostarczają klientom informacji o produkcie czy usłudze. Prócz powyższego, boty mogą również pomagać klientom w umawianiu się na spotkania,, planowaniu podróży, dowiadywaniu się o dostępności produktów i wielu innych kwestiach[6].

Najbardziej przydatne chatboty są opracowywane w oparciu o technologie uczenia maszynowego i wyróżniają się tym, że są szkolone przez ludzi oraz przeznaczone do użytku przez człowieka. W ten sposób osoba bierze aktywny udział we wszystkich etapach rozwoju programu-rozmówcy. Wraz ze wzrostem upowszechnienia i korzystania z chatbotów, paradygmat interakcji „messaging-as-an-interface" nadal zyskuje na popularności, a zgodnie z nim wiadomości, które są szybkie, bezpośrednie, dobrze zwizualizowane i umożliwiające komunikację z dużą liczbą rozmówców jednocześnie. Stają się one więc jednym z kluczowych sposobów komunikacji. Rosnącą popularność tego paradygmatu ułatwia również fakt, że większość przedsiębiorstw opanowuje sieci społecznościowe, które służą jako kanały możliwej komunikacji z klientami. Kolejnym powodem jest fakt, iż dość często użytkownicy czują się bardziej komfortowo, pisząc swoje pytania i otrzymując bezpośrednią odpowiedź, niż zadając je podczas rozmowy „na żywo" z operatorem.

[5] *Czym jest chatbot?* https://www.oracle.com/pl/chatbots/what-is-a-chatbot/ (dostęp 24.06.2022 r.).
[6] Sourabh Nagar, *What Is a Chatbot?* (2018) https://dzone.com/articles/3-minute-guide-to-understand-what-is-a-chatbot (dostęp 24.06.2022 r.).

Chatbot to program, z którym w interakcję w jednym momencie, może wchodzić nie jeden użytkownik, ale kilku czy nawet kilkunastu, co pozwala dotrzeć do wielu odbiorców. Użytkownicy mogą rozmawiać z chatbotem na różne sposoby: za pomocą wiadomości tekstowych, głosu, gestów i kliknięć elementów interfejsu (jeśli są one przewidziane dla konkretnego bota). Jednocześnie, chatbot jest dostępny dla użytkowników przez całą dobę, bez przerw, w przeciwieństwie do operatora na żywo[7].

Chatboty są używane w środowiskach obsługi klienta dopiero od kilku lat, ale obecnie spektrum ról w korzystaniu z chatbotów rośnie wszędzie - napędzane jest to dążeniem do poprawy jakości usług i wydajności biznesowej w różnych przedsiębiorstwach.

Należy pamiętać, że chatboty charakteryzują się również różnym stopniem inteligencji: od najprostszych chatbotów, które mogą być nieco więcej niż interfejsowym rozwiązaniem do odpowiadania na standardowe pytania, po zaawansowane konwersacyjne chatboty AI z inteligentnymi możliwościami i możliwością rozumienia, co pisze do nich użytkownik, a także uczenia się i dostosowywania do cech konkretnego użytkownika, na przykład jego zachowania lub sposobu komunikacji[8].

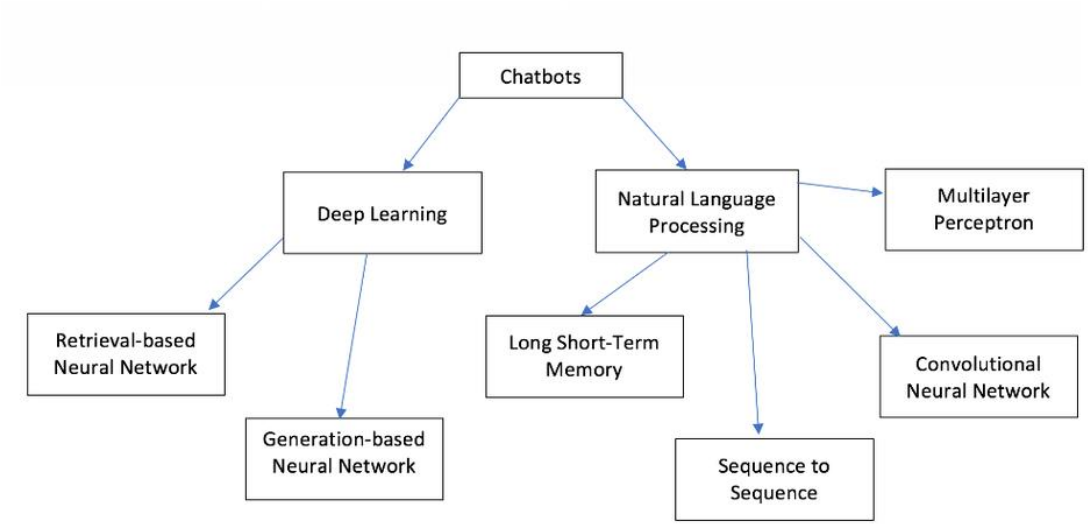
3.2 Wprowadzenie do AI chatbota

Wiele firm zdaje sobie sprawę z niebagatelnych korzyści związanych z włączeniem konwersacyjnej sztucznej inteligencji do swojego modelu biznesowego. Od pierwszego chatbota ELIZA, po dzisiejszą Alexę Amazon, chatboty przeszły długą drogę. Nowoczesne systemy sztucznej inteligencji mogą wchodzić w interakcje z użytkownikami, rozumieć ich potrzeby, dopasowywać ich preferencje i zalecać odpowiednią linię działania przy minimalnej interwencji człowieka lub nawet i bez niej. Obecnie dostępnych jest wiele popularnych agentów konwersacyjnych, takich jak Siri firmy Apple, Cortana firmy Microsoft, Asystent Google i Alexa firmy Amazon. Głównym celem chatbotów jest zapewnienie jak najlepszej odpowiedzi na każde otrzymane żądanie.

Poniższy rysunek pokazuje koncepcyjną mapę chatbota wykorzystującą głębokie uczenie się.

[7] Dharmi Gohil, *A Guide on Chatbots* (2018) https://dzone.com/articles/here-is-a-complete-guide-of-chatbots (dostęp 25.06.2022 r.).

[8] *Chatbots:The Definitive Guide* https://www.artificial-solutions.com/chatbots (dostęp 25.06.2022 r.).



Rysunek 3.1 - Koncepcyjną mapę chatbota[9]

Chatbot musi być w stanie zrozumieć intencje wiadomości nadawcy, określić, jaki rodzaj odpowiedzi (pytanie sekwencyjne, odpowiedź bezpośrednia, itp.) jest wymagany, a ponadto przestrzegać prawidłowych zasad gramatycznych i leksykalnych podczas tworzenia odpowiedzi. Niektóre modele mogą wykorzystywać dodatkowe metainformacje z danych, takie jak identyfikator mówcy, płeć, emocje. Czasami analiza nastrojów jest używana, aby chatbot mógł „zrozumieć" nastrój użytkownika, analizując wskazówki werbalne i strukturę zdań.

[9]Vyas Ajay Bhagwat, *Deep Learning for Chatbots* (2018). *Master's Projects.* 630.
DOI: https://doi.org/10.31979/etd.9hrt-u93z.

4. Rodzaje chatbotów

4.1 Chatbot oparty na regułach

W podejściu opartym na regułach, bot odpowiada na pytania w oparciu o niektóre reguły, na podstawie których został przeszkolony. Niektóre reguły mogą być zarówno bardzo proste, jak i złożone. Tworzenie botów jest stosunkowo proste przy użyciu podejścia opartego na regułach, ale bot nie będzie wtedy skuteczny w odpowiadaniu na pytania, których wzór nie pasuje do reguł, według których był szkolony. Jednak systemy te nie są w stanie reagować na wzorce wejściowe lub słowa kluczowe, które nie są zgodne z istniejącymi regułami. Jednym z takich języków jest AIML (Artificial Intelligence Markup Language). AIML został opracowany przez Alice Bot Free Software Community i Dr. Richarda S. Wallace w latach 1995-2000. AIML jest używany do tworzenia lub dostosowywania Alice Bot, który jest aplikacją chat-box opartą na A. L. I. C. E. (Artificial Linguistic Internet Computer Entity) wolnego oprogramowania. Co więcej, jest to język znaczników oparty na XML oraz tagach. Tagi to identyfikatory odpowiedzialne za tworzenie fragmentów kodu i wstawianie poleceń do chatbota. AIML definiuje klasę obiektów danych o nazwie AIM objects, która jest odpowiedzialna za modelowanie wzorców komunikacji[10].

4.2 Chatboty AI oparte na wyszukiwaniu

Podczas wprowadzania danych przez użytkownika, system wykorzystuje heurystykę do wyszukiwania najlepszej odpowiedzi z bazy danych predefiniowanych odpowiedzi. Wybór dialogu jest zasadniczo problemem predykcji, a wykorzystanie heurystyki do określenia najbardziej odpowiedniego wzorca odpowiedzi może obejmować proste algorytmy, takie jak dopasowanie słów kluczowych. Może również wymagać bardziej złożonego przetwarzania, poprzez uczenie maszynowe lub głębokie uczenie się. Niezależnie od techniki, te chatboty dostarczają tylko predefiniowane odpowiedzi i nie generują nowego wyniku.

Przy dużej ilości dostępnych danych, tworzenie systemu dialogowego opartego na wyszukiwaniu jest intuicyjne, ponieważ metody wyszukiwania informacji szybko rozwijają się. Biorąc pod uwagę wypowiedź wprowadzoną przez użytkownika jako zapytanie, system wyszukuje odpowiedzi kandydatów według odpowiednich metryk.

[10] Maria das Graças Bruno Marietto, *Artificial Intelligence MArkup Language: A Brief Tutorial* In: CoRR.2013.Vol. abs/1307.3091. https://arxiv.org/abs/1307.3091 (dostęp 26.06.2022 r.).

Rdzeń systemów konwersacyjnych opartych na wyszukiwaniu jest sformułowany jako problem dopasowania między wyrażeniem zapytania a odpowiedziami kandydatów. Typowym sposobem dopasowania jest zmierzenie wewnętrznego iloczynu dwóch reprezentujących wektorów cech dla zapytań i odpowiedzi kandydujących w przekształconej przestrzeni Hilberta. Wysiłki związane z modelowaniem sprowadzają się do znalezienia dopasowania oryginalnych danych wejściowych do wektorów cech, co znane jest jako uczenie się reprezentacji. Istnieje dwuetapowa metoda wyszukiwania, która pozwala znaleźć odpowiednią odpowiedź z ogromnego repozytorium danych. Proces wyszukiwania składa się z szybkiego rankingu za pomocą standardowego pomiaru TF-IDF i procesu ponownego rankingu za pomocą funkcji orientowanych na konwersację zaprojektowaną z ludzką wiedzą. Systemy przyczyniają się do wyboru najbardziej odpowiedniej odpowiedzi na zapytanie z par pytanie-odpowiedź przy użyciu statystycznego modelu językowego jako międzyjęzykowego wyszukiwania informacji (cross-lingual information retrieval (CLIR)[11]. Metody te opierają się na płytkich reprezentacjach, które zasadniczo wykorzystują reprezentacje słów typu „hot-one". Większość zaufanych wyszukiwarek uczy się reprezentacji za pomocą głębokich sieci neuronowych (DNN).

DNN to wysoce zautomatyzowane maszyny uczące się. Mogą one automatycznie wyodrębniać podstawowe abstrakcyjne cechy danych, badając wiele nieliniowych poziomów transformacji. Dominujące DNN do modelowania na poziomie zdania obejmują konwolucyjne sieci neuronowe (CNN) i rekurencyjne sieci neuronowe (RNN). Wiele metod dopasowywania można zastosować do krótkich dialogów tekstowych dla systemów opartych na wyszukiwaniu. Zasadniczo metody te modelują zdania za pomocą sieci konwolucyjnych lub rekurencyjnych do konstruowania abstrakcyjnych reprezentacji. Chociaż nie wszystkie z tych metod są pierwotnie przeznaczone do konwersacji, są one skuteczne w zadaniach dopasowywania krótkiego tekstu i są zawarte jako silne linie bazowe dla badań konwersacyjnych opartych na wyszukiwaniu[12].

Jednym z przykładów chatbota opartego na wyszukiwaniu jest Mitsuko. Zawiera on ponad 300000 predefiniowanych szablonów odpowiedzi i bazę wiedzy składającą się z ponad 3000 obiektów. Ten chatbot może przykładowo tworzyć piosenki i wiersze na jej podstawie.

[11] *Cross-language information retrieval* https://en.wikipedia.org/wiki/Cross-language_information_retrieval (dostęp 26.06.2022 r.).

[12] Ahtsham Manzoor, Dietmar Jannachhttps, *Towards retrieval-based conversational recommendation* https://doi.org/10.1016/j.is.2022.102083 (dostęp 26.06.2022 r.).

4.3 Generatywny AI chatbot

Chatbot modelu generatywnego nie używa żadnego predefiniowanego repozytorium. Ten rodzaj chatbota jest bardziej zaawansowany, ponieważ uczy się od zera za pomocą procesu zwanego „głębokim uczeniem się". Modele generatywne są zwykle oparte na technikach tłumaczenia maszynowego, ale zamiast tłumaczyć z jednego języka na inny, tłumaczenie odbywa się z danych wejściowych na dane wyjściowe (odpowiedź)[13].

Innym sposobem budowania systemu konwersacyjnego jest użycie technik generowania języka. Możemy łączyć generowanie wzorców językowych z metodami opartymi na wyszukiwaniu. Dzięki zastosowanym technikom głębokiego uczenia, systemy oparte na generacji są znacznie zaawansowane.

Mamy strukturę Sequence to Sequence (seq2seq), która pojawiła się w dziedzinie neuronowego tłumaczenia maszynowego i została z powodzeniem dostosowana do problemów dialogowych. Architektura składa się z dwóch rekurencyjnych sieci neuronowych (RNN) z różnymi zestawami parametrów. Podejście to obejmuje dwie sieci rekurencyjne, z których jedna koduje oryginalną sekwencję, i jest nazywana koderem, a druga dekoduje wyjście pierwszej sieci (zakodowaną sekwencję oryginalną) do sekwencji docelowej, i jest nazywana dekoderem. Struktura została pierwotnie opracowana dla problemów z tłumaczeniem maszynowym, chociaż okazała się skuteczna w powiązanych zadaniach przewidywania sekwencji do sekwencji, takich jak sumowanie tekstu i odpowiadanie na pytania[14].

4.4 Zespół systemów dialogowych opartych na wyszukiwaniu i generowaniu

Zazwyczaj rekurencyjna sieć neuronowa (RNN), przechwytuje semantykę zapytania za pomocą jednego lub kilku rozproszonych wektorów o rzeczywistej wartości (znanych również jako embedding); inny RNN dekoduje osadzanie zapytania na odpowiedź. Głębokie sieci neuronowe umożliwiają złożoną interakcję poprzez wiele nieliniowych transformacji; RNN są ponadto odpowiednie do modelowania danych szeregów czasowych (np. sekwencji słów), zwłaszcza gdy są wzmocnione pamięcią długo-krótkotrwałą (LSTM) lub gated recurrent unit (GRUs). Pomimo tego, RNN ma również swoją własną słabość, gdy stosuje się ją do systemów dialogowych: generowane zdanie

[13] Nikki Singh, Dr. Sachin Bojewa, *Generative Dialogue System using Neural Network,* 2019 JETIR May 2019, Volume 6, Issue 5, https://www.jetir.org/papers/JETIRCS06034.pdf.

[14] Janson Brownlee, *Deep Learning For Natural Language Processing* Machine Learning Mastery, 2017, s. 250-255.

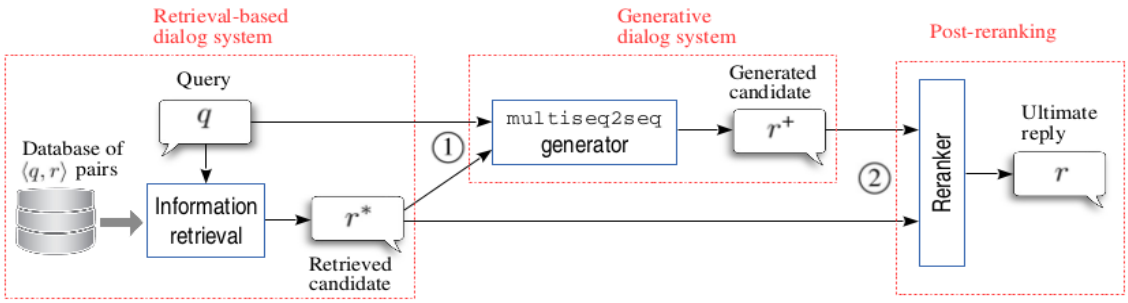
jest zwykle krótkie, uniwersalne i bez znaczenia, na przykład „nie wiem”. Jest tak prawdopodobnie dlatego, ponieważ dialogi, takie jak jak prowadzą chatboty, są bardzo zróżnicowane, a żądanie może nie zawierać wystarczających informacji, aby na nie odpowiedzieć. Pomimo faktu, że takie uniwersalne wypowiedzi mogą być odpowiednie w określonym kontekście dialogu, użytkownik staje się znudzony i traci zainteresowanie, więc takie odpowiedzi są niepożądane w rzeczywistych aplikacjach.

Jak można budować w zespole wyszukiwarek i generatywnych systemów dialogowych? Biorąc pod uwagę żądanie wydane przez użytkownika, najpierw otrzymujemy propozycję odpowiedzi, pobierając informacje z dużej bazy danych. Zapytanie wraz z propozycją odpowiedzi jest następnie przekazywane do generatora wypowiedzi opartego na modelu *bi-sequence to sequence* (biseq2seq). Taki generator sekwencji uwzględnia informacje zawarte nie tylko w żądaniu, ale także w otrzymanej odpowiedzi; w związku z tym eliminuje problem niskiej zawartości i może generować odpowiedzi, które są bardziej znaczące. Następnie ponownie używamy scorera w systemie wyszukiwania do ponownej analizy. Ten krok może odfiltrować mniej istotne pobrane odpowiedzi lub bezsensownie wygenerowane. Kandydat o wyższym rankingu (pobrany lub wygenerowany) jest zwracany użytkownikowi jako odpowiedź. Zasadniczo systemy wyszukiwania i generatywne są połączone dwoma mechanizmami:

(1) pobrany kandydat jest podawany do generatora sekwencji w celu złagodzenia problemu „niskiej zawartości”;

(2) ponowna analiza może lepiej wykorzystać zarówno pobrany kandydat, jak i wygenerowaną wypowiedź[15].

Poniższy rysunek przedstawia ogólną strukturę takiego zespołu systemów dialogowych.



Rysunek 4.1 - struktura zespołu systemów dialogowych opartych na wyszukiwaniu i generowaniu[16]

[15] Yiping Song, Rui Yan, Xiang Li, Dongyan Zhao, Ming Zhang, Two are Better than One: An Ensemble of Retrieval- and Generation-Based Dialog Systems

[16] Yiping Song, Rui Yan, Xiang Li, Dongyan Zhao, Ming Zhang, Two are Better than One: An Ensemble of Retrieval- and Generation-Based Dialog Systems

5. Rola NLP w konwersacyjnej sztucznej inteligencji

Przetwarzanie języka naturalnego lub NLP jest warunkiem wstępnym dla chatbota AI. NLP umożliwia komputerom i algorytmom zrozumienie interakcji międzyludzkich w różnych językach. Aby przetworzyć dużą ilość danych w języku naturalnym, AI z pewnością będzie potrzebować NLP lub przetwarzania języka naturalnego. Przetwarzanie języka naturalnego pozwala podzielić dowolną objętość tekstu na zdania i słowa, a następnie przetworzyć go i sklasyfikować. Oto niektóre z podstawowych zadań, które NLP może wykonać:

- normalizacja: proces konwertujący listę słów na jednolite sekwencje, takie jak małe litery,
- tokenizacja i podział zdań: zadanie dzielenia tekstu na jego części składowe, na przykład słowa lub zdania,
- korekta pisowni i tolerancja na błędy ortograficzne: proces poprawiania błędów ortograficznych lub akceptowania literówek, jednocześnie rozumiejąc, jakie słowo było zamierzone,
- part of Speech (POS) tagowanie i analiza morfologiczna: definiowanie atrybutów każdego słowa, na przykład, czy słowo jest rzeczownikiem czy czasownikiem, a następnie tagowanie go w przyszłości[17],
- wykrywanie języka: identyfikacja języka, który jest używany,
- wykrywanie nastrojów i intensywności: rozpoznawanie słów, które wskazują ton i/lub intensywność w tekście.

Konwersacyjna sztuczna inteligencja i NLP pozwalają chatbotowi komunikować się z użytkownikami końcowymi w sposób odzwierciedlający ludzką rozmowę. Istnieje wiele różnych komponentów, które nakładają się na siebie i rozszerzają możliwości NLP. Podstawowymi elementami systemów dialogowych są:

- moduł rozumienia języka naturalnego (NLU);
- menedżer dialogu (DM);
- moduł generowania języka naturalnego (NLG)[18].

