PDMGR

Budowa ontologii i baz wiedzy w celu zarządzania usługami sieciowymi

Sofiia Levchenko 308996
08/01/2024

Spis treści

Wstęp

Opis pomysłu

Wzięte ze sprawozdania do PPMGR

Zdefiniowanie problemu:

Organizacje zajmujące się dostarczaniem usług telekomunikacyjnych obecnie są istotnie rozwarstwione. Oczekiwania klienta są spełniane poprzez przenoszenie wymagań z jednych warstw do innych. W tym przypadku wymagania biznesowe są najbardziej abstrakcyjne, a wymagania infrastrukturalne - najmniej (z punktu widzenia przełożenia tych wymagań na fizyczne dodawanie, łączenie itp zasobów fizycznych). Niestety często powstaje problem braku precyzyjności przekazywania takich wymagań pomiędzy warstwami, co w skutku przekłada się na wydłużenie czasu realizacji usługi/projektu, jego jakości i wydawanych kosztów.

Hipoteza:

Aby pomyślnie przekazywać wymagania w inteligentny sposób, zbierając wszystkie niezbędne szczegóły i doprecyzowania, mogą zostać wykorzystane algorytmy sztucznej inteligencji. Łącząc możliwości AI ze systemami eksperckimi i narzędziami analitycznymi, można zoptymalizować ten proces automatyzując proces orkiestracji usług, zarówno jak i komunikacji międzyludzkiej (przekazywanie tzw. Intentu “w dół” struktury zarządzania).

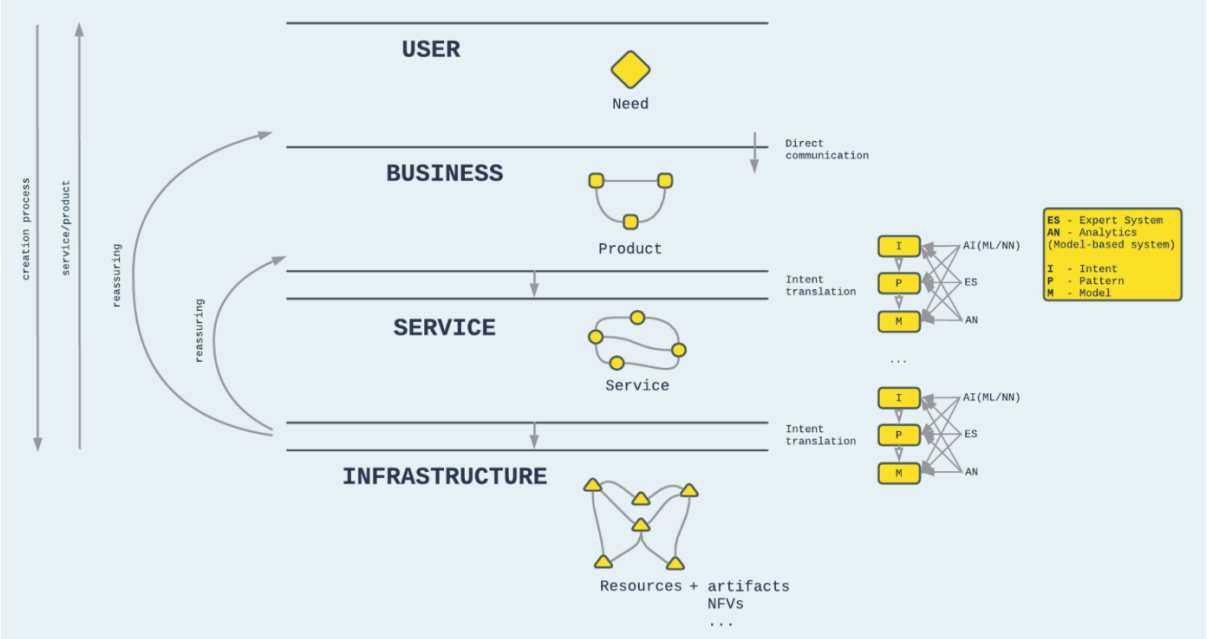
Chcielibyśmy skupić się w tym semestrze na punkcie nr 1 z poniższej listy:

1. Intent, przykładowo sporządzony w formie języka naturalnego, jest przekazywany do modelu sztucznej inteligencji, którego zadaniem jest przetłumaczenie wymagań z języka naturalnego (na poziomie biznesowym) na pewnego rodzaju zbiór cech, zdefiniowanych na poziomie niżej.