[17] *Categorizing and Tagging Words*, https://www.nltk.org/book/ch05.html (dostęp 26.06.2022 r.)
[18] Xiaojie Wang, Caixia Yuan, *Recent Advances on Human-Computer Dialogue*, December 2016

5.1 Moduł rozumienia języka naturalnego (NLU)

Jednostka NLU odpowiada za przekształcenie wypowiedzi użytkownika w predefiniowaną strukturę semantyczną zgodnie z konwencjami systemu, czyli w format zrozumiały dla systemu. Obejmuje to zadanie wypełniania slotów i wykrywania intencji[19] Na przykład intencją może być powitanie, takie jak „Hello", „Hi", „Hey", lub może mieć ono charakter informacyjny, na przykład „I like pizza", gdzie użytkownik podaje dodatkowe informacje. W zależności od zainteresowań, sloty mogą być bardzo zróżnicowane, na przykład imię aktora, cena, czas rozpoczęcia, miasto docelowe, itp. Jak widać, intencje i sloty określają zamknięty charakter chatbota. Zadanie wypełniania slotów i wykrywania intencji jest postrzegane jako problem tagowania sekwencji. Z tego powodu komponent NLU jest zwykle implementowany jako rekurencyjna sieć neuronowa oparta na LSTM z warstwą warunkowego pola losowego (Conditional Random Field (CRF)) na wierzchu. Model opisany w pracy „Multi-domain joint semantic frame parsing using bi directional rnn-lstm"[20] pozostaje modelem Sequence to Sequence, wykorzystującym dwukierunkową sieć LSTM, która wypełnia sloty i przewiduje intencję w tym samym czasie. Z drugiej strony, model [Liu, B. and Lane, I.][21] robi to samo za pomocą RNN opartego na uwadze. Aby osiągnąć takie zadanie, etykiety zestawu danych składają się z: połączonych tegow slotów B–I–O (Begin, Inside, Outside), znacznika intent i dodatkowego znacznika end-of-string (EOS).

5.2 Moduł generowania języka naturalnego (NLG)

Generowanie języka naturalnego (NLG) to proces generowania tekstu z reprezentacji semantycznej. Można ją uznać za odwrotność rozumienia języka naturalnego (NLU). Systemy NLG zapewniają kluczową rolę w podsumowaniu tekstu, tłumaczeniu maszynowym i systemach dialogowych. W NLG odpowiedź systemowa jako frame semantyczny jest mapowana z powrotem do zdania w języku naturalnym, zrozumiałego dla użytkownika końcowego. Komponent NLG może być oparty na regułach lub

[19] H. Weld, X. Huang, S. Long, J. Poon, S. C. *Han: A survey of joint intent detection and slot-filling models in natural language understanding,* CoRR.2021.Vol. abs/2101.08091, https://arxiv.org/abs/2101.08091

[20] Dilek Hakkani-Tur, Gokhan Tur, Asli Celikyilmaz, Yun-Nung Chen,Jianfeng Gao, Li Deng, Ye-Yi Wang, *Multi-Domain Joint Semantic Frame Parsing using Bi-directional RNN-LSTM*

[21] Bing Liu, Ian Lane *Attention-Based Recurrent Neural Network Models for Joint Intent Detection and Slot Filling,* CoRR.2016.Vol. abs/1609.01454, https://arxiv.org/abs/1609.01454

,

modelach. W niektórych scenariuszach może to być model hybrydowy, czyli połączenie obu.

Oparty na regułach NLG wyprowadza niektóre predefiniowane zdania szablonów dla danego framu semantycznego, więc są one bardzo ograniczone i nie mają żadnej zdolności generalizacji. Chociaż opracowano kilka systemów generowania opartych na regułach ogólnego przeznaczenia, często są one dość trudne do dostosowania do małych aplikacji zorientowanych na zadania, ze względu na ich ogólność[22].

Systemy NLG oparte na uczeniu maszynowym (trainable) są bardziej powszechne w dzisiejszych systemach dialogowych. NLG wykorzystują kilka źródeł danych wejściowych, takich jak: plan treści, przekazujący znaczenie reprezentacji tego, jak komunikować się z użytkownikiem, baza wiedzy, ustrukturyzowana baza danych zawierająca jednostki specyficzne dla domeny, model użytkownika, model, który nakłada ograniczenia na wypowiedzi wyjściowe, historia dialogów, informacje z poprzednich zwrotów, aby uniknąć powtórzeń, odesłanie wyrażeń, itp.

Wyszkolone systemy NLG mogą tworzyć różne wypowiedzi kandydatów (takie jak scholastyczne lub oparte na regułach) i wykorzystywać model statystyczny do ich uszeregowania. Model statystyczny przypisuje wyniki do każdej wypowiedzi i jest szkolony na podstawie danych tekstowych. Większość z tych systemów wykorzystuje modele językowe bigramów i trygramów do generowania wypowiedzi[23].

Z drugiej strony, w NLG opartym na semantycznie sterowanej sieci rekurencyjnej z pamięcią długo-krótkoterminową (LSTM), może on uczyć się na podstawie niezaliczonych danych, wspólnie optymalizując komponenty planowania zadań i realizacji powierzchniowej przy użyciu prostego kryterium treningu entropii krzyżowej bez żadnych heurystyk, a dobrej jakości zmienność języka uzyskuje się po prostu poprzez losowe pobieranie próbek z wyjść sieci[24].

5.3 Menedżer dialogu (DM)

DM może być podłączony do jakiejś zewnętrznej bazy wiedzy lub bazy danych, dzięki czemu może generować bardziej znaczące odpowiedzi. Menedżer dialogów składa się z

[22] Ehud Reiter, *An architecture for data-to-text systems*, https://dl.acm.org/doi/10.5555/1610163.1610180 (dostęp 26.06.2022 r.).

[23] Ankit Arun , Soumya Batra, Vikas Bhardwaj, Ashwini Challa, Pinar Donmez, Peyman Heidari, Hakan Inan, Shashank Jain, Anuj Kumar, Shawn Mei, Karthik Mohan, Michael White: *Best Practices for Data-Efficient Modeling in NLG: How to Train Production-Ready Neural Models with Less Data*, 2020.

[24] Tsung-Hsien Wen, Milica Gasic, Nikola Mrkicc, Pei-Hao Su, David Vandyke and Steve Young: *Semantically Conditioned LSTM-based Natural Language Generation forSpoken Dialogue SystemsTsung*, 2015, s. 1711–1721, Lisbon, Portugal. Association for Computational Linguistics.

dwóch następujących elementów: narzędzia do śledzenia stanu dialogów (Dialogue State Tracker) i narzędzia do nauki zasad (Policy Learning), które pozostaje agentem uczenia się wzmocnionego. Tracker stanu dialogu jest złożonym i niezbędnym elementem, który powinien prawidłowo wnioskować o stanie dialogu, biorąc pod uwagę całą historię. Policy Learning odpowiada za wybór najlepszego działania, czyli reakcji systemu na wypowiedź użytkownika, które powinno prowadzić użytkownika do osiągnięcia celu w minimalnej liczbie obrotów dialogowych.

5.4 Identyfikacja intencji i ekstrakcja informacji

Identyfikacja intencji jest istotnym elementem każdego systemu konwersacyjnego, który koncentruje się na wykonywaniu zadań. Aby zrozumieć obecny cel użytkownika, system musi użyć swojego detektora intencji w celach sklasyfikowania wypowiedzi użytkownika (reprezentowaną w innym języku naturalnym) do jednej z kilku predefiniowanych klas. Wraz z rozwojem głębokiego uczenia się coraz więcej uczonych stosuje embedding słów, konwolucyjne sieci neuronowe (CNN), rekurencyjne sieci neuronowe (RNN), sieć długiej krótkoterminowej pamięci (LSTM), Gated Recurrent Unit (GRU), mechanizm uwagi i Capsule Networks, do intencji wykrywania zadań. W porównaniu z tradycyjnymi metodami uczenia maszynowego, model uczenia głębokiego działa poprawniej pod względem wydajności wykrywania[25].

Jeśli chodzi o ekstrakcję informacji, podstawową odpowiedzialnością NLU jest nie tylko zrozumienie znaczenia frazy, ale także zrozumienie znaczenia samego tekstu. Aby wydobyć znaczenie z tekstu, przekształcamy go na nieustrukturyzowany, napisany do chatbota — w ustrukturyzowane gramatyczne obiekty danych, które będą dalej przetwarzane przez menedżera dialogów. Pierwszym krokiem w tym procesie jest rozbicie zdania na tokeny, które reprezentują każdą z jego części składowych: słowa, znaki interpunkcyjne, liczby, itp. Tokenizacja jest trudna ze względu na częstotliwość niejednoznacznych lub zniekształconych danych wejściowych, w tym: frazy, skróty, akronimy i spacje. Tokeny te można analizować za pomocą szeregu metod opisanych poniżej, w celu utworzenia szeregu różnych struktur danych, które będą obsługiwane przez menedżera dialogów.

Istnieje kilka podejść, które można wykorzystać do wydobywania informacji, tak jak pokazano poniżej:

[25] Jiao Liu, Yanling Li, Min Lin, *Review of Intent Detection Methods in the Human-Machine Dialogue System*, Inner Mongolia Normal University, 2019

Bag of Words: w tym podejściu ignorujemy strukturę zdania, kolejność i składnię oraz liczymy liczbę wystąpień każdego słowa. Używamy tego do utworzenia modelu przestrzeni wektorowej, w którym stop-słowa są usuwane, a warianty morfologiczne przechodzą proces zwany lemmatyzacją i są przechowywane jako instancje lematu podstawowego. W fazie menedżera dialogów, zakładając, że bot działa w oparciu o reguły, te wynikowe słowa zostaną zmapowane na dokumenty, przechowywane w bazie danych wiedzy bota, aby znaleźć dokumenty z danymi wejściowymi zawierającymi podobne słowa kluczowe. Podejście wydaje się być proste, ponieważ nie wymaga znajomości składni, ale z jakiegoś powodu nie jest ono wystarczająco dokładne, aby poradzić sobie z bardziej złożonymi problemami.

Latent Semantic Analysis: takie podejście jest podobne do Bag of Words. Znaczenia / pojęcia, a nie słowa, są jednak podstawową jednostką porównania, analizowaną z danego zdania lub wypowiedzi. Po drugie, grupy słów, które często występują razem, są zgrupowane razem. W LSA tworzymy macierz, w której każdy wiersz reprezentuje unikalne słowo, każda kolumna reprezentuje dokument, a wartość każdej komórki jest częstotliwością słowa w dokumencie. Obliczamy odległość między wektorem reprezentującym każdą wypowiedź i dokument, używając rozkładu wartości pojedynczej w celu zmniejszenia wymiarowości macierzy, i wyznaczamy najbliższy dokument.

Wyrażenia regularne: zdania, wypowiedzi mogą być traktowane jako wyrażenia regularne i mogą być porównywane z dokumentami w bazie wiedzy bota.

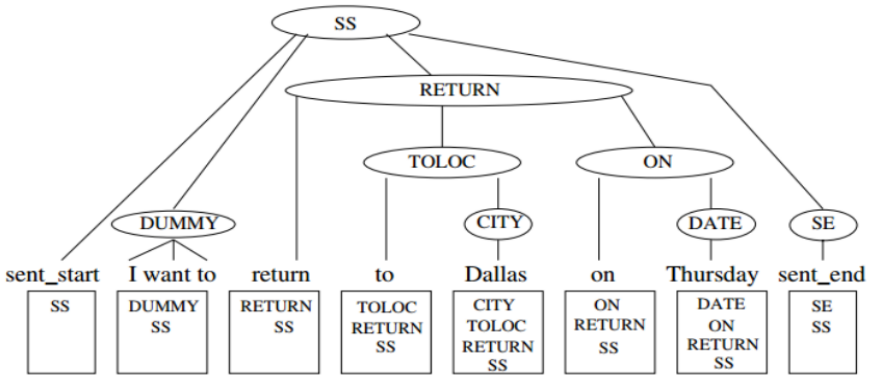
Part of Speech (POS) Tagging: tagowanie POS oznacza każde słowo w ciągu wejściowym jego częścią mowy (na przykład rzeczownikiem, czasownikiem, przymiotnikiem i inne). Tagi te mogą być oparte na regułach (ręcznie tworzony zestaw reguł został zrobiony w celu określenia części mowy dla niejednoznacznych słów, biorąc pod uwagę ich kontekst). Można je również tworzyć przy użyciu modeli stochastycznych, które są szkolone na zdaniach oznaczonych prawidłowymi POS. W Menedżerze dialogów POS można wykorzystać do przechowywania odpowiednich informacji w historii dialogów. POS jest również używany podczas generowania odpowiedzi w celu określenia typu obiektu POS pożądanej odpowiedzi.

Named/Relation Entity Recognition: w rozpoznawaniu jednostek nazwanych (NER) nazwy osób, miejsc, grup i lokalizacji są wyodrębniane i odpowiednio oznaczone. Pary NER-name mogą być przechowywane przez menedżera dialogu w historii dialogu, aby śledzić kontekst rozmowy bota. Ekstrakcja relacji idzie o krok dalej do relacji tożsamości (np. „kto co komu zrobił") i oznacza każde słowo w tych frazach.

Semantic Role Labelling: argumenty czasownika są oznaczone na podstawie ich roli semantycznej (na przykład podmiot, temat, itp.). W tym procesie orzecznik jest najpierw oznaczony, a następnie jego argumenty. Znane klasyfikatory do oznaczania ról semantycznych zostały przeszkolone w FrameNet i PropBank, bazach danych, w których zdania są już oznaczone ich rolami semantycznymi. Te semantyczne pary rola-słowo mogą być przechowywane przez menedżera dialogów w historii dialogów w celu śledzenia kontekstu.

Tworzenie gramatycznych struktur danych: zdania i wypowiedzi mogą być przechowywane w uporządkowany sposób w formalizmie gramatycznym, takim jak gramatyki bez kontekstu (CFGs) i gramatyki zależności (DGs). Gramatyki bez kontekstu to podobne do drzewa struktury danych, które reprezentują zdania jako zawierające wyrażenia rzeczownikowe i wyrażenia czasownikowe, z których każda zawiera rzeczowniki, czasowniki, podmioty i inne konstrukcje gramatyczne. Gramatyka zależności natomiast skupia się na relacji między słowami.

Hidden Vector State (HVS) Model: Celem statystycznych modeli stanu wektora ukrytego jest automatyczne wytworzenie pewnych dokładnych, ustrukturyzowanych znaczeń. Rozważmy przykład jako „I want to return to Dallas on Thursday". Poniższe drzewo parsowania jest jednym ze sposobów reprezentowania ustrukturyzowanej wartości zdania. SS reprezentuje węzeł początkowy send_start, a SE reprezentuje węzeł końcowy send_end. Traktujemy każdy węzeł końcowy jako stan wektorowy opisany przez jego węzły macierzyste: Stan wektorowy Dallas - [CITY, TOLOC, RETURN, SS].



Rysunek 5.1 - Przykład drzewa parsowania i jego odpowiednika stanu wektorowego[26]

Całe drzewo parsowania można następnie przedstawić jako sekwencję stanów wektorowych reprezentowanych przez powyższą sekwencję kwadratów. Jeśli każdy stan

[26] Yulan He, Steve Young, *Semantic processing using the Hidden Vector State model*, https://www.sciencedirect.com/science/article/abs/pii/S0885230804000117 (dostęp 26.06.2022 r.)

wektorowy jest postrzegany jako zmienna ukryta, wówczas sekwencja stanów wektorowych (na przykład kwadraty powyżej) może być postrzegana jako ukryty model Markowa: zaczynamy od SS i mamy pewne prawdopodobieństwa osiągnięcia szeregu możliwych stanów ukrytych jako następnego stanu. Każdy stan wektorowy może być postrzegany jako „automat push-down".

Support Vector Machine (SVM): Support Vector Machines są nadzorowanym narzędziem uczenia maszynowego. Biorąc pod uwagę zestaw oznakowanych danych treningowych, algorytm generuje optymalny hyperplane, który dzieli próbkę na odpowiednie etykiety. Tradycyjnie maszyny SVM są uważane za rozwiązanie problemów z klasyfikacją binarną, jednak wiele hiperpłaszczyzn można wykorzystać do podziału danych na więcej niż dwie kategorie etykiet. Optymalna hiperpłaszczyzna jest zdefiniowana jako hiperpłaszczyzna, która tworzy maksymalny margines lub odległość między różnymi zestawami punktów danych.

Conditional Random Field Models: CRF są logarytmicznie liniowymi modelami statystycznymi często stosowanymi do prognozowania strukturalnego. W przeciwieństwie do uśrednionego klasyfikatora, który przewiduje etykietę dla pojedynczego obiektu i ignoruje kontekst, CRF uwzględniają poprzednie cechy sekwencji wejściowej za pomocą prawdopodobieństw warunkowych. Do szkolenia modelu można użyć wielu różnych funkcji, w tym informacji leksykalnych, prefiksów i sufiksów, wielkich liter i innych funkcji.

Deep Learning: najnowszym postępem w stosowaniu modeli statystycznych do przewidywania struktury koncepcji jest głębokie uczenie dla przetwarzania języka naturalnego lub głębokie NLP. Architektury sieci neuronowych Deep learning różnią się od tradycyjnych sieci neuronowych faktem, iż wykorzystują więcej ukrytych warstw, a każda warstwa obsługuje coraz bardziej złożone funkcje. W rezultacie sieci mogą uczyć się na podstawie wzorców i nieoznakowanych danych, a uczenie głębokie może być wykorzystywane do uczenia się bez nadzoru. Metody Deep learning zostały wykorzystane do generowania znaczników POS zdań (fragment tekstu na frazę rzeczownikową, wyrażenia czasownikowe, itp.) oraz do rozpoznawania nazw i oznaczania ról semantycznych.

W przetwarzaniu języka naturalnego i uczeniu maszynowym nie ma jednego, optymalnego podejścia do wykonania zadania i dotyczy to również zadań związanych z wydobywaniem informacji.

6. Struktura modelu seq2seq

Modele Sequence to Sequence (seq2seq) to modele głębokiego uczenia się, które osiągnęły wielki sukces w takich zadaniach, jak tłumaczenie maszynowe, sumowanie tekstu, adnotacja obrazu itp. Na przykład pod koniec 2016 r. podobny model został wbudowany w Google Translate. Podstawy modeli seq2seq zostały ustanowione w 2014 roku wraz z wydaniem dwóch artykułów — [Sutskever et al.,2014][27] oraz [Cho et al., 2014][28].

Sequence to Sequence to model, który przyjmuje sekwencję elementów (słów, liter, cech obrazu itp.) do wejścia i zwraca inną sekwencję elementów. Składa się z enkodera i dekodera. Enkoder przetwarza każdy element sekwencji wejściowej, przekłada otrzymane informacje na wektor zwany kontekstem (context). Po przetworzeniu całej sekwencji wejściowej enkoder przekazuje kontekst do dekodera, który następnie zaczyna generować sekwencję wyjściową element po elemencie. Kontekst jest wektorem (tablicą liczb), a enkoder i dekoder są z kolei najczęściej rekurencyjnymi sieciami neuronowymi. Domyślnie w każdym przedziale czasowym RNN przyjmuje na wejście dwa elementy: bezpośrednio element wejściowy (w przypadku enkodera, jedno słowo z oryginalnego zdania) i stan ukryty (hidden state). Słowo musi jednak być reprezentowane przez wektor. Aby przekonwertować słowo na wektor, stosuje się szereg algorytmów zwanych „embeddingami słów" (word embeddings). Embeddingi odwzorowują słowa na przestrzenie wektorowe zawierające informacje semantyczne na ich temat.

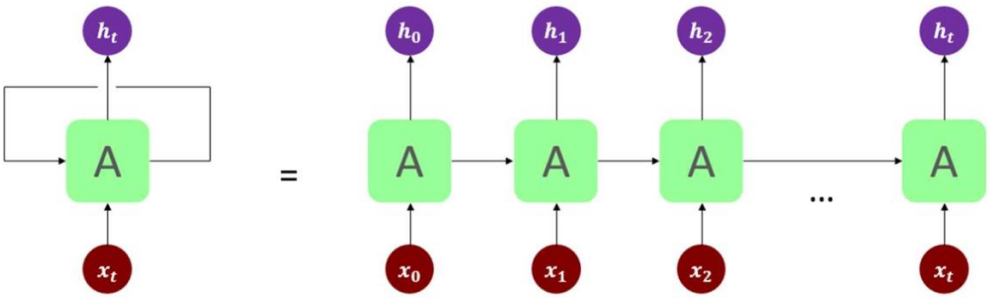
W poniższych rozdziałach przyjrzymy się bliżej projektom sieci neuronowych zastosowanym w tym modelu, a także konwersji słów na wektor (embedding).

6.1 Zasada działania sieci LSTM

Przyjrzyjmy się bliżej architekturze sieci neuronowej LSTM (Long Short Term Memory, długa pamięć krótkoterminowa). Sieci tego typu są podgatunkami RNN (Recurrent Neural Network, rekurencyjna sieć neuronowa).

Na początek powiedzmy, czym różnią się RNN od innych rodzajów architektur sieci neuronowych. Sieci neuronowe o najprostszej architekturze nie mają pamięci i trwałości, a przy tym nie mogą wykorzystywać informacji o decyzjach podjętych w poprzednich iteracjach ich pracy. Aby rozwiązać ten problem, opracowano rekurencyjne sieci neuronowe, które zawierają sprzężenia zwrotne i umożliwiają przesyłanie informacji z poprzedniego kroku sieci do bieżącego, jak pokazano na rysunku 6.1 - A jest fragmentem rekurencyjnej sieci neuronowej, x_t jest wartością wejściową, a h_t jest wartością zwracaną.

[27] Ilya Sutskever, Oriol Vinyals, Quoc V. Le, *Sequence to Sequence Learning with Neural Networks*, 2014.
[28] Kyunghyun Cho Bart van Merrienboer Caglar, Gulcehre Dzmitry Bahdanau, Fethi Bougares, Yoshua Bengio - *Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation*.



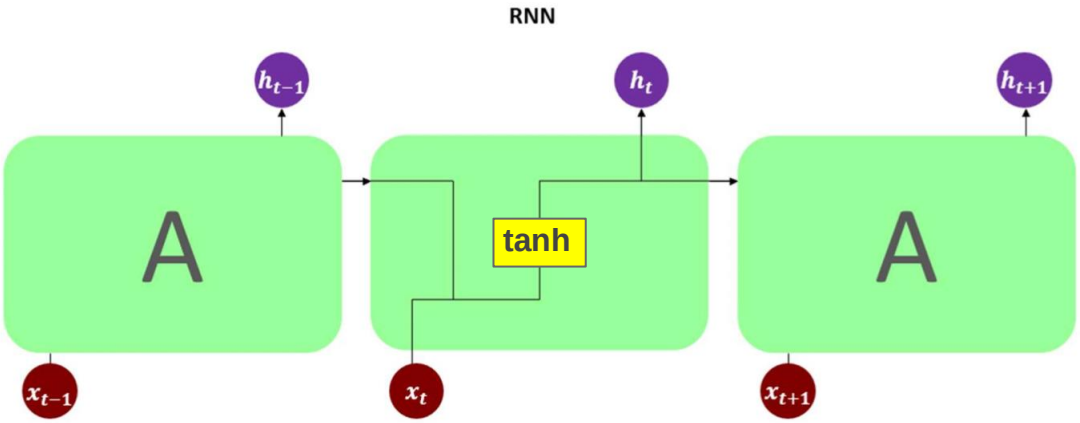
Rysunek 6.1 – Fragment rekurencyjnej sieci neuronowej[29]

Przejdźmy do modyfikacji rekurencyjnej sieci neuronowej – LSTM, zdolnej do uczenia się długotrwałych zależności. Głównym powodem pojawienia się tej modyfikacji był problem śledzenia kontekstu i powiązań między elementami sekwencji w miarę wzrostu odległości między tymi elementami. Na przykład, jeśli RNN zostanie zastosowana do problemu, w którym należy przewidzieć ostatnie słowo zdania, wówczas prognoza sieci tego typu zakończy się powodzeniem, jeśli odległość między miejscem, w którym należy przewidzieć słowo, a miejscem, w którym znajdowały się istotne informacje dla prawidłowej prognozy, jest niewielka. Innymi słowy, RNN mogą uwzględniać tylko najbliższy kontekst.

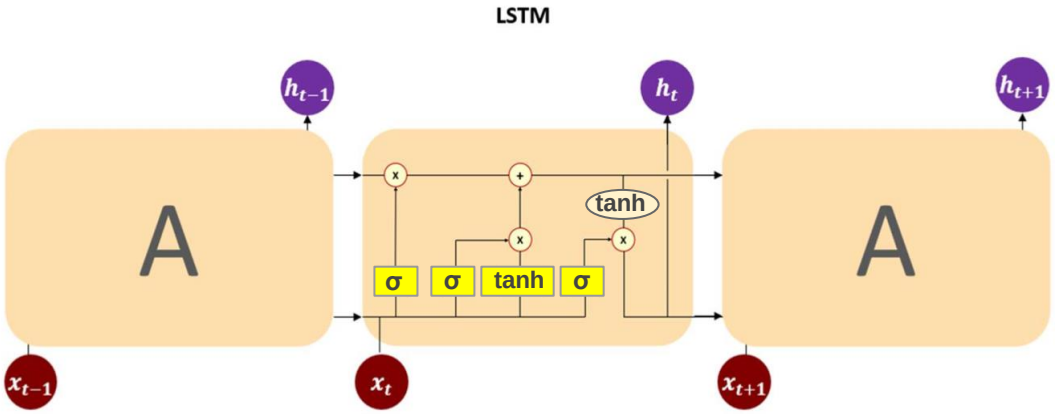
W przeciwieństwie do RNN, sieci neuronowe LSTM są stosowane do rozwiązywania problemów w warunkach, w których wśród ważnych zdarzeń możliwe są przerwy czasowe o nieokreślonym czasie trwania, przy czym LSTM ma przewagę nad alternatywnymi RNN ze względu na stosunkowo niewrażliwość na czas trwania podobnych przerw.

Podczas gdy powtarzający się moduł standardowego RNN składa się z jednej warstwy, na przykład niech będzie to warstwa z tangensem hiperbolicznym (tanh) jako funkcją aktywacji, jak pokazano na rysunku 6.2, powtarzający się moduł sieci LSTM ma inną strukturę, która ma cztery współdziałające warstwy w klasycznej reprezentacji, jak pokazano na rysunku 6.3.

[29] *Understanding LSTM Networks*, http://colah.github.io/posts/2015-08-Understanding-LSTMs/(dostęp 27.06.2022 r.).



Rysunek 6.2 - Struktura modułu powtarzalnego w standardowym RNN[30]



Rysunek 6.3 - struktura powtarzalnego modułu w LSTM

Rozważmy, jak zbudowana jest pojedyncza komórka LSTM, której struktura jest schematycznie przedstawiona na rysunku 6.4. Na tym schemacie każda linia przekazuje wektor niektórych wartości z węzła do węzła, linie łączone odpowiadają połączeniu wektorów, a linie rozgałęzione odpowiadają kopiowaniu wektora. Niech x_t będzie wektorem wartości wejściowych, h_t będzie wektorem wartości zwracanych, t będzie bieżącym krokiem sieci, t-1 będzie poprzednim krokiem sieci.

Kluczowym elementem architektury sieci tego typu jest jej stan C, reprezentowany jako wektor, który przechodzi przez łańcuch komórek sieci, uczestnicząc w kilku

[30] *Understanding LSTM Networks*, http://colah.github.io/posts/2015-08-Understanding-LSTMs/(dostęp 27.06.2022 r.).

przekształceniach liniowych i poddając się niewielkim zmianom. Niech C_{t-1} będzie stanem sieci przed wykonaniem transformacji w odpowiedniej komórce, C_t będzie stanem sieci po transformacjach wykonanych w komórce.

Proces usuwania informacji ze stanu komórki odbywa się za pomocą filtrów (bramek, gates), które składają się z warstwy sigmoidalnej i operacji mnożenia punktowego. Liczby od 0 do 1 pojawiające się na wyjściu filtrów określają, ile procentowej ilości każdej jednostki informacji zostanie dalej pominiętych. Rozważmy filtry i ich wpływ na stan w komórce LSTM.

Warstwa filtra utraty (brama zapominania) jest odpowiedzialna za decydowanie o tym, jakie informacje należy usunąć ze stanu komórki - może to być przydatne, jeśli na przykład w sekwencji, z którą działa sieć, pojawia się nowy element, który powoduje, że o wcześniejszym elemencie sieci można zapomnieć. W tym przypadku wektor f_t bramki zapominania zawierający wagę zapamiętywania starych informacji oblicza się według formuły (1), gdzie W_f jest macierzą parametrów, b_f jest wektorem parametrów.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \qquad (1)$$

Sigmoidalna warstwa filtra wejściowego (bramki wejściowej) ma na celu decydowanie, które nowe informacje zostaną zapisane w stanie komórki. Wektor bramki wejściowej i_t zawierający wagę otrzymywania nowych informacji jest obliczany za pomocą formuły (2), gdzie W_i jest macierzą parametrów, a b_i jest wektorem parametrów.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \qquad (2)$$

Warstwa $tanh$ jest przeznaczona do tego samego celu, co warstwa bramki wejściowej, ale w rezultacie tworzy wektor $\tilde{C}_t$ z nowymi kandydującymi wartościami do przechowywania w stanie komórki. Jest on obliczany według formuły (3), gdzie W_C jest macierzą parametrów, a b_C jest wektorem parametrów. Kolejno następuje aktualizacja stanu zgodnie z formułą (4).

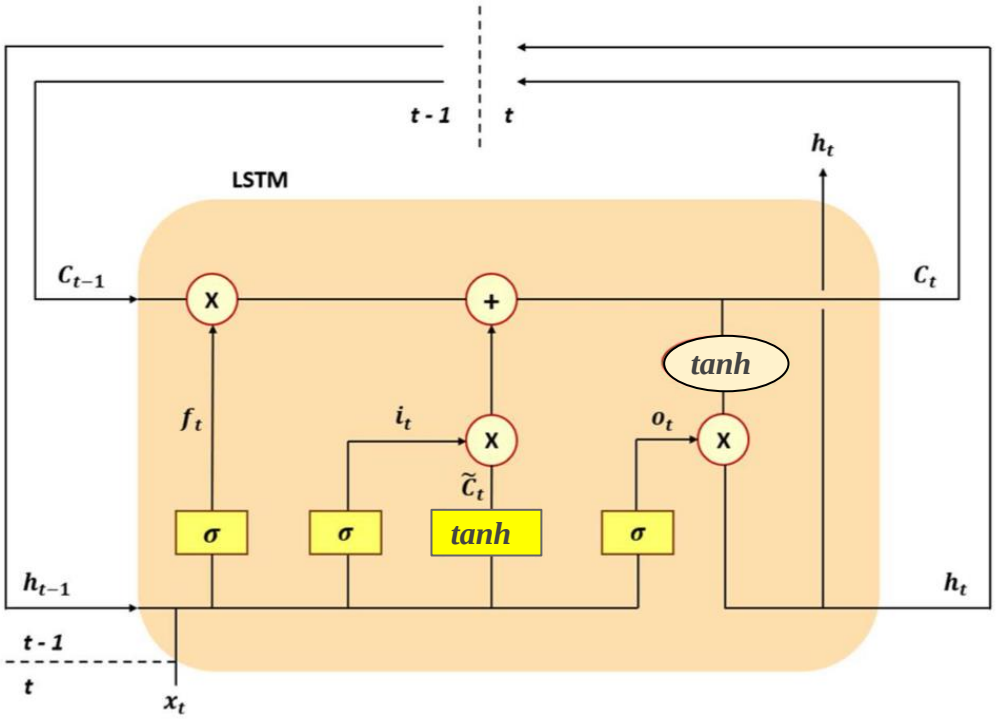
$$\tilde{C}_{t-1} = tanh(W_C[h_{t-1}, x_t] + b_C) \qquad (3)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \qquad (4)$$

Sigmoidalna warstwa filtra wyjściowego (bramki wyjściowej) jest odpowiedzialna za podejmowanie decyzji o tym, co trafi do wyjścia komórki sieci, biorąc pod uwagę jej stan. Również na tym etapie warstwa $tanh$ jest używana w celu doprowadzenia wartości do przedziału [-1, 1]. Według formuły (5) obliczany jest wektor bramki wyjściowej, gdzie W_o jest macierzą parametrów, b_o jest wektorem parametrów. Wektor wyjściowy h_t jest obliczany według wzoru (6).

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \qquad (5)$$

$$h_t = o_t * tanh(C_t) \qquad (6)$$



Rysunek 6.4 - Struktura komórki LSTM

Klasyczny wariant LSTM został po raz pierwszy zaproponowany w 1997 r. przez Seppa Hochreitera i Jürgena Schmidhubera[31]. Następnie pojawiły się różne modyfikacje architektury sieci tego typu, rozważmy niektóre z nich.

Istnieje modyfikacja LSTM, która pozwala warstwom bramkowym zobaczyć stan komórki sieci, schematyczny widok struktury takiej sieci przedstawiono na rysunku 6.5. Wektory f_t (brama zapominania), i_t (brama wejściowa) i o_t (brama wyjściowa) są

[31] Jason Brownlee, *Long Short-term Memory Networks with Python: Develop Sequence Prediction Models with Deep Learning*, 2017.

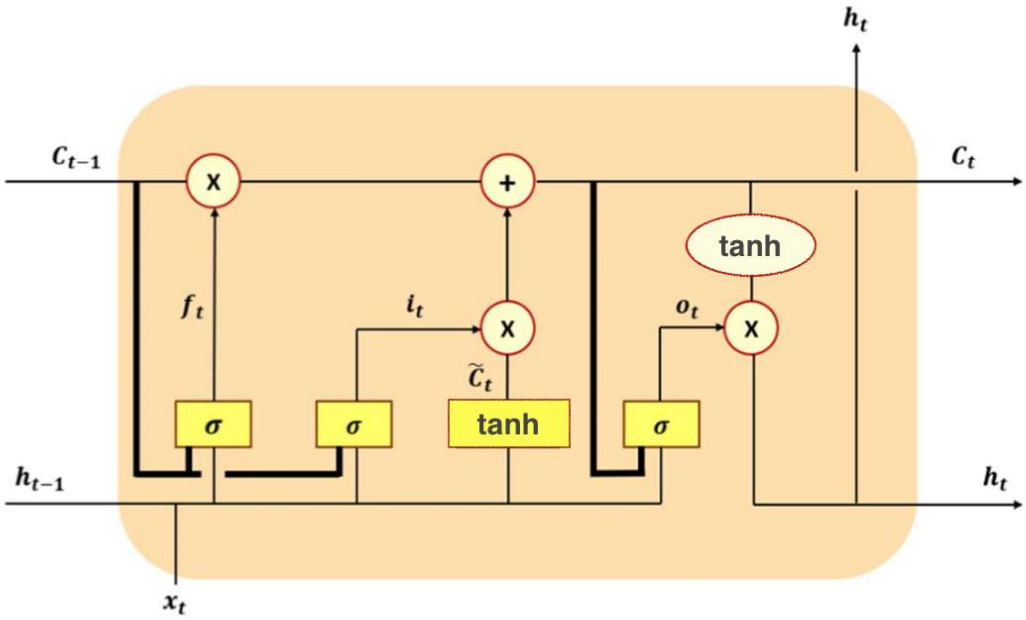
obliczane na podstawie wzorów (7), (8), (9).

$$f_t = \sigma(W_f[C_{t-1}, h_{t-1}, x_t] + b_f) \qquad (7)$$

$$i_t = \sigma(W_i\,[C_{t-1}, h_{t-1}, x_t] + b_i) \qquad (8)$$

$$f_o = \sigma(W_o[C_{t-1}, h_{t-1}, x_t] + b_o) \qquad (9)$$

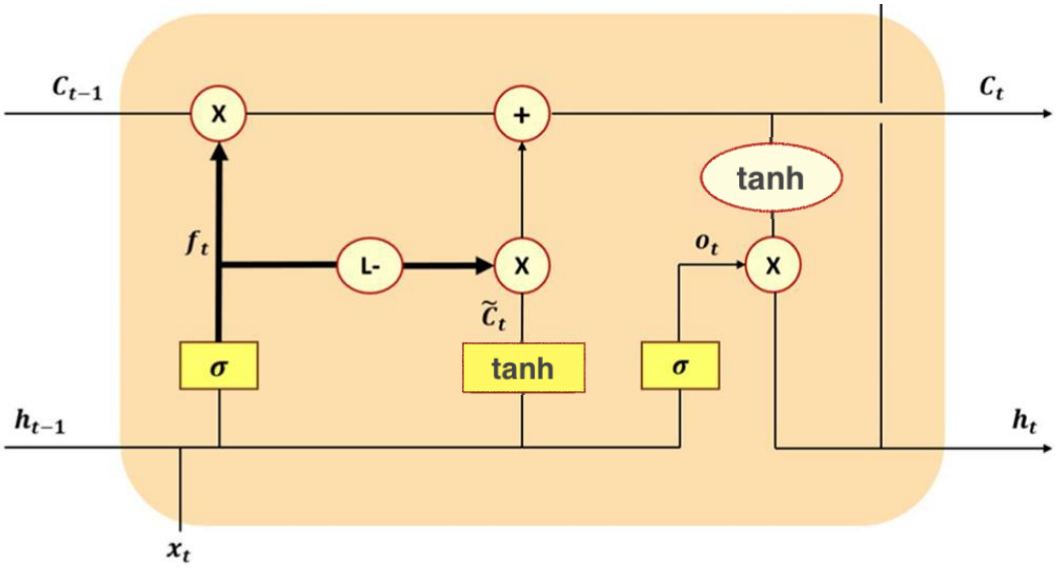
Jednocześnie istnieją różne odmiany tej modyfikacji, które pozwalają na możliwość przeglądania stanu przez niektóre warstwy bramek.



Rysunek 6.5-struktura komórki LSTM z możliwością wyświetlania stanu komórki przez warstwy bramek

Istnieje również rodzaj LSTM, w którym bramki utraty i wejścia są połączone w jedną bramę. Jest ona odpowiedzialna za jednoczesne podejmowanie decyzji zarówno o tym, jakie informacje można zapomnieć, tak i o aktualizowaniu stanu komórki, aby pasował on do nowych informacji. Innymi słowy, sieć zapomina o pewnych informacjach, gdy pojawia się inna, zastępująca poprzednią. Schemat komórki danej odmiany LSTM przedstawiono na rysunku 6.6. W takim przypadku zaktualizowany stan komórki C_t jest obliczany według formuły (10).

$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_{t-1} \qquad (10)$$



Rysunek 6.6-struktura komórki LSTM z połączonymi bramkami zapominania i wejścia

Również dość popularną modyfikacją LSTM jest kontrolowana jednostka rekurencyjna (GRU, Gated recurrent Unit). Schemat komórki przedstawiono na rysunku 6.7. W sieciach GRU Brama aktualizacji określa, jakie nowe informacje należy dodać do stanu komórki. Brama resetowania natomiast mówi o jakich informacjach sieciowych należy zapomnieć. Jednocześnie, GRU posiada inne różnice - przykładowo, dotyczą one stanu komórki i stanu ukrytego, które są łączone. GRU ma mniejszą liczbę parametrów w porównaniu do LSTM, co pozwala trenować sieci GRU nieco szybciej. Obliczenia odbywają się według formuł (11), (12), (13) i (14), gdzie z_t jest wektorem bramki aktualizacji, r_t jest wektorem bramki zrzutu, $\tilde{z}_t$ jest wektorem wyjściowym wartości kandydatów, h_t jest wektorem wyjściowym; W_z, W_r, W są macierzami parametrów[32]:

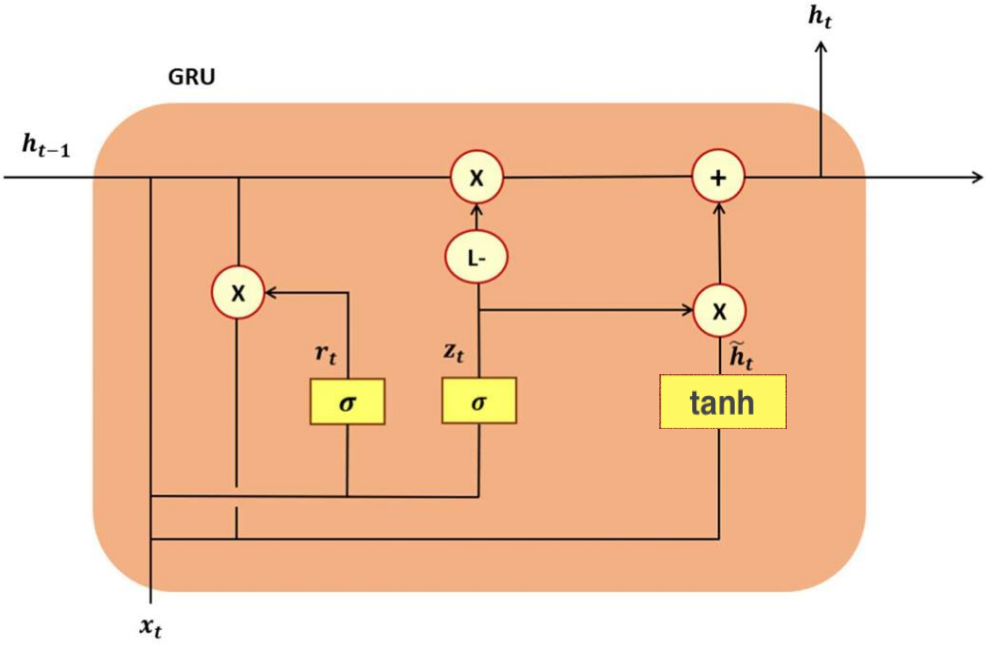
$$z_t = \sigma(W_z[h_{t-1}, x_t]) \qquad (11)$$

$$r_t = \sigma(W_r[h_{t-1}, x_t]) \qquad (12)$$

$$\tilde{h}_t = tanh(W[r_t * h_{t-1}, x_t]) \qquad (13)$$

[32] Jason Brownlee, *Making Predictions with Sequences*, https://machinelearningmastery.com/sequence-prediction/(dostęp 11.07.2022 r.)