2. Zbiór cech następnie jest analizowany pod względem kompletności poprzez próbę wyłapania wzorca serwisu/produktu w danej warstwie. Jeżeli taki zbiór nie jest kompletny, to osoba która wytworzyłą intent, jest “dopytywana” w celu uzupełnienia brakujących danych (do tego celu mogą zostać użyte systemu eksperckie).

3. Następnie dany wzorzec jest uzupełniany o niezbędne dane przez systemy eksperckie i wartości określonych metryk w celu uzyskania modelu produktu. Dane ukształtowane w taki sposób następnie są przekazywane do systemu analitycznego.

4. System analityczny sprawdza prawidłowość danych i następnie przekazuje dane poziom niżej w postaci intentu.



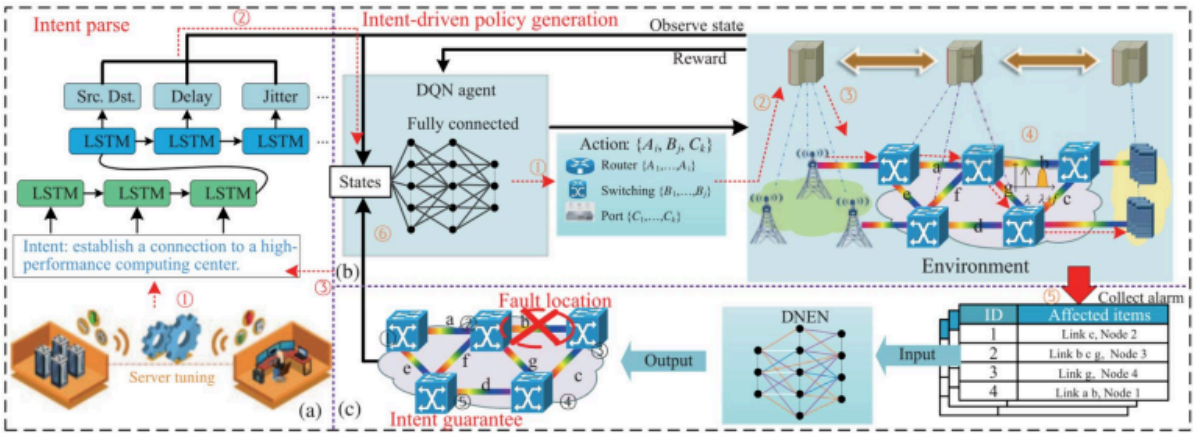
Stan sztuki

Aby zapoznać się z obecnymi podejściami do wdrażania intentów, została przeprowadzona analiza literatury w danym kierunku.

Pierwszym przeanalizowaniu rozwiązaniem była “**Sieć zdefiniowana przez intencję (IDON) z zautomatyzowaną obsługą i utrzymywaniem opartymi na sztucznej inteligencji**”. Autorzy odpowiedniego artykułu zaproponowali sposób na konfigurowanie usług w sieciach optycznych w sposób nieinwazyjny.

Architektura IDON (Intent-defined Optical Network) pozwala na adaptywne generowanie i optymalizację żądanych polityk przy użyciu metod sztucznej inteligencji. Dane rozwiązanie łączy tłumaczenie intentu, generowanie i optymalizację wytwarzanych polityk i zamkniętą pętlę intentu (gwarancja wdrożenia usługi).