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \qquad (14)$$



Rysunek 6.7-struktura komórki GRU

Istnieją różne metody odpowiednie do szkolenia sieci LSTM, przy czym oprócz metod nauczania sieci z nauczycielem możliwe jest zastosowanie metod uczenia się ze wzmocnieniem[33]. Jedną z najpopularniejszych metod nauczania LSTM z nauczycielem jest metoda propagacji wstecznej błędu wdrożona w czasie (BPTT, Backpropagation Through Time), dla której istnieją również pewne modyfikacje, takie jak skrócona propagacja wsteczna błędu w czasie (TBPTT, Truncated Backpropagation Through Time).

BPTT można krótko opisać jako sekwencję następujących kroków:

1) do wejścia sieci podawana jest sekwencja kroków czasowych, zawierających pary wartości wejściowych i wyjściowych;

2) sieć rozwija się, następuje obliczenie i nagromadzenie błędów dla każdego kroku czasowego;

3) sieć zwija się, następuje aktualizacja wag;

4) powtarzanie poprzednich kroków.

Podczas korzystania z BPTT zwiększa się ilość obliczeń. W celu rozwiązania tego problemu, zaproponowano TBPTT, którego główną różnicą jest to, że jeden krok czasowy

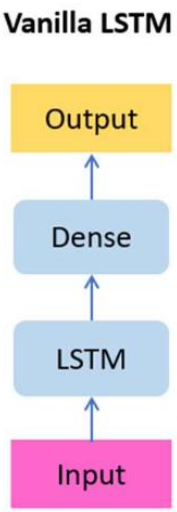
[33] Sepp Hochreiter and Jürgen Schmidhuber. *Long short-term memory. Neural computation*, 9(8):1735–1780, 1997.

jest przetwarzany na raz, co zdarza się okresowo. Liczba takich kroków jest oznaczona jako k1, a obliczenia i akumulacja błędów są wykonywane dla pewnej stałej liczby kroków czasowych, która jest oznaczona jako k2. Przy tym wartość k2 musi być wystarczająco duża, aby sieć mogła się poprawnie uczyć, a wartość k1 jest odpowiedzialna za szybkość uczenia się sieci[34].

6.2 Architektury sieci LSTM

Rozważmy niektóre często stosowane architektury sieci LSTM. Najprostsza jest tak zwana Vanilla LSTM, która odnosi się do architektury przedstawionej na rysunku 6.8 i opisanej w artykule Zeppa Hochreitera i Jürgena Schmidhubera z 1997 roku. Vanilla LSTM jest w stanie poradzić sobie z małymi zadaniami, związanymi z przewidywaniem sekwencji i składa się z:

1) warstwa wejściowa,
2) w pełni połączonej ukrytej warstwy LSTM,
3) w pełni połączonej warstwy wyjściowej konwencjonalnej sieci neuronowej.

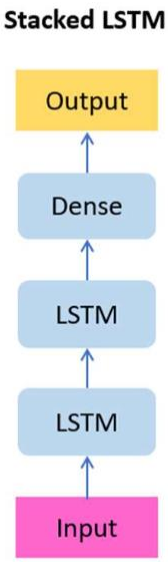


Rysunek 6.8-Architektura Vanilla LSTM

Następną często stosowaną architekturą jest Stacked LSTM, która ma wiele ukrytych warstw LSTM. Każda warstwa zawiera w niej wiele komórek pamięci. Wiele ukrytych warstw sprawia, że sieć jest głębsza, co pozwala na używanie bardziej złożonych zadań,

[34] *Understanding LSTM Networks,* http://colah.github.io/posts/2015-08-Understanding-LSTMs/ [dostęp 11.07.2022 r.].

takich jak rozwiązywanie problemów z rozpoznawaniem mowy, w porównaniu do Vanilla LSTM. Architektura Stacked LSTM jest przedstawiona na rysunku 6.9.



Rysunek 6.9-Architektura Stacked LSTM

Architektura CNN LSTM (LRCN, long-term Recurrent convolutional Network) przedstawiona na rysunku 6.10, Sugeruje użycie warstw konwolucyjnej sieci neuronowej (CNN, Convolutional Neural Network) do wyodrębnienia cech na danych wejściowych w połączeniu z LSTM do przewidywania sekwencji.

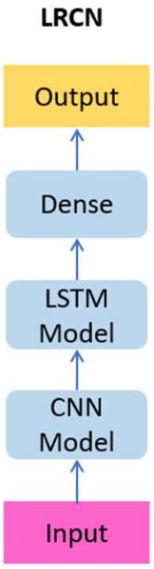
CNN LSTM zostały opracowane dla wizualnych problemów przewidywania szeregów czasowych i generowania opisów tekstowych sekwencji obrazów, takich jak wideo. CNN LSTM są wykorzystywane do:

1) rozpoznawanie aktywności: generowanie tekstowego opisu działania pokazanego przez sekwencję obrazów;

2) opisy obrazu: tworzenie opisu tekstowego pojedynczego obrazu,

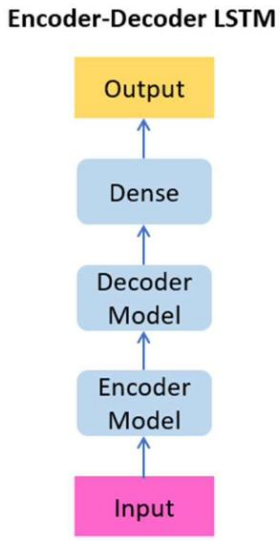
3) opisy wideo: generowanie opisu tekstowego sekwencji obrazów.

Dzięki temu warstwa CNN w LRCN może również określać cechy sekwencji wejściowych w formacie tekstowym i audio, a zatem może być używana nie tylko do rozwiązywania problemów wizualnych.



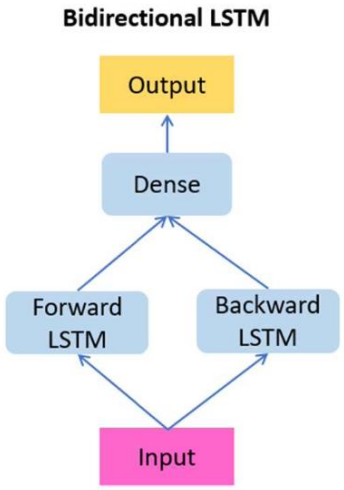
Rysunek 6.10 - Architektura CNN LSTM (LRCN)

Do rozwiązywania problemów przewidywania sekwencji (seq2seq, Sequence to Sequence) stosuje się architekturę Encoder-Decoder (Enkoder-Dekoder) LSTM, przedstawioną na rysunku 6.11. Encoder-Decoder LSTM składa się z 2 głównych części składowych: Encoder – do odczytu sekwencji wejściowej i kodowania jej do wektora o stałej długości oraz Decoder – do dekodowania i wyprowadzania prognozy sieci. W tym przypadku zakłada się, że Encoder i Decoder są warstwami LSTM, jednak jeśli weźmiemy pod uwagę Encoder-Decoder jako całość, nie jest konieczne użycie LSTM. Za pomocą Encoder-Decoder LSTM można rozwiązywać zadania związane z tłumaczeniem maszynowym, obliczaniem wyniku wykonania małych programów, a także można używać CNN jako Encoder (LRCN) do opisywania obrazów tekstowych, na przykład do tworzenia napisów, do klasyfikacji ruchów.



Rysunek 6.11 - Architektura Encoder-Decoder LSTM

Wśród szeroko stosowanych architektur LSTM można również wyróżnić dwukierunkową architekturę LSTM (Bidirectional LSTM) przedstawioną na rysunku 6.12. Dwukierunkowy LSTM jest w stanie przeglądać sekwencję wejściową w dwóch przeciwnych kierunkach; w tym celu pierwsza rekurencyjna warstwa sieci jest zduplikowana i znajduje się na tym samym poziomie, przy czym jedna z warstw otrzymuje sekwencję wejściową taką, jaka jest, a druga – jej odwrotną kopię. Takie podejście pozwala poprawić wydajność sieci LSTM.



Rysunek 6.12 - Architektura Bidirectional LSTM

Początkowo dwukierunkowe LSTM były przeznaczone do rozwiązywania problemów z

rozpoznawaniem mowy, jednak takie podejście do oglądania sekwencji w dwóch kierunkach znalazło zastosowanie również w rozwiązywaniu problemów z prognozowaniem sekwencji.

Jeśli sieć LSTM jest stosowana do rozwiązywania problemów związanych z problemem sekwencyjnego prognozowania seq2seq, oznacza to, że LSTM używany do tego celu jest generatywny (Generative LSTM). Istnieje możliwość więc przy tym wybrać architekturę sieci, a więc może być to na przykład Vanilla LSTM lub inny rodzaj architektury[35].

6.3 Zasada działania sieci Transformer

Jeśli chodzi o problem przewidywania sekwencji seq2seq, do rozwiązywania problemów zwykle używane są sieci oparte na podejściu Encoder-Decoder. Jednak przy użyciu LSTM lub innych rodzajów RNN wektor kontekstu, w którym Encoder konwertuje sekwencję wejściową, jest najbardziej „wąskim" gardłem całego systemu. Jedno z rozwiązań[36] zostało zaproponowane przez Manh-Thang Long (Minh-Thang Long) w 2015 r. W tym rozwiązaniu zaproponowano użycie koncepcji uwagi (Attention). Uwaga pozwala modelowi skupić się na odpowiednich i ważnych częściach sekwencji wejściowej w razie potrzeby.

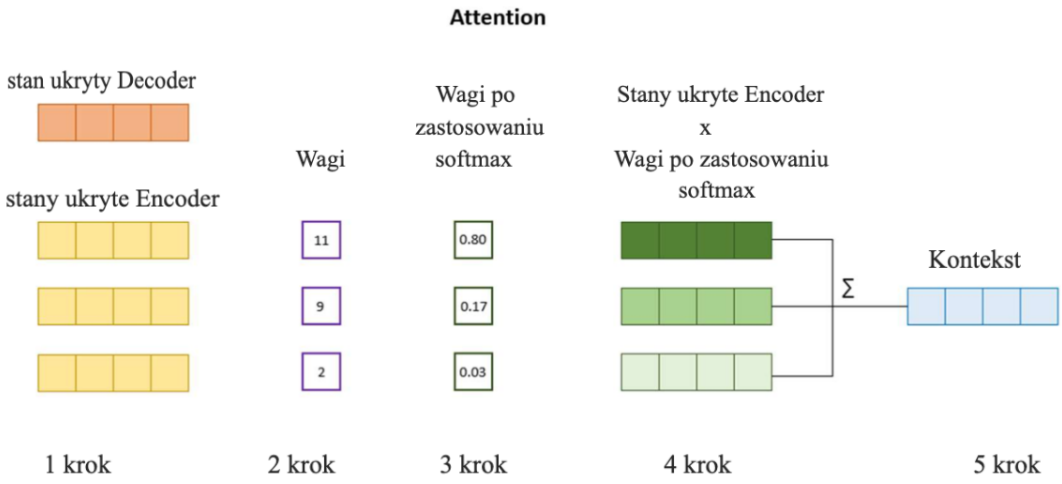
Modele seq2seq oparte na uwadze są różne od zwykłych. W zwykłych modelach koder (Encoder) przekazuje tylko ostatni stan ukryty jako kontekst do dekodera (Decoder). Gdy używany jest Attention, koder wysyła wszystkie ukryte stany do dekodera, który również wykonuje dodatkowe przetwarzanie przed wyjściem, ponieważ teraz każdy ukryty stan wysłany przez koder jest powiązany z określoną częścią sekwencji wejściowej. Na przykład, jeśli model oparty na Attention tłumaczy frazę z jednego języka na inny, wówczas pierwszy ukryty stan w kontekście będzie powiązany z pierwszym słowem, drugi z drugim itd. W ten sposób dekoder otrzymuje więcej informacji i może z nich korzystać.

Dekoder oblicza wagę każdego ukrytego stanu, który otrzymuje od enkodera, przy użyciu również ukrytego stanu dekodera (tj. wyjście z poprzedniego kroku czasowego), softmax jest stosowany dla tych wag. Dekoder następnie mnoży każdy ukryty stan przez uzyskaną wagę i podsumowuje uzyskane wyniki. W ten sposób tworzony jest nowy wektor kontekstowy z uwagą na ważne części, który wykorzystywany jest jako dane wejściowe dla neuronowej sieci sprzężenia do przodu - umożliwia ona uzyskanie elementu

[35] Jason Brownlee, *Gentle Introduction to Generative Long Short-Term Memory Networks* https://machinelearningmastery.com/gentle-introduction-generative-long-short-term-memory-networks/ (dostęp 11.07.2022 r.).

[36] Minh-Thang Luong, Hieu Pham, Christopher D. Manning, *Effective Approaches to Attention-based Neural Machine Translation*, 2015.

wyjściowego sekwencji. Dane wyjściowe są wykorzystywane przez dekoder jako stan ukryty do obliczania wag w następnym kroku czasowym. Opisany proces jest schematycznie przedstawiony na rysunku 6.13.



Rysunek 6.13 - proces Attention dla dekodera

Istnieje jeszcze jedna interesująca koncepcja, która jest już stosowana dla kodera-SELF-Attention[37]. Celem tej koncepcji jest połączenie dwóch elementów z sekwencji wejściowej. Na przykład musimy przetłumaczyć zdanie „the animal didn' t cross the street because it was too tired", rozważmy słowo „it": to, że odnosi się do słowa „animal", jest zrozumiałe dla osoby, ale nie dla sieci. Jako iż koder obsługuje każdy element z sekwencji wejściowej, koncepcja SELF-Attention daje mu możliwość wyszukiwania wskazówek w innych elementach z sekwencji wejściowej, aby lepiej ją zakodować. Podobnie jak RNN używa stanu ukrytego do pobierania informacji o poprzednim elemencie z sekwencji, koder używa SELF-Attention do wyodrębnienia zrozumienia bieżącego przetwarzanego elementu z innych danych wejściowych w sekwencji.

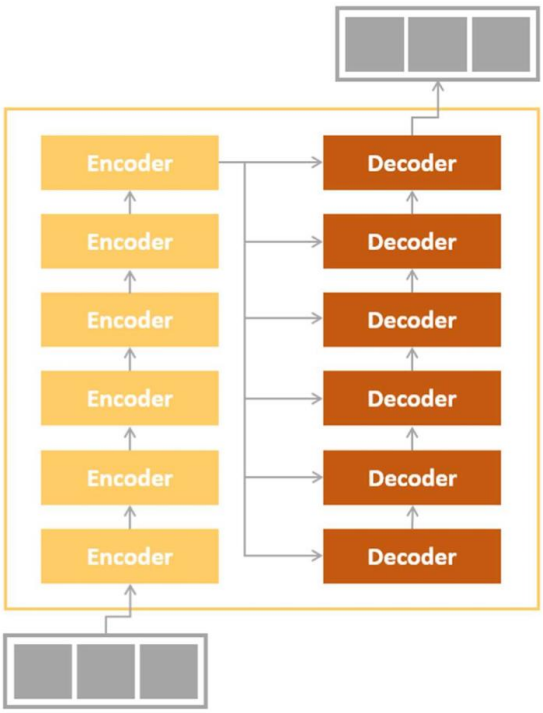
Rozważmy główne etapy tego procesu. Dla każdego elementu sekwencji wejściowej obliczane są trzy wektory: Query – Q, Key – K I Value – V. Są one obliczane przy użyciu trzech macierzy, które są tworzone podczas szkolenia. Następnie obliczane są wyniki (score) dla każdego elementu w sekwencji wejściowej za pomocą iloczynu skalarnego (Dot Product) wektora Query z wektorem Key innych elementów w sekwencji (w praktyce zamiast wektorów dla każdego elementu stosuje się macierze łączące te wartości). Na

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin. *Attention Is All You Need,* 2017.

przykład, jeśli obliczana jest SELF-Attention dla słowa „You” w zdaniu „You are awesome”, tworzony jest wynik dla każdego słowa w zdaniu w odniesieniu do danego słowa, tj. obliczane są iloczyn skalarny Q („You”) i K („You”), Q („You”) i K („are”) oraz Q („You”) i K („awesome”). Uzyskane wyniki są podzielone przez pewną wartość (domyślnie jest to 8, ale można użyć innych), a następnie stosuje się do nich funkcję softmax, aby były znormalizowane, pozytywne i wynosiły 1 jako całość. Następnie musimy pomnożyć wektor Value i uzyskane wyniki zsumować w jeden wektor, tak abyśmy otrzymali wynik SELF-Attention, który jest przekazywany do wejścia sieci neuronowej ze sprzężeniem w przód.

Rozważmy architekturę sieci neuronowych-Transformer (Transformer), która łączy i wykorzystuje wszystkie wyżej wymienione pojęcia. Transformery doskonale sprawdzają się w problemach z tłumaczeniem maszynowym, ale mogą być również używane do innych zadań seq2seq. A niektóre części tej architektury można również wykorzystać do rozwiązania innych problemów w ramach komplikacji z przewidywaniem sekwencji.

Transformery składają się z kilku koderów i dekoderów ułożonych jeden po drugim, jak pokazano na rysunku 6.14. Oryginalny artykuł, w którym zaproponowano tę architekturę, wykorzystuje 6 koderów i 6 dekoderów, jednak ich liczba może się różnić.



Rysunek 6.14 - Architektura Transformer[38]

[38] *Introduction to Transformers Architecture,* https://rubikscode.net/2019/07/29/introduction-to-transformers-architecture/ (dostęp 11.07.2022 r.)

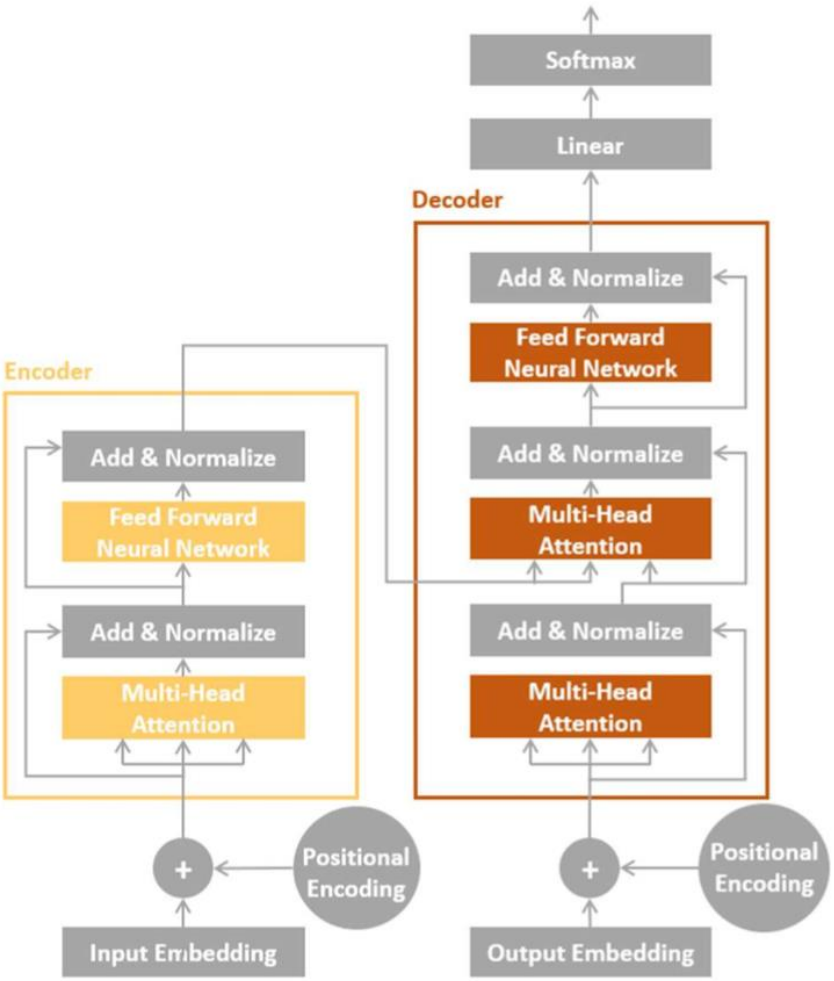
Każdy koder składa się z dwóch części: warstwy SELF-Attention i sieci neuronowej jednokierunkowej (Feed Forward Neural Network), natomiast każdy dekoder składa się z trzech części: warstwy SELF-Attention, warstwy Attention dekodera i sieci neuronowej jednokierunkowej (Feed Forward Neural Network), jak pokazano na rysunku 6.15.