W ramach implementacji danego rozwiązania autorzy proponują użycie przetwarzania języka naturalnego do tworzenia baz wiedzy, głębokiego uczenia przez wzmocnienie do konfiguracji polityk spełniających wymagania intentu oraz głębokich ewolucyjnych sieci neuronowych do zapewnienia wdrożenia żądanej usługi.

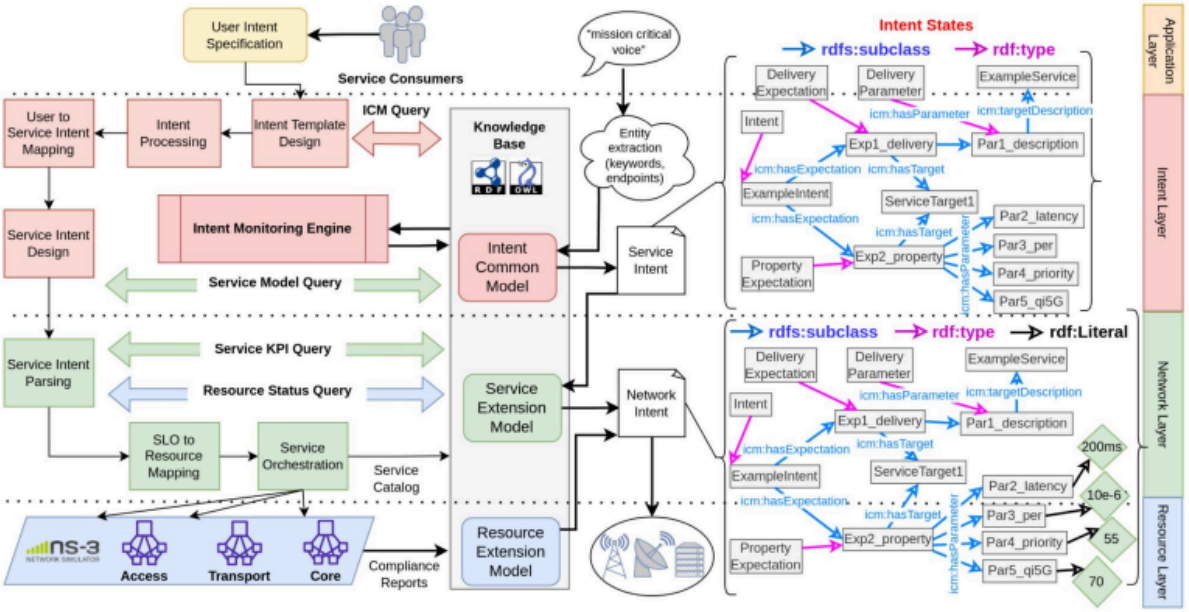


Rys. 2 Architektura IDON

Kolejnym przeanalizowanym rozwiązaniem pod tytułem “Modelowanie intentu oparte na wykorzystaniu zdobytej wiedzy do jego wykorzystania w sieciach nowej generacji” przedstawia próbę implementacji IBN (intent-based networking) poprzez wykorzystanie baz wiedzy do tłumaczenia intentu i następnego wdrożenia serwisu spełniającego wymienione wymagania.

Reprezentacja ontologii intentu jest zapewniona przez wykorzystanie modelu RDF (resource description framework) specyficznego dla domeny w której jest wykorzystywany. W dany sposób jest dokonywane zapewnienie tłumaczenia funkcyjnych i niefunkcyjnych wymagań oraz kontekstu zdefiniowanych w intencie i następne ich przekazywanie do orkiestratora sieci 5G.

W tym przypadku Intent użytkownika jest uzupełniany o dodatkowe parametry niezbędne do definicji określonych serwisów na etapie projektowania serwisu, który następnie jest weryfikowany w warstwie sieci oraz przekazywany do orkiestratora.



Rys. 3: Architektura framework’u IBN przedstawiona w [2]

Ostatnim rozwiązaniem które zostało przeanalizowane był GitHub Copilot. Dane narzędzie jest rozszerzeniem do IDE, które pozwala na automatyczne uzupełnianie kodu pisanego przez programistę oraz tłumaczenie słownego opisu problemu na kod który go rozwiązuje.

To narzędzie jest oparte o model GPT-3 (Codex do którego Microsoft wykupił licencję i który został dostrojony poprzez użycie milionów linijek kodu dostępnego publicznie w repozytoriach GitHub). Używany Codex używa ekstrakcji słów kluczowych i algorytmów głębokiego uczenia do zrozumienia intentu użytkownika.

Niestety dane narzędzie nie można uważać za samodzielne, jako że potrafi generować błędny kod, a zatem potrzebuje weryfikacji przez osobę używającej go.

Metody AI w zakresie LLM

Przygotowanie (pre-processing) danych

Natural Language Processing (NLP) [A Complete Guide] (deeplearning.ai)
Przed tym, jak modelowi AI przekażemy tekst o przetworzenia na rzecz okr**eślonego zadania, tekst musi być przetworzony w celu poprawy wydajności poprzez zamianę słów i znaków na format, który model może zrozumieć**. W tym przetwarzaniu danych mogą być używane różne techniki:

- **Stemming i lematyzacja**:
    - Stemming to nieformalny proces konwertowania słów na ich formy podstawowe przy użyciu *heurystycznych reguł*. Na przykład "uniwersytet," "uniwersytety" i "uniwersytetu" mogą być przypisane do bazy "univers."
    - Lematyzacja to bardziej formalny sposób znajdowania rdzeni, analizując morfologię słowa przy użyciu słownika.
    - Stemming i lematyzacja są dostarczane przez biblioteki takie jak spaCy i NLTK.
- **Segmentacja zdań** dzieli duży fragment tekstu na lingwistycznie znaczące jednostki zdań. Jest to oczywiste w językach takich jak angielski, gdzie koniec zdania jest oznaczony kropką, ale nadal nie jest to trywialne. Kropka może być używana do oznaczenia skrótu, a także do zakończenia zdania, a w tym przypadku kropka powinna być częścią samego skrótu. Proces staje się jeszcze bardziej skomplikowany w językach, takich jak starożytny chiński, które nie mają separatora oznaczającego koniec zdania.
- **Usuwanie słów “zatrzymujących”** ma na celu usunięcie najczęściej występujących słów, które nie dodają wiele informacji do tekstu. Na przykład w języku angielskim to są "the," "a," "an," itp.
- **Znacznikowanie** dzieli tekst na pojedyncze słowa i fragmenty słów. Wynik zazwyczaj składa się z indeksu słowa i tekstu w formacie znaczników, w którym słowa mogą być reprezentowane jako numeryczne tokeny do użycia w różnych metodach uczenia maszynowego. Metoda, która instruuje modele językowe, aby ignorowały nieistotne tokeny, może poprawić efektywność.

Sieci rekurencyjne RNN

Cechą odróżniającą sekwencję od innych typów danych to ważność umieszczenia **elementów danej sekwencji w określonej kolejności,** bo dane w niej zawarte są między sobą powiązane (nie są niezależne). Przykładowym typem danych sekwencyjnych są szeregi czasowe (ceny akcji, nagrania głosowe). Dane tekstowe i sekwencje DNA też są przykładami danych sekwencyjnych.

Modelowanie sekwencji ma dużo różnych zastosowań, i każde z nich charakteryzuje się inną relację między danymi wejściowymi i wyjściowymi:

- **Wiele-do-jednego**: dane wejściowe stanowią sekwencję, ale wyjściowe mają postać wektora/skalara o stałej długości. W analizie sekwencji wejście przyjmuje tekst, a na wyjściu występuje etykieta klasy.