Rysunek 6.15-struktura kodera i dekodera

Pierwszy koder otrzymuje listę wektorów wejściowych. Kiedy pracujemy ze słowami, zwykle jest to wyjście jakiejś warstwy Embedding. Embedding jest odpowiedzialny za konwersję tekstu na reprezentację numeryczną w oparciu o jego znaczenie semantyczne, do tego przedstawienia dodaje się również przekonwertowane informacje o pozycji słów w tekście, za co odpowiada konwersja Positional Encoding. Koder przetwarza listę wektorów wejściowych za pomocą warstwy SELF-Attention, a następnie Feed Forward NN, po czym dane wyjściowe są wysyłane do następnego kodera. Koder końcowy wysyła te informacje do dekoderów, które wykonują podobny proces[39].

Bardziej szczegółowo strukturę warstw Transformera Encoder-Decoder przedstawiono na rysunku 6.16, gdzie oprócz wspomnianych wcześniej warstw występują również warstwy Multi-Head Attention, implementujące koncepcję uwagi, Add & Normalize do sumowania i normalizacji danych przychodzących do wejścia danej warstwy, a także warstwa liniowa Linear na wyjściu dekodera.

[39] *Introduction to Transformers Architecture,* https://rubikscode.net/2019/07/29/introduction-to-transformers-architecture/ (dostęp 11.07.2022 r.)



Rysunek 6.16 - struktura Encoder - Decoder warstw Transformera[40]

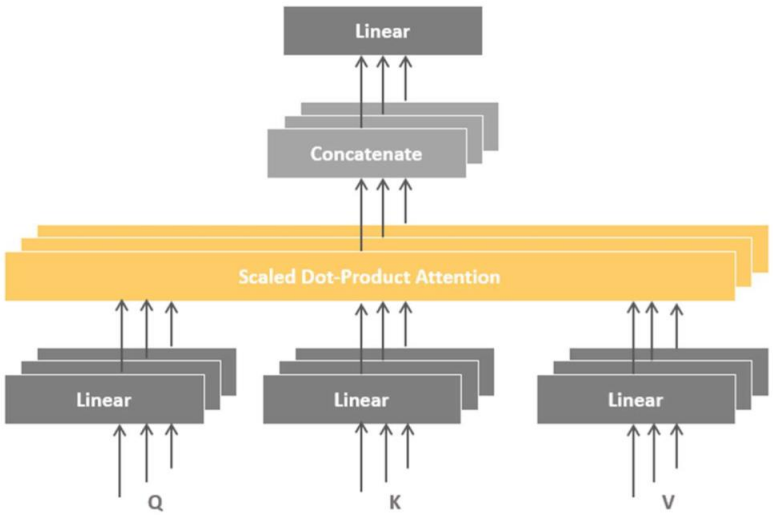
Aby dokładniej przeanalizować warstwę Multi-Head Attention, należy wziąć pod uwagę jej część składową – warstwę Scaled Dot-Product Attention, której strukturę przedstawiono na rysunku 6.17. Podobnie jak w innych warstwach uwagi, danymi wejściowymi warstwy są wektory (w praktyce kolumny macierzy) Q, K (z pomiarem d_k) i V (z pomiarem d_v). Obliczany jest iloczyn skalarny elementów Q ze wszystkimi elementami K podzielnymi przez pierwiastek kwadratowy z d_k, a następnie zastosowana zostaje funkcja softmax. Następnie, obliczane są iloczyn skalarny uzyskanych wyników i elementów V. Matematycznie tę warstwę można opisać formułą (15).

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) V \qquad (15)$$

[40] *Transformer with Python and TensorFlow 2.0 – Attention Layers*, https://rubikscode.net/2019/08/05/transformer-with-python-and-tensorflow-2-0-attention-layers/(dostęp 15.07.2022 r.)

Rysunek 6.17 - Scaled Dot-Product Attention[41]

Rysunek 6.18 przedstawia strukturę warstwy Multi-Head Attention, której głównym składnikiem jest rozpatrywana przez nas warstwa Scaled Dot-Product Attention. Aby poprawić jakość wyniku, stosuje się kilka warstw Scaled Dot-Product Attention że zamiast jednej „głowy", uwagi, Q, K i V są podzielone na kilka „głów". W ten sposób model może uzyskiwać dostęp do informacji na różnych pozycjach z różnych oddzielnych przestrzeni.



Rysunek 6.18 - Multi-Head Attention[42]

[41] *Transformer with Python and TensorFlow 2.0 – Attention Layers*, https://rubikscode.net/2019/08/05/transformer-with-python-and-tensorflow-2-0-attention-layers/(dostęp 15.07.2022 r.)

Na tej warstwie wykonywane są cztery kroki. Najpierw ta warstwa pobiera dane na trzy wejścia-Q, K, V i dzieli je za pomocą warstwy liniowej na trzy gałęzie lub „głowy". Warstwa Scaled Dot-Product Attention jest następnie wyzwalana dla każdej „głowy", a następnie wyniki są łączone i przekazywane do końcowej warstwy liniowej[43]. Matematycznie warstwę Multi-Head Attention można przedstawić formułami (16) i (17).

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h)W^O \qquad (16)$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \qquad (17)$$

Transformery mają duży potencjał i pomagają osiągnąć doskonałe wyniki w rozwiązywaniu problemów związanych z prognozowaniem sekwencji. Dość często wyniki są lepsze niż w przypadku sieci rekurencyjnych, chociaż te ostatnie są również nadal wykorzystywane w wielu zadaniach.

6.4 Osadzenie słów(Word Embedding)

Najpopularniejsze na dzień dzisiejszy algorytmy tworzenia reprezentacji wektorowych tekstu opierają się na idei semantyki dystrybucyjnej. Polegają na tym, że słowa występujące w podobnych kontekstach i posiadające zbliżoną lub taką samą częstotliwość użycia, są semantycznie bliskie. Jednocześnie odpowiadające im skompresowane reprezentacje wektorowe, których wymiar jest znacznie mniejszy niż wymiar słownika (embeddings), są blisko siebie w skali kosinusowej w pewnej słownej przestrzeni wektorowej.

Jedną z podstawowych metod reprezentowania dokumentu tekstowego jako wektora jest miara statystyczna TF-IDF, która jest obliczana jako iloczyn częstotliwości słów w tekście i odwrotności częstotliwości słów w zbiorze dokumentów. Wymiar wektorów TF-IDF zależy od liczby słów w słowniku dokumentu, a ponieważ może być to obszerne, wymiar przestrzeni wektorowej jest wystarczająco duży; ponadto wektory TF-IDF będą blisko

[42] *Transformer with Python and TensorFlow 2.0 – Attention Layers*, https://rubikscode.net/2019/08/05/transformer-with-python-and-tensorflow-2-0-attention-layers/(dostęp 15.07.2022 r.)

[43] *Transformer with Python and TensorFlow 2.0 – Attention Layers*, https://rubikscode.net/2019/08/05/transformer-with-python-and-tensorflow-2-0-attention-layers/(dostęp 15.07.2022 r.)

tylko dla tych dokumentów, które mają wspólne słownictwo[44]. Modele wektoryzacji tekstów informacyjnych zyskały prawdziwą popularność w 2013 r. po opublikowaniu pracy: T. Distributed Representations of Words and Phrases and their Compositionality. In: Proceedings of Workshop z 2013 roku. opisującej podejście, które stało się znane jako Word2Vec. Ta metoda konstruowania skompresowanej przestrzeni wektorów słów (tzw. embeddings) ma dwa główne rozwiązania: 1) na podstawie algorytmu CBOW (continuous-bag-of-words) dla każdego słowa, które przewiduje prawdopodobieństwo jego wystąpienia w określonym kontekście (otaczające słowa) oraz 2) na podstawie algorytmu skip-gram, który w przeciwieństwie do CBOW działa odwrotnie – oblicza prawdopodobieństwo kontekstu wokół konkretnego słowa. Wynikowa reprezentacja wektorowa odzwierciedla kontekstową bliskość słów. Słowa występujące w tekście obok tych samych mają wysokie podobieństwo cosinusowe. Oznacza to, że możemy mówić o semantycznej bliskości. W wyniku szkolenia modelu Word2Vec tworzony jest stały słownik, w celu uzupełnienia którego konieczne będzie ponowne przeszkolenie modelu. Rozwiązanie problemu brakujących słów charakterystycznego dla tego podejścia zostało zaproponowane w modelu fastText. Będąca modyfikacją Word2Vec, fastText oblicza najpierw reprezentacje wektorowe części słów, z których już składa się wektor całego słowa.

Do tej pory istnieje wiele innych modeli wektoryzacji tekstów, wśród których warto zwrócić uwagę na model GloVe[45], zaproponowany przez laboratorium lingwistyki komputerowej Uniwersytetu Standford; łączy ona algorytmy dekompozycji macierzy i Word2Vec. Powyższe metody wektoryzacji nazywane są statycznymi. Pozostają one dość popularne, ale mają znaczne ograniczenie: nie uwzględniają wielowartościowości i kontekstowego charakteru słów, co oznacza, że dla jednego słowa występującego w różnych kontekstach zostanie zaproponowane jedno uśrednione embedding. Aby pokonać tę wadę, ostatnio opracowano i rozpowszechniono dynamiczne (kontekstualizowane) modele językowe, które umożliwiają obliczanie embeddingów dla słowa (lub całego zdania) w zależności od kontekstu użycia.

W 2018 roku firma Google zaprezentowała nowy model o nazwie BERT, który dokonał przełomu w dziedzinie przetwarzania języka naturalnego. Chociaż BERT ma teraz wielu konkurentów, w tym modyfikacje klasycznego modelu (RoBERTa, DistilBERT i in.) jak i zupełnie nowe (takie jak XLNet), nadal pozostaje w top modelach NLP[46].

[44] Sanjeev Arora, Yingyu Liang, Tengyu Ma, *A SIMPLE BUT TOUGH-TO-BEAT BASELINE FOR SENTENCE EMBEDDINGS*, Princeton University, 2017

[45] Jeffrey Pennington, Richard Socher, Christopher D. Manning - *GloVe: Global Vectors for Word Representation*

[46] *10 Leading Language Models For NLP In 2022,* https://www.topbots.com/leading-nlp-language-models-2020/ (dostęp 16.07.2022 r.)

Model ten jest stosowany w wielu zadaniach, takich jak klasyfikacja tekstów, generowanie tekstu, sumowanie tekstu, itp. Można natomiast również użyć go do uzyskania wektorowych reprezentacji tekstu-embeddings.

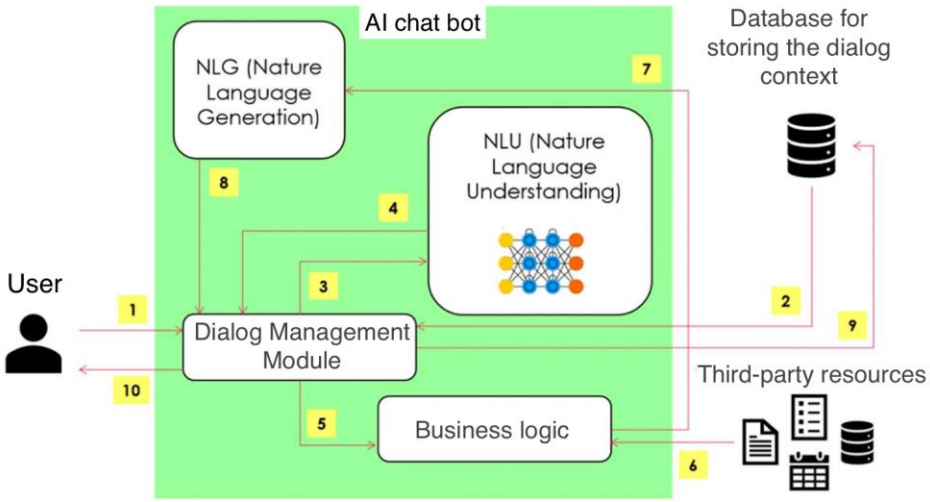
Szkolenie reprezentacji wektorowych Berta polega na rozwiązaniu problemu, którego istotą jest wstępne maskowanie słów w tekstach próbki treningowej (a także łatwość mieszania słów w celu poprawy późniejszej konfiguracji modelu), które model próbuje następnie zrekonstruować (przewidzieć) z otaczającego ich kontekstu. Najnowsze osiągnięcia w dziedzinie kontekstualizacji embeddingów obejmują również modele takie jak ELMO, XLNET i GPT-2[47]

[47] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, *Language Models are Unsupervised Multitask Learners*

7. Opis architektury chatbota

W ramach tej pracy proponowany jest prototyp chatbota. Zaprojektowałam również strukturę jego ostatecznej implementacji, która jest oparta na przestudiowaniu literatury na temat implementacji chatbotów używanych w procesie biznesowym.

Struktura chatbota AI jest przedstawiona na rysunku 7.1 i składa się z kilku części składowych. Zawiera: moduł zarządzania dialogiem, bazę danych do przechowywania kontekstu bieżących dialogów z użytkownikami, moduł rozpoznawania języka naturalnego (NLU, Natural Language Understanding) w formacie tekstowym, logikę biznesową i skrypty wykonujące dodatkowe czynności, takie jak dostęp do danych zewnętrznych, a także moduł do generowania odpowiedzi tekstowej w celu wysłania jej do użytkownika (NLG, Natural Language Generation).



Rysunek 7.1-Struktura ostatecznej implementacji chatbota AI

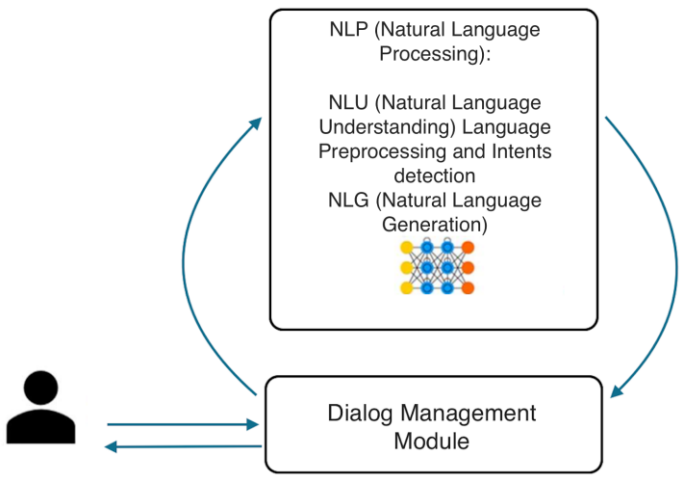
Rozważmy również schemat interakcji komponentów ostatecznej implementacji chatbota AI, których kolejność kroków działania znajduje odzwierciedlenie na rysunku 7.1. Gdy system otrzyma żądanie użytkownika w formacie wiadomości tekstowej, jest ono przekazywane do modułu zarządzania dialogami (1), który ładuje z bazy danych kontekst okna dialogowego (2). Odzwierciedla to aktualny stan korespondencji z użytkownikiem na podstawie wcześniejszych zdarzeń w danym oknie dialogowym. Moduł sterujący dialogiem przesyła wiadomość i kontekst do modułu NLU (3), gdzie tekst wejściowy jest konwertowany. Następnie przekształcone dane wejściowe są przekazywane do sieci neuronowej w celu sklasyfikowania intencji użytkownika.

Moduł NLU odpowiada również za uzupełnienie najbardziej prawdopodobnego zagadnienia o parametry zapytania obecne w frazie użytkownika, które są niezbędne do

dalszego wykonania żądanej akcji (4). Następnie, moduł zarządzania dialogiem określa kolejny, najbardziej odpowiedni stan dialogu (5), a logika biznesowa (6) jest wykonywana, na przykład w przypadku, gdy konieczne jest przekierowanie odwołania użytkownika do operatora na żywo.

Moduł NLG (7) tworzy odpowiedź chatbota AI dla użytkownika (8). W najprostszym przypadku do wygenerowania odpowiedzi można użyć gotowych szablonów oznaczonych dla każdego zapytania i podstawień makr, w bardziej złożonym – użycie mechanizmu sieci neuronowej do wygenerowania odpowiedzi dla użytkownika. Te podejścia można łączyć, ponieważ w przypadku niektórych pytań odpowiedź zawsze będzie ogólna, lecz dla innych może mieć bardziej dowolną formę.

Rozważmy strukturę zaimplementowanego prototypu chatbota AI, pokazaną na rysunku 7.2. Główną częścią struktury jest moduł NLP, który składa się z modułu NLU i NLG. NLU konwertuje dane wejściowe na formę odpowiednią do podawania danych wejściowych sieci neuronowej i określa zamiar użytkownika. Moduł zarządzania dialogami zapewnia interakcję z użytkownikiem na platformie komunikatora (na przykład Telegram) i jednocześnie uzyskuje dostęp do modułu NLP, przesyłając jako wejście jeszcze nie przekonwertowany tekst wiadomości użytkownika, próbując odebrać jego intencję. NLG koncentruje się na konstruowaniu tekstu w języku angielskim na podstawie danego zestawu danych.



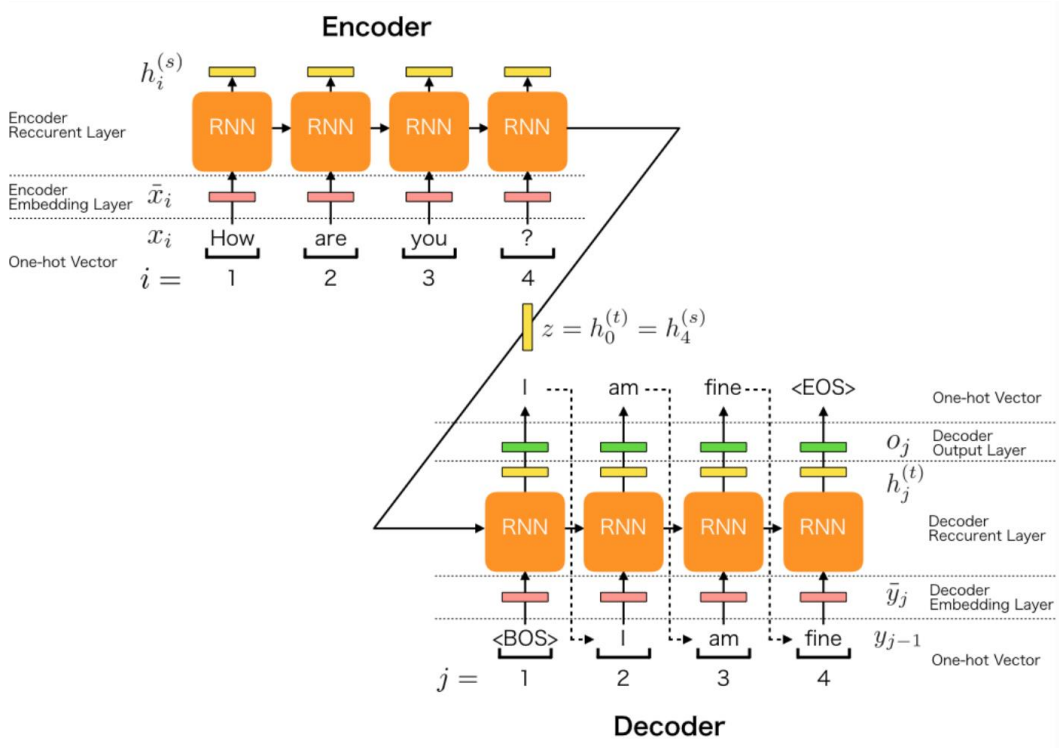
Rysunek 7.2-Struktura prototypu chatbota AI

Rozważmy bardziej szczegółowo proces zachodzący w module NLP. Jak już omówiono w pracy użyjemy modelu Seq2Seq do wdrożenia prototypu chatbota. Model Seq2Seq składa się z dwóch głównych jednostek: enkoder i dekoder.

Istnieją różne modele Sequence to Sequence. W tej pracy użyjemy standardowego modelu opartego na RNN z dodaną warstwą uwagi. Dla porównania weźmiemy model głębokiego uczenia Transformer (transformator), który jest również modelem Sequence to Sequence, lecz nie wykorzystującym rekurencyjnych sieci neuronowych.

Teraz przyjrzyjmy się bliżej architekturze obu modeli użytych w tej pracy.

Rysunek 7.3 przedstawia standardową architekturę Seq2Seq z RNN. Do komórek sieci rekurencyjnej enkoder podawana jest oryginalna fraza podzielona na słowa: „How are you?". Enkoder przetwarza je i na wyjściu otrzymuje zakodowaną sekwencję **z**. Dekoder, oprócz informacji z wyjścia enkodera, otrzymuje odpowiedź referencyjną, na której się uczy: „I am fine". W procesie uczenia się dekoder zmienia swoje wagi w taki sposób, że po otrzymaniu oryginalnego pytania na wejście, pragnie jak najlepiej wydać frazę referencyjną w wyjściu. Podczas treningu fraza jest otoczona tagiem. W tym przypadku <BOS> - jest znacznikiem początkowym i <EOS> - jest znacznikiem końcowym.