- **Jeden-do-wielu**: dane wejściowe nie są sekwencyjne, w przeciwieństwie do wyjściowych. Mamy z tym do czynienia przy np. podpisywaniu obrazków, gdzie dane wejściowe to obraz, a dane wyjściowe angielskie zdania podsumowujące zawartość obrazu.

- **Wiele-do-wielu**: dane wejściowe i wyjściowe są sekwencjami, w tym przypadku może również zaistnieć synchronizacja pomiędzy danymi sekwencjami (przetwarzanie całości danych, czy część po części i na tej podstawie zwracanie danych wyjściowych).

Omówmy sobie też główny element różniący zwykła sieć neuronową od rekurencyjnej sieci neuronowej. Jest to **mechanizm zapętlania**.

W standardowej sieci jednokierunkowej informacja przepływa z warstwy wejściowej do warstwy ukrytej, a stamtąd do wyjściowej. Z kolei w sieci rekurencyjnej warstwa ukryta otrzymuje informacje zarówno z warstwy wejściowej, tak i z warstwy ukrytej z poprzedniego taktu.

Przepływ informacji w sąsiadujących między sobą taktach w warstwie ukrytej umożliwia sieci zapamiętywanie przeszłych zdarzeń. Jest on najczęściej przedstawiany w formie pętli, w notacji grafowej zwanej **krawędzią rekurencji**.

W standardowej sieci neuronowej każda jednostka ukryta otrzymuje tylko jeden punkt danych wejściowych - pobudzenie całkowite powiązane z warstwą wejściową. Z kolei w sieci rekurencyjnej jednostki ukryte otrzymują dwa oddzielne zestawy danych wejściowych - aktywację wstępną z warstwy wejściowej oraz aktywację z tej samej warstwy wejściowej w poprzednim takcie $t-1$.

W pierwszym takcie $t=0$ jednostki ukryte zostają zainicjalizowane z wartością 0 lub bliską zeru. Następnie w kolejnych taktach zostają do nich przesłane wejściowe punkty danych z bieżącego czasu $x^{(n)}$, a także poprzednie wartości jednostek ukrytych z taktu $t-1$, przedstawionych w postaci zapisu h^{t-1}. W tym przypadku każda warstwa rekurencyjna musi otrzymywać na wejściu sekwencję, dlatego oprócz ostatniej warstwy rekurencyjnej wszystkie pozostałe **muszą zwracać sekwencje na wyjściu**. Konkretny sposób działania sieci rekurencyjnej zależy od rozwiązywanego problemu.

Przejdźmy zatem do zagadnienia **obliczania aktywacji** w sieciach rekurencyjnych. Każda ukierunkowana krawędź w sieci rekurencyjnej jest powiązana z macierzą wag. Wagi te są niezależne od czasu t, zatem są współdzielone po osi czasu. W jednowarstwowej sieci rekurencyjnej mamy do czynienia z następującymi wagami:

- **Wxh**: macierz wag pomiędzy wejściem $x^{(n)}$ a warstwą ukrytą h,
- **Whh**: macierz wag powiązana z krawędzią rekurencyjną,
- **Wxy**: macierz wag pomiędzy warstwami ukrytą i wyjściową.

Obliczanie aktywacji bardzo przypomina ten proces w perceptronach wielowarstwowych i innych typach jednokierunkowych sieciach neuronowych. W przypadku warstwy ukrytej całkowite pobudzenie z_h (aktywacja wstępna) jest wyliczane poprzez kombinację liniową (obliczanie sumy iloczynów macierzy wag z odpowiednimi wektorami z uwzględnieniem jednostki obciążenia):

$$z_h^{(t)} = W_{xh}x^{(t)} + W_{hh}h^{(t-1)} + b_h$$

Następnie pobudzenie jednostek ukrytych w takcie t otrzymujemy w następujący sposób:

$$h^{(t)} = \phi_h(z_h^{(t)}) = \phi_h(W_{xh}x^{(t)} + W_{hh}h^{(t-1)} + b_h)$$

Tutaj b_h jest wektorem obciążenia dla jednostek ukrytych, a $\phi_h(.)$ stanowi funkcję aktywacji warstwy ukrytej. Po obliczeniu aktywacji jednostek ukrytych w bieżącym takcie możemy przystąpić do obliczania aktywacji jednostek wyjściowych:

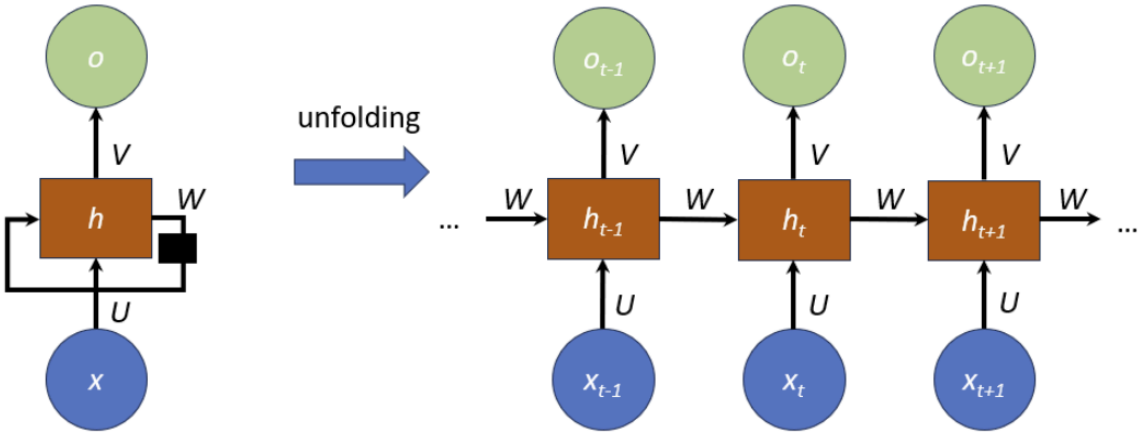
$$o^{(t)} = \phi_o(W_{ho}h^{(t)} + b_o)$$

Uczenie sieci natomiast można przeprowadzić poprzez użycie algorytmu BPTT (Backpropagation through time).

Warto też wspomnieć że w sieciach RNN rekurencja może być zaimplementowana nie w warstwie ukrytej, a w warstwie wyjściowej. W takim przypadku pobudzenia całkowite z warstwy wyjściowej w poprzednim takcie,o^{t-1}, można dokonać na dwa sposoby:

- do warstwy ukrytej w bieżącym takcie, h^t (rekurencja z wyjścia-do-ukrytej)
- do warstwy wyjściowej w bieżącym takcie, o^t (rekurencja z wyjścia-do-wyjścia)

W tym przypadku nazewnictwo macierzy wag dla warstw wyjściowych jest podobne do nazewnictwa w przypadku rekurencji w warstwie ukrytej. Czasem do określenia wag powiązanym z połączeniami rekurencyjnymi są określane jako W_{rec}.



Rys. 4: Diagram sieci RNN wraz z rozwinięciem sieci

Jak pokazano na Rysunku, graf RNN składa się z wejścia x, ukrytych połączeń h i wyjścia o. Połączenia wejściowe do ukrytych są parametryzowane przez wagi U, połączenia ukryte do ukrytych są parametryzowane przez wagi W , a połączenia ukryte do wyjściowych przez wagi V. Czarny kwadrat reprezentuje pojedynczy krok. Po rozwinięciu grafu RNN można zauważyć, że RNN składa się z tych samych bloków obliczeniowych, z których każdy reprezentuje stan w danym czasie t. Wagi są współdzielone we wszystkich krokach czasowych, dzięki czemu informacje wyciągnięte z poprzednich stanów są przechowywane przez następujące kroki.