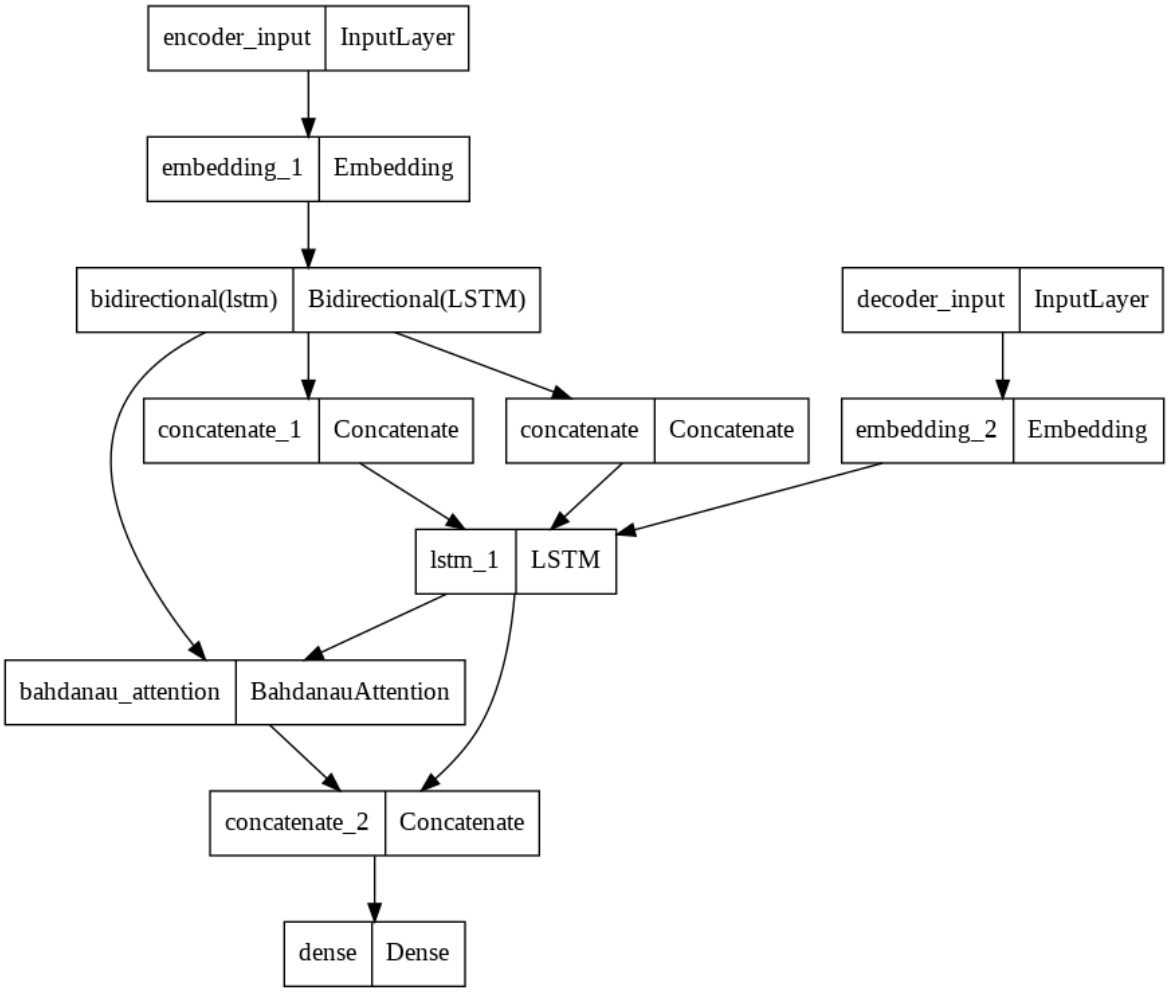
Rysunek 7.3-Architektura modelu seq2seq oparta na RNN[48]

[48] *Write a Sequence to Sequence (seq2seq) Model*, https://docs.chainer.org/en/stable/examples/seq2seq.html (dostęp 18.08.2022 r.)

Pierwszy model został zbudowany z architekturą Encoder-decoder z uwagą (model uwagi Bahdanau[49]).

- Enkoder zawiera warstwę embedding, po której następuje warstwa BiLSTM.
- Dekoder zawiera również warstwę embedding, a następnie warstwy LSTM i Dense.
- Warstwa uwagi pobiera dane wyjściowe enkodera i dane wyjściowe dekodera.

Model Enkoder-Dekoder z Attention zwraca uwagę na określone części tekstu wejściowego. W przypadku sieci neuronowych Seq2Seq użycie Uwagi umożliwia utworzenie wektora kontekstu w dowolnym momencie, biorąc pod uwagę aktualny stan ukryty dekodera i podzbiór stanów ukrytych kodera. Schemat modelu przedstawiono na rysunku 7.4.



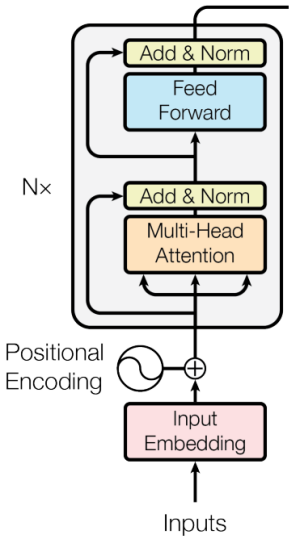
Rysunek 7.4-Model BiLSTM/LSTM z uwagą

[49] Dzmitry Bahdanau, KyungHyun Cho Yoshua Bengio, *Neural Machine Translation by Jointly Learning to Align and Translate*, In: CoRR.2014.Vol. abs/1409.0473. url: https://arxiv.org/abs/1409.0473(dostęp 18.08.2022 r.).

Jeśli chodzi o model Transformer (transformator), jego architektura została szczegółowo opisana i przeanalizowana w części 6.3.

Transformer (transformator) użyty w tym badaniu jest zbudowany bezpośrednio z transformatora opisanego François Chollet w „English-to-Spanish translation with a sequence-to-sequence Transformer”[50] ze zmodyfikowanym wyjściem dla chatbota. Rozważmy jednak bardziej szczegółowo procesy zachodzące w enkoderze i dekoderze modelu Transformer (transformator), który jest realizowany w tej pracy.

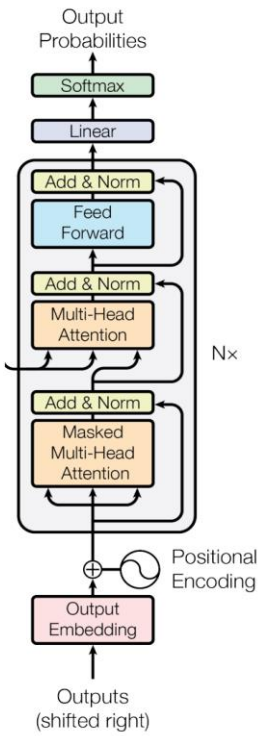
Enkoder składa się z 6 identycznych warstw, gdzie każda z nich składa się z dwóch podwarstw. Pierwsza podwarstwa implementuje multi-head self-attention mechanizm, który otrzymuje liniowe projekcje zapytania (query), kluczy (keys) i wartości (values). Każda projektowana wersja wytwarza równolegle wyjścia, aby wygenerować końcowy wynik tej podwarstwy. Druga podwarstwa jest zdefiniowana przez w pełni połączoną (fully connected) feed-forward sieć. Sieć zawiera dwie warstwy transformacji liniowej z aktywacją ReLU pomiędzy nimi. Połączenie resztkowe jest stosowane wokół każdej podwarstwy, a także warstwy normalizacyjnej. Ze względu na strukturę Transformera (transformatora) nie możemy uzyskać informacji o względnej pozycji słów w sekwencji. W ten sposób obliczamy kodowanie pozycyjne za pomocą funkcji sinus i cosinus o różnych częstotliwościach. Wprowadzamy to kodowanie pozycyjne, po prostu sumując je do embeddingów wejściowych. Rysunek 7.5 przedstawia model enkodera.



Rysunek 7.5 -Enkoder(model Transformer)[51]

[50] François Chollet, *English-to-Spanish translation with a sequence-to-sequence Transformer*, 2021 https://keras.io/examples/nlp/neural_machine_translation_with_transformer/(dostęp 03.10.2022 r.).
[51] Ashish Vaswani et al., *Attention Is All You Need.* In: CoRR.2017.Vol. abs/1706.03762. url: http://arxiv.org/abs/1706.03762(dostęp 18.08.2022 r.).

Dekoder jest bardzo podobny do kodera. Składa się również z 6 identycznych warstw. Jednak tym razem każda warstwa składa się z 3, a nie tylko z 2 podwarstw. Pierwsza podwarstwa otrzymuje wyjście dekodera z poprzedniego kroku czasowego, wzbogacone o kodowanie pozycyjne. Po raz kolejny wdrażamy multi-head self-attention mechanizm. Jednak tym razem skupiamy się tylko na poprzedzających słowach, a nie na wszystkich słowach w sekwencji jednocześnie. Zapewnia to, że przewidywanie w bieżącym kroku czasowym może zależeć tylko od poprzednich słów i nie jest oparte na nieprawidłowych połączeniach w przyszłości. Osiągamy ten efekt poprzez implementację maski, która powstrzymuje istotne wartości macierzy. To określa dekoder jako jednokierunkowy. Druga podwarstwa implementuje multi-head self-attention warstwę podobną do enkodera. Ta warstwa otrzymuje zapytania z poprzedniej podwarstwy oraz klucze i wartości z wyjścia kodera. Ostatnią warstwą jest w pełni połączona sieć feedforward, która jest zasadniczo taka sama jak zaimplementowana w enkoderze. Dodatkowo wdrażamy połączenie resztkowe i stosujemy warstwę normalizacyjną po każdej podwarstwie. Rysunek 7.6 przedstawia model dekodera.



Rysunek 7.6-Dekoder[52]

Opisany model będzie działał w następujący sposób:

- przede wszystkim tworzymy wektor embedding dla każdego słowa w sekwencji;

[52] Ashish Vaswani et al., *Attention Is All You Need.* In: CoRR.2017.Vol. abs/1706.03762. url: http://arxiv.org/abs/1706.03762(dostęp 18.08.2022 r.).

- następnie rozszerzamy embedding o kodowanie pozycyjne;
- następnie to wejście jest wprowadzane do enkodera i przechodzimy do dwóch podwarstw, jak opisano wcześniej;
- teraz dekoder otrzymuje swoje poprzednie wyjście również wzbogacone o kodowanie pozycyjne;
- to wejście jest kierowane do trzech podwarstw;
- nakładamy maskę w pierwszej podwarstwie, aby powstrzymać nieprawidłowe połączenia;
- w drugiej podwarstwie otrzymujemy wyjście kodera obok wyjścia z pierwszej podwarstwy dekodera i przekazujemy je przez w pełni połączoną sieć neuronową, aby wygenerować ostateczne wyjście dekodera;
- w końcu kierujemy wyjście dekodera przez inną w pełni połączoną sieć neuronową i stosujemy funkcję softmax, aby wygenerować prognozę dla następnego słowa w sekwencji.

8. Modele oceny efektywności chatbota

Istnieją dwa główne sposoby oceny chatbota: ocena człowieka i automatyczne wskaźniki oceny. Ocena człowieka polega na poproszeniu grupy uczestników o interakcję z chatbotem, a następnie ocenie różnych aspektów interakcji zgodnie z frameworkiem ewaluacyjnym lub kwestionariuszem. Uczestnicy zazwyczaj oceniają różne aspekty interakcji na podstawie skali, której można użyć do opracowania średnich i pomiaru jakości pracy pod względem efektywności, wydajności i zadowolenia użytkowników[53].

Metryki są również używane do oceny chatbotów. Metryka jest wymierną miarą używaną do oceny procesu biznesowego. Istnieje kilka wskaźników, które pomagają w opracowaniu skutecznego chatbota, takie jak **wynik Bleu**: wynik dwujęzycznego dublera oceny (BLEU); technika stosowana do porównywania wygenerowanej sekwencji słów z sekwencją wzorcową. Wynik BLEU został zaproponowany przez Kishore Papineni w 2002 roku i początkowo był rozwijany tylko do zadań tłumaczeniowych.

Perplexity: jest to pomiar tego, jak dobrze model przewiduje dane testowe. Intuicyjnie perplexity oznacza niezdolność do zrozumienia czegoś skomplikowanego lub niezrozumiałego, więc im niższa perplexity, tym lepiej. Badane jest to za pomocą testowej części korpusu. Jest obliczany przez wykonanie logarytmicznego prawdopodobieństwa poprawnego zdania w całym zestawie testowym. Formuła wygląda następująco:

$$Perplexity = \frac{1}{m}\sum_{i=1}^{m} \sum_{t=1}^{T_y} log(P(y^{<t>})).$$

F1: Miary te obliczają: $F1 = 2 * \frac{precision*recall}{precision+recall}$, gdzie *precision* jest ułamkiem słów w odpowiedzi generowanej, które są również w odpowiedzi ground truth i *recall* jest ułamkiem słów w odpowiedzi ground truth, które są w odpowiedzi generowanej.

Turing Test: Test Turinga jest popularną metodą oceny stosowaną do testowania zdolności maszyny do wykazywania zachowań intelektualnych równoważnych ludzkim. Jeśli tester nie jest w stanie odróżnić odpowiedzi dostarczonych przez człowieka i maszynę, mówimy, że maszyna przeszła test Turinga.

Skalowalność: Chatbot jest bardziej skalowalny, jeśli akceptuje ogromną liczbę użytkowników i dodatkowych modułów.

Szybkość: Jeśli chodzi o szybkość, pomiar szybkości odpowiedzi chatbota odgrywa ważną rolę. Wysokiej jakości chatboty powinny być w stanie szybko dostarczać odpowiedzi.

[53] N. Radziwill, M. Benton. *Evaluating Quality of Chatbots and Intelligent Conversational Agents*

Zarówno automatyczne metryki, jak i ocena człowieka zostaną wykorzystane do określenia skuteczności chatbota w tej pracy. Zostaną użyte następujące automatyczne metryki: Perplexity, F1.

Jeśli chodzi o ocenę przez człowieka, 21 osób weźmie udział w badaniu. Człowiek będzie rozmawiał z chatbotem w sumie około 10 wypowiedzi. Ponieważ chcemy ocenić możliwość wyszkolonych modeli do wykorzystania w biznesie, prosimy o wyobrażenie sobie, że komunikuje się on z konsultantem sklepu internetowego. Następnie człowiek zostanie poproszony o ocenę chatbota na podstawie następujących wskaźników:

1. Płynność: mierzy to między 0-5, ile odpowiedzi chatbota ma sens, biorąc pod uwagę kontekst rozmowy.
2. Spójność: mierzy to między 0-5, jak spójny jest chatbot z odpowiedziami podczas rozmowy.
3. Naturalność: mierzy to między 0-5, jak naturalnie brzmią odpowiedzi chatbota.
4. Perswazyjność: mierzy to między 0-5, szacuje się, jak perswazyjnie może odpowiedzieć chatbot
5. Zaangażowanie: mierzy to między 0-5, jak angażujący jest chatbot podczas rozmowy.

Następnie zostanie obliczony średni wynik. Im wyższy średni wynik modelu, tym lepszy model.

9. Realizacja prototypu AI chatbota i ocena

W poprzednich rozdziałach opisano wiele koncepcji teoretycznych, jak i praktycznych, które dotyczyły tworzenia oraz szkolenia modeli, wykorzystywanych przy tworzeniu chatbotów.
W tym rozdziale zostaną omówione niektóre z konfiguracji modeli opisanych powyżej, wymagania funkcjonalne, a także szkolenie i porównanie modeli używanych do stworzenia prototypu chatbota.

9.1 Przetwarzanie Danych

Aby stworzyć prototyp chatbota, który może generować naturalnie brzmiące odpowiedzi symulujące rozmowę między ludźmi, użyjemy natural language processing do wstępnego przetwarzania danych z Cornell Movie Dialogues Corpus[54]. Ta baza danych zawiera 220 579 dialogów między 10 292 parami postaci filmowych, czyli w sumie 304 713 wypowiedzi. Pomysł wykorzystania tych danych polega na tym, że rozmowy filmowe imitują codzienną rozmowę zawieraną przez ludzi. Jako, iż celem pozostaje stworzenie chatbota, który będzie mógł przeprowadzić naturalną rozmowę z drugą osobą, wykorzystujemy te dane, aby sprawdzić, czy możemy wyszkolić go tak, by w rozmowie brzmiał jak realna osoba.
Poniżej znajduje się podsumowanie danych:
plik movie_lines.txt - zawiera 304 713 linijek z filmów, rzeczywisty tekst każdej wypowiedzi. Posiada pola: lineID, characterID (kto wypowiedział to zdanie), movieID, imię postaci, tekst wypowiedzi.
plik movie_conversations.txt - zawiera 83 098 rozmów. Posiada pola: characterID pierwszej postaci biorącej udział w rozmowie, characterID drugiej postaci biorącej udział w rozmowie, movieID filmu, w którym doszło do rozmowy, lista wypowiedzi, które tworzą rozmowę.
Moduł zarządzania dialogami AI chatbot przyjmuje wiadomości od użytkowników w formacie tekstowym i przesyła je do modułu NLU. Przed przesłaniem danych wejściowych do sieci neuronowej, wejściowe dane tekstowe muszą zostać przekonwertowane na postać numeryczną, z którą sieć może pracować. Poniżej opisano kroki, w jaki sposób osiągnąć ten cel.

[54] https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html(dostęp 24.06.2022 r.).

Przede wszystkim wyodrębniamy wszystkie ciągi dialogowe, jednocześnie wykorzystując każdą rozmowę jako pytanie, po którym następuje odpowiedź. W przypadku sieci neuronowej powinno być jasne, że na tekst pytania udzielana zostaje pewna odpowiedź. Po drugie, każdy element w ciągu dialogowym musi zostać przekształcony w formę, która ułatwi szkolenie modelom. Zmiana została dokonana za pomocą:

- ustawienia wszystkich znaków za pomocą małych liter,
- zastąpienia wszystkich skrótów. Na przykład „i'm" staje się „i am",
- usunięcia wszystkich znaków interpunkcyjnych.

Można to skutecznie zrobić za pomocą wyrażeń regularnych. Teraz będziemy mieli wyczyszczoną listę gotową do dalszej pracy. Następnie musimy tokenizować każdy element w każdym dialogu. Oznacza to przypisanie każdemu słowu unikalnego indeksu ze stałego słownika, a następnie przekonwertowanie surowego tekstu na listę liczb. Dzięki tej zmianie, będziemy mieli listę dialogów, w których każdy element sam w sobie jest listą liczb. Aby utworzyć słownik, użyjemy pakietu Tokenizer nltk. Tworzymy słownik słów, czytając wszystkie słowa ze zbioru danych i przypisując każdemu słowu unikalny indeks. Do słownika dodano również następujące tokeny: 'PAD', 'BOS', 'EOS', ''UNK'. 'BOS' oraz 'EOS' aby wskazać początek i koniec odpowiedzi. 'PAD', aby wskazać zera w słowniku, a 'UNK' stanowi oznaczenie dla nieznanych słów.

Dane wejściowe szeregów czasowych muszą mieć taką samą liczbę kroków czasowych. W ten sposób lista pytań oraz lista odpowiedzi zostaną uzupełnione zerami do określonej długości. Modele powinny używać maskowania wartości zerowych, aby to wypełnienie nie miało wpływu na żadne prognozy.

Jako poprawną odpowiedź na wyjście sieci neuronowej zostanie podany One Hot Encoding (OHE), który został uzyskany z tablicy odpowiedzi w modelu seq2seq. To y_train jest tym, z czym będzie porównywane wyjście dekodera. Pojawia się tu jednak jeden problem. Podczas konwersji na OHE każda z liczb w oryginalnym wektorze rozwinie się w wektor 0 i 1, posiadający długość równą długości słownika. Otrzymamy wtedy dużą, rzadką macierz, która opróżni całą przydzieloną pamięć, po czym Google Colab upadnie. Trening modelu lepiej będzie wykonać na TPU, a nie na GPU. W tym przypadku zasobów jest przydzielanych więcej, a rzadka macierz pozostaje normalnie zbudowana. Niestety, czas nauki wtedy się wydłuża.

Nawet przy użyciu potężnego sprzętu komputerowego pojedyncze szkolenie modelu może potrwać kilka dni. W ograniczonym czasie aktualnie opisywanego projektu było to jednak dość trudne. Aby modele w tej pracy były stosunkowo szybkie, maksymalna liczba próbek szkoleniowych została pobrana przez MAX_SAMPLES i wyniosła 35000. Składają się na

nią, między innymi, trening równy 22400, walidacja o wartości 5600, a także test wynoszący 7000. Maksymalna długość zdania to MAX_LENGTH, równe 20.

Określmy więc hiperparametry dla każdego modelu.

Sequence to Sequence + Attention:

- Typ RNN: BiLSTM/LSTM
- Wymiar Stanu Ukrytego (LATENT_DIM): 128
- Wymiar Embeddingu (EMBED_DIM): 50
- Rodzaj Uwagi: Additive (Bahdanau uwaga)
- Wymiarowość Uwagi: 256
- Dropout: 0.6

Transformer:

- Wymiar Stanu Ukrytego (LATENT_DIM): 512
- Wymiar Embeddingu (EMBED_DIM): 300
- Multi-Head Self Attention heads h: 8
- Dropout: 0.7

Jeśli chodzi o Batch Size dla modeli RNN - w tym projekcie duże Batch Sizes są trudne do osiągnięcia, ponieważ każdy przykład szkoleniowy musi mieć etykietę one-hot, która, jak wspomniano powyżej, jest macierzą o rozmiarze $T_y = V$. Dlatego także każdy przykład szkoleniowy zużywa stosunkowo dużą ilość pamięci, co uniemożliwia duże Batch Sizes z powodu błędów Out of Memory (Brak pamięci). Model oparty na RNN użyje więc Batch Size, w którym użycie GPU w Google Colab nie spowoduje błędu. Ta sama zasada będzie obowiązywać w przypadku Transformer (transformatora), ponieważ eksperymenty wykazały, że większy Batch Size jest zawsze lepszy.[55]

Nie zostało rozjaśnione, jaka powinna być liczba epok, które miałyby zostać przyswojone przez każdy model. Podstawowe parametry dostarczone przez badania są dobrym punktem wyjścia, natomiast pozostają jednak zbyt ogólne, aby można było je zastosować do każdego zadania związanego z generowaniem tekstu. Niestety, w przypadku takich zadań konfiguracja może stać się bardzo czasochłonna, zwłaszcza jeśli ogromna moc obliczeniowa nie jest dostępna.

9.2 Zakres pracy

[55] Martin Popel and Ondrej Bojar. *Training Tips for the Transformer Model.* In: CoRR abs/1804.00247 (2018). arXiv: 1804.00247. url: http://arxiv.org/abs/1804.00247.

Poniżej określamy zakres pracy, i co projekt musi zrobić, aby można go było ocenić jako udany.

1. Seq2Seq (RNN z uwagą): W projekcie zostanie wdrożona wersja modelu Sequence to Sequence, z uwagą, zdolna do generowania dialogów.
2. Transformer: W projekcie zostanie wdrożony model Transformer (transformator), który będzie mógł generować dialogi.
3. Porównanie modeli: Modele wymienione powyżej zostaną przeszkolone i porównane zgodnie z automatycznymi i ludzkimi wskaźnikami oceny.

9.3 Implementacja chatbota

Powyżej omówiliśmy zakres pracy, projekt i specyfikację do wdrożenia owej pracy. Jednak szczegóły, takie jak używane języki programowania, nie zostały tu uwzględnione. W związku z tym, w tym rozdziale opisujemy pewne szczegóły, a także trudności napotkane podczas wdrażania.

Python to język, który został wybrany do realizacji tego projektu, z następujących powodów:

- prosty i elastyczny: Python to język wysokiego poziomu, który abstrahuje wiele złożonych procesów, takich jak zarządzanie pamięcią. Umożliwia on programiście skupienie się wyłącznie na budowaniu wydajnych modeli uczenia maszynowego.
- zróżnicowany ekosystem: Python ma duży zestaw bibliotek i frameworków do głębokiego uczenia się, takich jak Tensorflow, Pytorch i Keras. Python jest również podstawowym językiem aplikacji do uczenia maszynowego i pozostaje używany przez konglomeraty technologiczne na całym świecie.
- niezależność od platformy: Python jest obsługiwany przez wiele platform, wśród których występuje Windows, Linux i macOS.

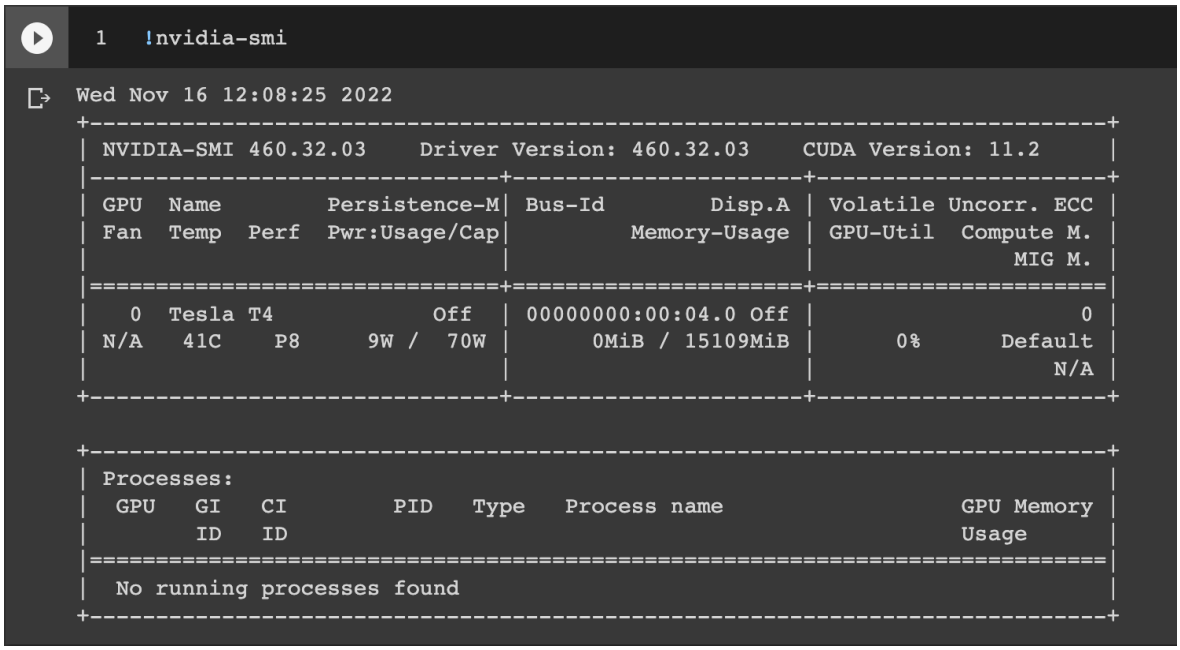
Istnieje kilka bibliotek głębokiego uczenia się, których można byłoby użyć w tym projekcie. Do obecnie opisywanej pracy wybraliśmy Tensorflow, ponieważ:

- posiada interfejs API wysokiego poziomu i elastyczność; Tensorflow zapewnia to, co najlepsze z obu światów, umożliwiając szybkie tworzenie modeli ogólnych za pomocą wysokiego poziomu interfejsu API, a także tworzenie modeli niestandardowych, z możliwością zawarcia podklas, warstw i modeli.
- jest wydajny podczas wdrażania: TensorFlow Serving zapewnia standardowy i zoptymalizowany sposób wdrażania modeli uczenia maszynowego.

Wykorzystano w nim również bibliotekę Keras, która działa na Tensorflow w celu uproszczenia i przyspieszenia szeregu funkcji.

9.4 Szkolenie modeli

Wszystkie modele zostały przeszkolone w Google Collab, w wersji Google Collab Pro, która zapewnia dostęp do GPU Tesla T4, a także priorytetowy dostęp do TPU.



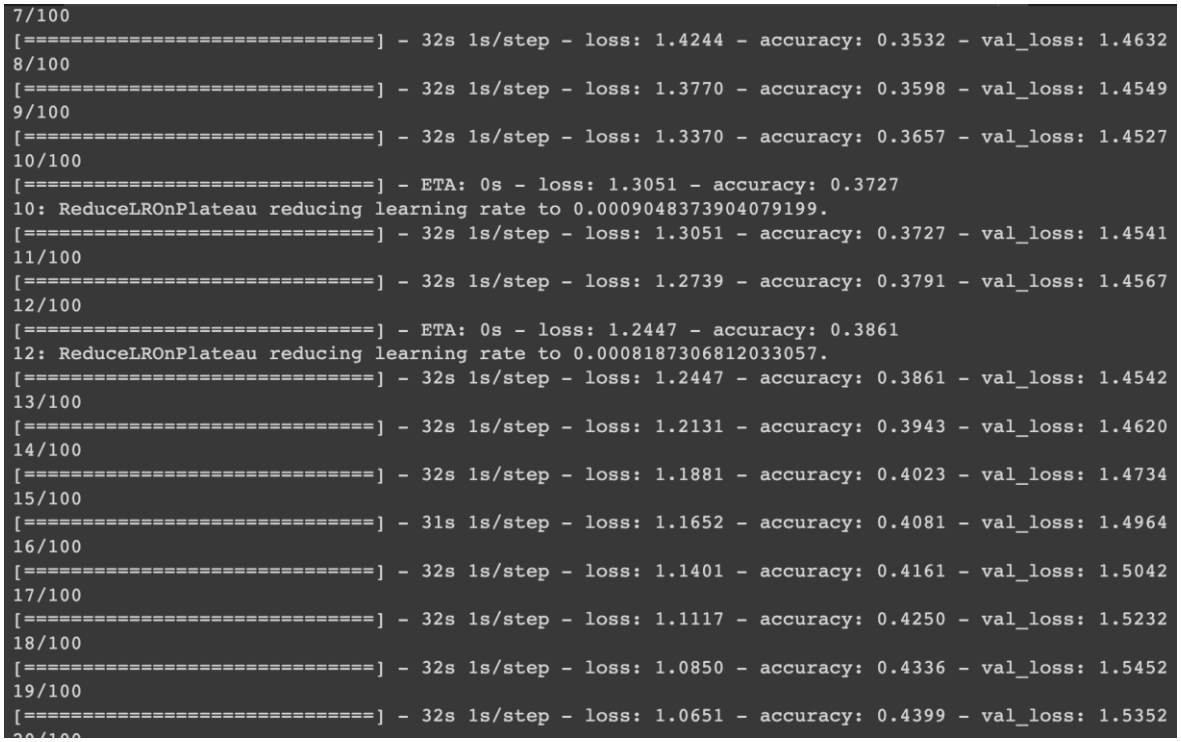
Rysunek 9.1 - nvidia-smi output

Niestety, nadal wymagane jest pewne dostrojenie hiperparametrów oraz architektury modelu, ponieważ modele nie działały tak dobrze, jak początkowo oczekiwano. Było to bardzo czasochłonne, ponieważ nie było klarownych wyjaśnień, co dokładnie należałoby zmienić.

Aby zidentyfikować różnice między wynikami a samym procesem uczenia się dla dwóch różnych konfiguracji, początkowo szkolenie przeprowadzono na niezbyt dużej części zestawu danych. Chociaż seq2seq + attention nie dawał złych wyników, często występowały jednak problemy z nadmiernym dopasowaniem w miarę powiększania się epok. Ujawniono również wadę związaną z faktem, iż proces uczenia się był niezwykle czasochłonny. Na objętości zestawu danych, którego użyliśmy w pracy oraz przy określonych hiperparametrach, trwała ona około 140 sekund na epokę, gdy na tej samej objętości danych na epokę uczono model transformator około 15 sekund.

Model Transformer (transformator) posiada również problemy z nadmiernym dopasowaniem, ponieważ już po kilku epokach loss w walidacji nie zmniejszał się, a loss

w szkoleniu faktycznie uległ pomniejszeniu. Problem nadmiernego dopasowania można natomiast rozwiązać za pomocą zmniejszenia złożoności modelu lub poprzez wprowadzenie regularyzacji, jaką pozostaje chociażby dropout.



Rysunek 9.2 - Przykład problemu nadmiernego dopasowania (overfitting)
Transformer model

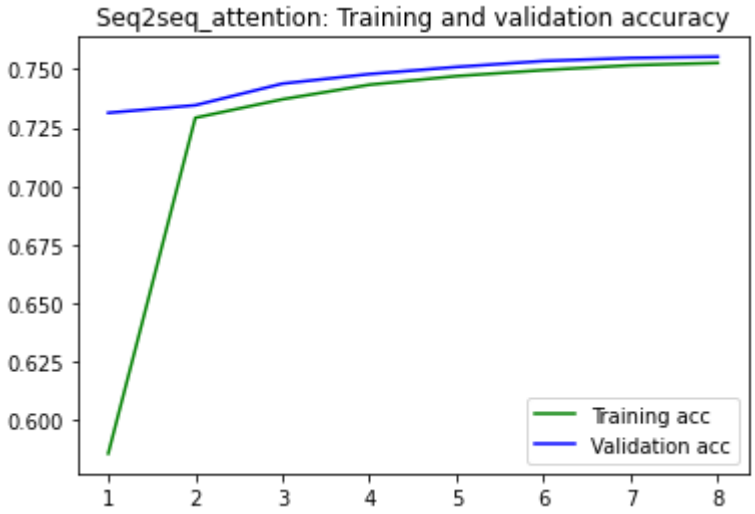
Zmniejszenie hiperparametru Hidden State Dimensionality (Wymiar Stanu Ukrytego) zmniejszyło także złożoność modelu, ale niestety, w dalszym ciągu nie wpłynęło to na problem nadmiernego dopasowania. Dodanie różnych poziomów dropout delikatnie pomogło, lecz dalej niewystarczająco. Modele zatrzymane przed rozpoczęciem zwiększania loss na danych walidacyjnych również przed rozpoczęciem nadmiernego dopasowania dawały bardzo ogólne i bezpieczne odpowiedzi. Na przykład „i don't know”, „im sorry” na prawie każde pytanie (Rysunek 9.3).

```
Input:Hello
Output:  BOS i am sorry EOS
--------------------------------------------------
Input:I need to book a ticket for the next flight to Boston.
Output:  BOS i am sorry EOS
--------------------------------------------------
Input:I understand that you can help me do this quickly?
Output:  BOS i am sorry EOS
--------------------------------------------------
Input:When is the next flight to Boston?
Output:  BOS i am not going to be a [UNK] EOS
```

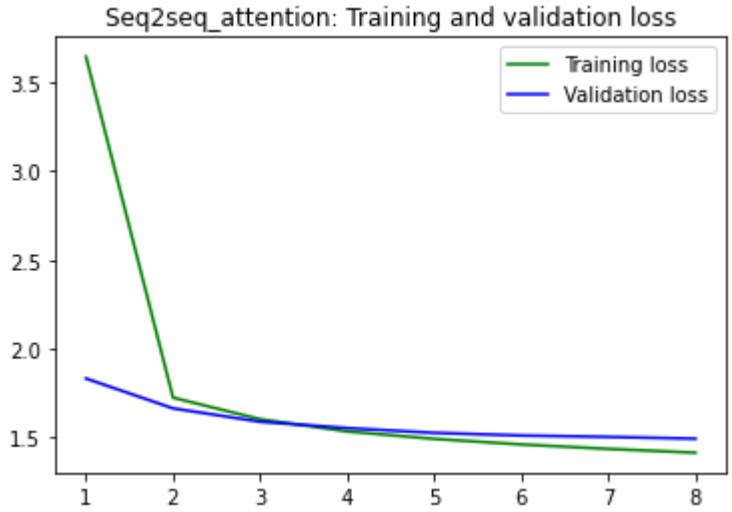
Rysunek 9.3 - odpowiedzi modeli Transformer(transformatora), zatrzymanego uczenie się przed rozpoczęciem nadmiernego dopasowania

Z tego powodu wydaje się, że validation loss (błąd na zbiorze sprawdzającym) nie jest dobrym wskaźnikiem jakości odpowiedzi chatbota. Inne projekty potwierdziły ten wniosek[56]. Aby umieć sobie z tym poradzić, każda klasa modelu została zapisana dwukrotnie: raz, gdy osiągnęła najmniejszą loss w walidacji, i jeszcze raz, aż loss w szkoleniu przestały się zmniejszać. Punkt kontrolny, który osiągnął najmniejszą wartość validation loss (błędu na zbiorze sprawdzającym), zostaje używany do automatycznych miar oceny efektywności, podczas gdy inny punkt kontrolny pozostaje w użyciu do miar ludzkich.

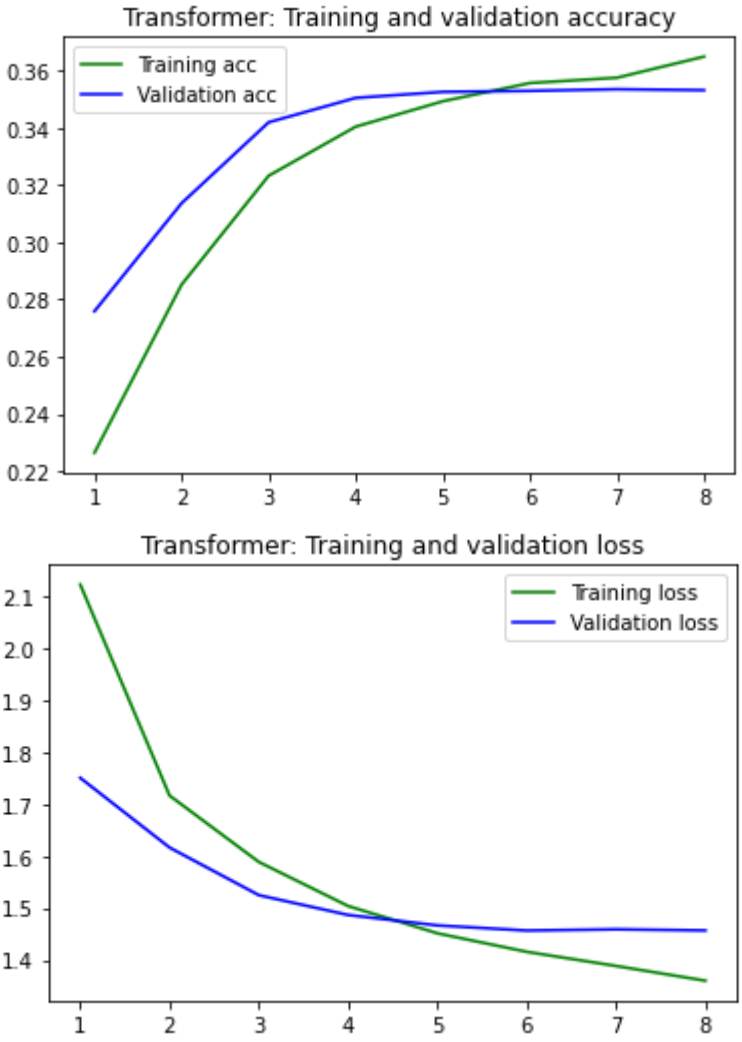
Wykresy zmian accuracy(dokładność) i loss(strata) modeli seq2seq + attention i Transformer, uzyskane w procesie szkolenia i testowania, przedstawiono odpowiednio na rysunkach 9.4, 9.5.



[56] Richard Kriszti. *Deep Learning Based Chatbot Models.* 2019. url: https://arxiv.org/pdf/1908.08835.pdf (dostęp 12.11.2022 r.).



Rysunek 9.4-fragment wykresu zmiany dokładności i straty sieci Seq2seq+attention



Rysunek 9.5-fragment wykresu zmiany dokładności i straty sieci Transformer(transformatora)

Notatniki z wyszkolonymi modelami można także znaleźć na stronie github: https://github.com/Tsyhankova/thesis.

10. Analizy i wnioski

Należy porównać wydajność wyszkolonych modeli pod kątem różnych wskaźników. Aby wybrać który z nich działa lepiej, model pozostaje oceniony, biorąc jego minimalną ocenę z Perplexity i F1. Wyniki dla metryk automatycznej oceny efektywności przedstawiono w poniższej tabeli:

Nazwa modelu	Perplexity	F1
Seq2Seq + Attention	**34.0**	0.35
Transformer	80.8	**0.37**

Tabela 1 - perplexity: niższe jest lepsze. F1: wyżej jest lepiej.

Trudno jest zobaczyć, który z modeli jest lepszy na podstawie tych danych. Z jednej strony perplexity jest miarą tego, jak bardzo model nie jest pewien poprawnej odpowiedzi. Natomiast F1 mierzy dopasowanie słów między przewidywaną a poprawną odpowiedzią.
Wskaźniki oceny efektywności przez człowieka zapewnią z pewnością większą przejrzystość. Ponadto, wskaźniki automatyczne nie korelują dobrze z pomiarami oceny przez człowieka.[57] W związku z tym, większy nacisk zostanie położony na wskaźniki oceny przez człowieka przy wyborze najkorzystniejszego modelu.
Sumując, dwadzieścia jedna osoba przeprowadziła rozmowę z każdym modelem. Ze względu na chęć oceny chatbota od strony wykorzystania w konkretnych procesach biznesowych, zostały one zaprezentowane jako konsultant online w sklepie elektronicznym. Każdy uczestnik został poproszony o rozmowę z „doradcą", która miała miarę dziesięciu wypowiedzi, a następnie o ocenę modeli. Badanie przeprowadzono na podstawie szacunkowych wskaźników, jak zostało to opisane w pracy Ahtsham Manzoor[58], gdzie uczestnicy badania musieli ocenić udzielone odpowiedzi chatbota w skali od 1 do 5 pod względem płynności, spójności, naturalności, perswazji, jak i zaangażowania. Średni wynik dla każdej miary każdego modelu pokazano w poniższej tabeli. Podsumowując, modelom łatwo było do osiągnięcia wysokiej płynności i spójności, jeśli zaangażowanie było niskie, tak jak miało to miejsce w przypadku jednego z nich, zatrzymanego z najmniejszą validation loss (błąd na zbiorze sprawdzającym). Dzieje się

[57] Chia-Wei Liu et al. *How NOT To Evaluate Your Dialogue System: An Empirical Study of Unsupervised Evaluation Metrics for Dialogue Response Generation.* In: CoRR abs/1603.08023 (2016). arXiv: 1603.08023. url: http://arxiv.org/abs/1603.08023
[58] Ahtsham Manzoor, Dietmar Jannach, *Towards retrieval-based conversational recommendation*, 2022 r.

tak, ponieważ mogą one dawać ogólne odpowiedzi, skutkujące wysoką rozdzielczością i spójnością - takie jak przykładowe „I'm not sure". Ten typ modelu można jednak uznać za oszustwo, ponieważ nie zapewnia on użytecznych odpowiedzi, a jedynie ogólne.

Nazwa modelu	Płynność	Spójność	Naturalność	Perswazyjność	Zaangażo-wanie	Średnia ocena
Seq2Seq + Attention	2,89	3,00	2,76	3,32	3,20	**15,16**
Transformer	2,46	2,52	1,90	2,20	2,20	11,29

Tabela 2 - Wyniki badań, przeprowadzonych na podstawie szacunkowych wskaźników, Seq2Seq + Attention okazał się najlepszy

Biorąc pod uwagę, że nasze modele zostały przeszkolone na zestawie danych z dialogów filmowych, odpowiedzi na postawione pytania przez potencjalnego nabywcy elektroniki, okazały się nie na temat pytania, poza kontekstem. Jeśli zadane zostały proste pytanie konwersacyjne, oba modele próbowały odpowiedzieć zgodnie z kontekstem.

Do szkolenia systemów Pytanie-Odpowiedź, odpowiednich dla procesu biznesowego, dataset dialogów filmowych nie jest jednak odpowiedni, ponieważ:

- kontekst często wykracza poza dialog,
- posiada wiele wyrażeń slangowych,
- nie ma bezpośredniego związku między pytaniami i odpowiedziami.

W przypadku takich zadań lepiej jest używać danych bardziej ukierunkowanych, które są gromadzone w zestawie danych przez konkretną firmę. W tym przypadku, kontekst jest maksymalny i istnieje wyraźny związek między pytaniami, jak i odpowiedziami.

Jeśli zadano rozrywkowe pytania, to Transformer często daje nieco zabawne odpowiedzi i wydaje się dość nieoczekiwany. Ponadto odpowiedzi Transformera są dłuższe, bardziej szczegółowe. W modelu Seq2seq + Attention odpowiedzi są bardziej naturalne. Większość uczestników uznała, że Seq2Seq + attention jest bardziej niezawodnym modelem. Poniżej kilka przykładów rozmów na etapie oceny przez człowieka:

```
--------------------------------------------------
Input:Do you have this type of motherboard in the warehouse?
Output: i do not know
--------------------------------------------------
Input:How fast it can be delivered to the my address?
Output: thomas thomas
--------------------------------------------------
Input:If this processor compare to the lga1155 socket?
Output: i am expecting you george
--------------------------------------------------
Input:What can you propose to buy to make laptop works faster?
Output: i am getting used to be a UNK
--------------------------------------------------
Input:Do you have a offline shop to have opportunity to test it?
Output: yes
--------------------------------------------------
Input:Do you give a guarantee for that hard disk?
Output: i do not know
--------------------------------------------------
Input:Do you have a discount program in your shop?
Output: i am shooting
--------------------------------------------------
Input:Can I sell you my old equipment?
Output: i am looking for you
--------------------------------------------------
Input:If I need to pay for delivery in other country?
Output: i will see you
--------------------------------------------------
Input:Can we negotiate the price?
Output: i will not let it go by day
--------------------------------------------------
```

Rysunek 10.1 - Seq2seq + attention

```
--------------------------------------------------
Input:Do you have this type of motherboard in the warehouse?
Output:  BOS shit you wanna start EOS
--------------------------------------------------
Input:How fast it can be delivered to the my address?
Output:  BOS congratulations EOS
--------------------------------------------------
Input:If this processor compare to the lga1155 socket?
Output:  BOS that is right frank EOS
--------------------------------------------------
Input:What can you propose to buy to make laptop works faster?
Output:  BOS what the fuck is through the walls EOS
--------------------------------------------------
Input:Do you have a offline shop to have opportunity to test it?
Output:  BOS never EOS
--------------------------------------------------
Input:Do you give a guarantee for that hard disk?
Output:  BOS shut up EOS
--------------------------------------------------
Input:Do you have a discount program in your shop?
Output:  BOS my client [UNK] faith [UNK] EOS
--------------------------------------------------
Input:Can I sell you my old equipment?
Output:  BOS [UNK] [UNK] numbers two EOS
--------------------------------------------------
Input:If I need to pay for delievery in other country?
Output:  BOS two million minutes to new york with that EOS
--------------------------------------------------
Input:Can we negotiate the price?
Output:  BOS you have not got that one EOS
--------------------------------------------------
```

Rysunek 10.2 -Transformer

Transformer ma duży potencjał. Jest to model rozrywkowy, o dużym zaangażowaniu. Trzeba nad nim jednak jeszcze popracować. Na przykład, jeśli zastosuje się bardziej specyficzny dataset, eksperymentowanie z hiperparametrami, to prawdopodobnie może on przewyższyć Seq2Seq + Attention.

Podsumowując, Seq2seq + Attention jest lepszy od Transformera zarówno pod względem automatycznych, jak i ludzkich wyników. Jednak, wytrenowany na bazie dialogów filmów, w rzeczywistym biznesie raczej nie miałby zastosowania. Podczas tworzenia chatbota dla procesu biznesowego wybór sekwencji szkoleniowej ma ogromne znaczenie. Projekt więc zakończył się sukcesem, ponieważ wszystkie wymagania zostały spełnione.

11. Podsumowanie

W pracy podano krótki opis informacji teoretycznych na temat chatbotów, a także ich rozwoju. Rodzaje chatbotów zostały szczegółowo omówione. Szczegółowo opisano metodologię i elementy tworzenia owego chatbota. Następnie przedstawiono dogłębny przegląd modeli Seq2Seq. Omówiono różne techniki i architektury, zaproponowane w celu rozszerzenia modelu kodowania i dekodowania oraz uczynienia agentów konwersacyjnych bardziej naturalnymi i ludzkimi. Ponadto, przeprowadzono wstępne eksperymenty w celu szkolenia modelu Transformer i Seq2Seq + Attention na zestawie danych dialogowych. Skuteczność szkoleń analizowano za pomocą wskaźników oceny automatycznej oraz ludzkiej. Przeprowadzony został eksperyment z możliwością wykorzystania wyżej wspomnianych modeli, w celu wykorzystania ich w określonych procesach biznesowych. Na tej podstawie, stwierdzono, że wyszkolone chatboty na podobnym zestawie danych są nieodpowiednimi konsultantami sklepu internetowego. Wreszcie stwierdzono, że potrzebne są dalsze, bardziej szczegółowe eksperymenty, aby ustalić, czy model Transformera jest rzeczywiście gorszy niż model Seq2Seq + Attention oparty na RNN, dla problemu modelowania konwersacyjnego. Oba modele wciąż posiadają wiele możliwości ulepszeń. Odkryliśmy również, że validation loss (błąd na zbiorze sprawdzającym) nie jest dobrym wskaźnikiem dla wczesnego zatrzymania uczenia się. Jest to bardzo jednowymiarowy wskaźnik, nie uwzględniający różnorodności odpowiedzi, które model jest w stanie dostarczyć, co pozostaje jednak niezwykle ważnym atrybutem dobrego zaangażowania.

Ta praca może być przydatna dla naukowców, ponieważ mogą dokładnie zobaczyć, który model jest lepszy dla tego zadania i dlaczego. Mogliby wykorzystać tę pracę do tworzenia bardziej złożonych modeli.

Bibliografia:

1. *10 Leading Language Models For NLP In 2022*, https://www.topbots.com/leading-nlp-language-models-2020/ (dostęp 16.07.2022 r.).

2. Arora Sanjeev, Yingyu Liang, Tengyu Ma, *A SIMPLE BUT TOUGH-TO-BEAT BASELINE FOR SENTENCE EMBEDDINGS*, Princeton University, 2017.

3. Arun Ankit, Soumya Batra, Vikas Bhardwaj, Ashwini Challa, Pinar Donmez, Peyman Heidari, Hakan Inan, Shashank Jain, Anuj Kumar, Shawn Mei, Karthik Mohan, Michael White: *Best Practices for Data-Efficient Modeling in NLG: How to Train Production-Ready Neural Models with Less Data*, 2020.

4. Bahdanau Dzmitry, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate.* In: CoRR.2014.Vol. abs/1409.0473.

5. Bhagwat Vyas Ajay, *Deep Learning for Chatbots* (2018). Master's Projects. DOI: https://doi.org/10.31979/etd.9hrt-u93z.

6. Brownlee Jason, *Deep Learning For Natural Language Processing Machine Learning Mastery*, 2017, s. 1-414.

7. Brownlee Jason, *Gentle Introduction to Generative Long Short-Term Memory Networks,* https://machinelearningmastery.com/gentle-introduction-generative-long-short-term-memory-networks/ (dostęp 11.07.2022 r.).

8. Brownlee Jason, *Long Short-term Memory Networks with Python: Develop Sequence Prediction Models with Deep Learning*, 2017, s. 1-229.

9. Brownlee Jason, *Making Predictions with Sequences,* https://machinelearningmastery.com/sequence-prediction/(dostęp 11.07.2022 r.).

10. *Categorizing and Tagging Words*, https://www.nltk.org/book/ch05.html (dostęp 26.06.2022 r.).

11. Chatbot Market (2021), https://www.marketsandmarkets.com/Market-Reports/smart-advisor-market-72302363.html, (dostęp 24.06.2022 r.).

12. *Chatbots:The Definitive Guide* https://www.artificial-solutions.com/chatbots (dostęp 25.06.2022 r.).

13. Cho Kyunghyun, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio, *Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation.*

14. Chollet François, *English-to-Spanish translation with a sequence-to-sequence Transformer,* 2021 https://keras.io/examples/nlp/neural_machine_translation_with_transformer/(dostęp 03.10.2022 r.).

15. *Cross-language information retrieval* https://en.wikipedia.org/wiki/Cross-language_information_retrieval (dostęp 26.06.2022 r.).

16. *Czym jest chatbot?* https://www.oracle.com/pl/chatbots/what-is-a-chatbot/ (dostęp 24.06.2022 r.).

17. Gohil Dharmi, *A Guide on Chatbots* (2018) https://dzone.com/articles/here-is-a-complete-guide-of-chatbots (dostęp 25.06.2022 r.).

18. Hakkani-Tur Dilek, Gokhan Tur, Asli Celikyilmaz, Yun-Nung Chen,Jianfeng Gao, Li Deng, Ye-Yi Wang, *Multi-Domain Joint Semantic Frame Parsing using Bi-directional RNN-LSTM.*

19. He Yulan, Steve Young, *Semantic processing using the Hidden Vector State model,* https://www.sciencedirect.com/science/article/abs/pii/S0885230804000117 (dostęp 26.06.2022 r.).

20. Hochreiter Sepp, Jürgen Schmidhuber. *Long short-term memory. Neural computation,* 9(8):1735–1780,1997.

21. https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html(dostęp 24.06.2022 r.).

22. *Introduction to Transformers Architecture,* https://rubikscode.net/2019/07/29/introduction-to-transformers-architecture/ (dostęp 11.07.2022 r.).

23. Kriszti Richard. *Deep Learning Based Chatbot Models.* 2019. url: https://arxiv.org/pdf/1908.08835.pdf (dostęp 12.11.2022 r.).

24. Liu Bing, Ian Lane. *Attention-Based Recurrent Neural Network Models for Joint Intent Detection and Slot Filling*, CoRR.2016.Vol. abs/1609.01454, https://arxiv.org/abs/1609.01454.

25. Liu Chia-Wei et al. *How NOT To Evaluate Your Dialogue System: An Empirical Study of Unsupervised Evaluation Metrics for Dialogue Response Generation.* In: CoRR abs/1603.08023 (2016). arXiv: 1603.08023. url: http://arxiv.org/abs/1603.08023.

26. Liu Jiao, Yanling Li, Min Lin, *Review of Intent Detection Methods in the Human-Machine*, Dialogue System, Inner Mongolia Normal University, 2019.
27. Luong Minh-Thang, Hieu Pham, Christopher D. *Manning, Effective Approaches to Attention-based Neural Machine Translation*, 2015.

28. Manzoor Ahtsham, Dietmar Jannachhttps, *Towards retrieval-based conversational recommendation* https://doi.org/10.1016/j.is.2022.102083 (dostęp 26.06.2022 r.).

29. Marietto Maria das Graças Bruno, *Artificial Intelligence MArkup Language*: A Brief Tutorial In: CoRR.2013.Vol. abs/1307.3091. https://arxiv.org/abs/1307.3091 (dostęp 26.06.2022 r.).

30. Nagar Sourabh, *What Is a Chatbot?* (2018) https://dzone.com/articles/3-minute-guide-to-understand-what-is-a-chatbot (dostęp 24.06.2022 r.).

31. Pennington Jeffrey, Socher Richard, Christopher D. Manning - *GloVe: Global Vectors for Word Representation.*

32. Popel Martin and Bojar Ondrej. *Training Tips for the Transformer Model.* In: CoRR abs/1804.00247 (2018). arXiv: 1804.00247. url: http://arxiv.org/abs/1804.00247.
33. Radford Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, *Language Models are Unsupervised Multitask Learners.*

34. Radziwill N.,Benton M.. *Evaluating Quality of Chatbots and Intelligent Conversational Agents.*

35. Reiter Ehud, *An architecture for data-to-text systems*, https://dl.acm.org/doi/10.5555/1610163.1610180 (dostęp 26.06.2022 r.).

36. Singh Nikki, Dr. Sachin Bojewa, *Generative Dialogue System using Neural Network*, 2019 JETIR May 2019, Volume 6, Issue 5, https://www.jetir.org/papers/JETIRCS06034.pdf.

37. Song Yiping, Rui Yan, Xiang Li, Dongyan Zhao, Ming Zhang, *Two are Better than One: An Ensemble of Retrieval- and Generation-Based Dialog Systems.*

38. Sutskever Ilya, Oriol Vinyals, and Quoc V. Le., *Sequence to Sequence Learning with Neural Networks.* In: CoRR. 2014. Vol. abs/1409.3215. url: http://arxiv.org/abs/1409.3215.

39. *Transformer with Python and TensorFlow 2.0 – Attention Layers*, https://rubikscode.net/2019/08/05/transformer-with-python-and-tensorflow-2-0-attention-layers/(dostęp 15.07.2022 r.).

40. *Understanding LSTM Networks*, http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (dostęp 11.07.2022 r.).
41. Vaswani Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin. *Attention Is All You Need,* 2017.

42. Wang Xiaojie, Caixia Yuan, *Recent Advances on Human-Computer Dialogue*, December 2016.

43. Weld H., Huang X., Long S., J. Poon, S. C. Han: *A survey of joint intent detection and slot-filling models in natural language understanding*, CoRR.2021.Vol. abs/2101.08091, https://arxiv.org/abs/2101.08091.

44. Wen Tsung-Hsien, Milica Gasic, Nikola Mrkicc, Pei-Hao Su, David Vandyke and Steve Young: *Semantically Conditioned LSTM-based Natural Language Generation forSpoken Dialogue SystemsTsung*, 2015, s. 1711–1721, Lisbon, Portugal. Association for Computational Linguistics.

45. *Write a Sequence to Sequence (seq2seq) Model*, https://docs.chainer.org/en/stable/examples/seq2seq.html(dostęp 18.08.2022 r.).

Załącznik 1

Wyniki badań

Seq2Seq + Attention

Nr	Płynność	Spójność	Naturalność	Perswazyjność	Zaangażowanie
1	3,7	4,2	2,7	3,3	3,3
2	4,2	4,3	3,5	3,9	4,1
3	2,5	1,9	0,7	2,4	3,6
4	3,3	3,9	2,1	3,1	2,7
5	4,7	4,4	4,1	4,4	4,8
6	1,9	1,3	0,3	1,1	2,1
7	2,9	2,6	2,2	1,9	1,1
8	4,4	3,5	2,4	2,9	4,2
9	4,2	1,9	3,6	4,4	3,2
10	3,7	2,7	2,8	3,8	1,9
11	2,8	2,3	2,2	2,7	1,5
12	4,4	3,7	3,1	4,5	4
13	1,1	0,9	0,5	1,3	1,1
14	3,7	4,5	4,3	2,9	3,1
15	3,3	4,8	4,2	3,9	3,8
16	3,2	3,3	4,1	4,2	4,4
17	2,6	2,8	1,3	3,1	2,5
18	4,1	3,9	2,7	3,7	4,3
19	3,7	3,3	4,6	3,1	3,8
20	4,7	4,6	4,1	4,8	3,9
21	3,5	4,1	2,5	4,3	3,7
	2,89	**3,00**	**2,76**	**3,32**	**3,20**

Średnia ocena: **15.16**
perplexity: **34.0**
f1: 0.35

Przykładowa Rozmowa z konsultantem internetowym sklepu elektronicznego
---
Input:Do you have this type of motherboard in the warehouse?
Output: BOS eventually EOS

Input:How fast it can be delivered to the my address?
Output: BOS none of UNK EOS

Input:If this processor compare to the lga1155 socket?
Output: BOS i am not going to be UNK EOS

Input:What can you propose to buy to make laptop works faster?
Output: BOS UNK EOS

Input:Do you have a offline shop to have opportunity to test it?
Output: BOS i am glad to know EOS

Input:Do you give a guarantee for that hard disk?
Output: BOS eventually EOS

Input:Do you have a discount program in your shop?
Output: BOS goodlooking baron EOS

Input:Can I sell you my old equipment?
Output: BOS i am not talking about it EOS

Input:If I need to pay for delievery in other country?
Output: BOS i am sorry EOS

Input:Can we negotiate the price?
Output: BOS ben EOS

Input:Can we negotiate the price?
Output: BOS ben EOS

Transformer

Nr	Płynność	Spójność	Naturalność	Perswazyjność	Zaangażowanie
1	2,3	2,2	1,5	1,9	1,8
2	3,1	2,7	2,2	2,1	3,8
3	4,1	4,2	3,1	4,4	4,8
4	1,3	0,9	0,4	1,1	0,7
5	2,5	2,2	3,1	2,6	1,9
6	3,3	3,1	2,9	2,6	1,8

7	2,7	3,1	1,2	2,2	1,9
8	3,9	4,6	3,9	4,2	3,7
9	1,1	1,4	1,3	1,1	0,9
10	1,8	1,4	0,2	1,8	1,1
11	3,1	2,4	3,5	2,6	3,2
12	2,7	3,1	2,1	1,7	2,2
13	1,5	1,5	1,8	1,2	1,1
14	0,8	0,6	0,1	0,3	0,7
15	2,2	3,1	1,9	2,4	2,2
16	4,3	3,8	1,5	4,3	3,5
17	2,7	3,6	2,1	2,2	3,1
18	1,9	1,6	0,5	2,1	2
19	2,7	2,3	3,1	2,5	2,1
20	1,3	1,9	2,1	1,2	1,7
21	2,3	3,2	1,4	1,8	2,1
	2,46	**2,52**	**1,90**	**2,20**	**2,20**

Średnia ocena: 11.29
perplexity: 80.8
f1: **0.37**

Przykładowa Rozmowa z konsultantem internetowym sklepu elektronicznego
---
Input:Do you have this type of motherboard in the warehouse?
Output: BOS shit you wanna start EOS

Input:How fast it can be delivered to the my address?
Output: BOS congratulations EOS

Input:If this processor compare to the lga1155 socket?
Output: BOS that is right frank EOS

Input:What can you propose to buy to make laptop works faster?
Output: BOS what the fuck is through the walls EOS

Input:Do you have a offline shop to have opportunity to test it?
Output: BOS never EOS

Input:Do you give a guarantee for that hard disk?
Output: BOS shut up EOS

Input:Do you have a discount program in your shop?
Output: BOS my client [UNK] faith [UNK] EOS

Input:Can I sell you my old equipment?
Output: BOS [UNK] [UNK] numbers two EOS

Input:If I need to pay for delievery in other country?
Output: BOS two million minutes to new york with that EOS

Input:Can we negotiate the price?
Output: BOS you have not got that one EOS