Głównym problemem w RNN staje się uczenie zależności w długich odstępach czasowych (ang. **long-term dependencies**). Z powodu użycia algorytmu wstecznej propagacji w czasie, podczas obliczania gradientów funkcji starty pojawia się problem tzw. **zanikających i eksplodujących** gradientów. W praktyce istnieją co najmniej trzy rozwiązania tego problemu:

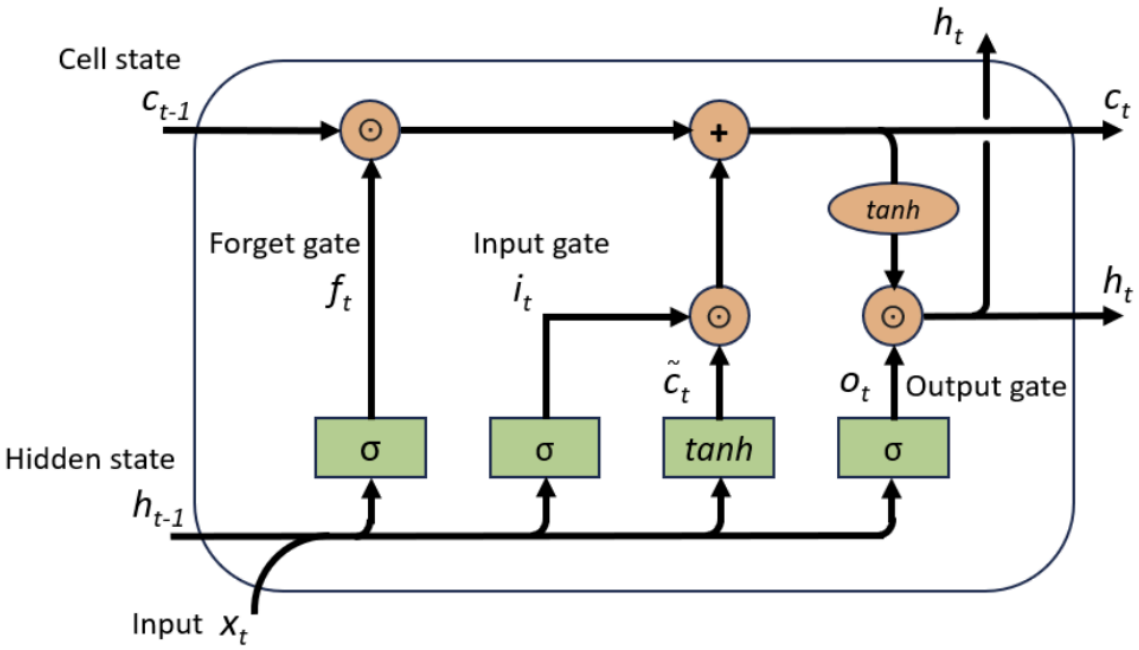
- obcinanie gradientów
- algorytm ograniczonej propagacji wstecznej w czasie (alg. TBPTT)
- komórka długiej pamięci krótkofalowej

Za pomocą obcinania gradientów wyznaczamy wartość odcięcia (progową) gradientów i wyznaczamy ją gradientem, które przekraczają tę wartość. Z kolei algorytm TBPTT ogranicza jedynie liczbę taktów, podczas których sygnał może być przekazywany wstecz po każdym przebiegu w przód. Rozwiązaniem tego problemu staje się sieć LSTM, która okazuje się skuteczniejsza w modelowaniu sekwencji długofalowych poprzez rozwiązanie problemu znikających gradientów

Sieć LSTM

Głównym składnikiem LSTM (ang. *Long-Short Term Memory*) jest komórka pamięci, która w istocie reprezentuje lub zastępuje warstwę ukrytą standardowej sieci rekurencyjnej. W każdej komórce pamięci występuje krawędź rekurencji mająca pożądana wagę $w = 1$, służącą przezwyciężaniu problemów z zanikającymi/eksplodującymi gradientami. Wartości powiązane z tą krawędzią rekurencji noszą nazwę **stanu komórki**. Wagi z

różnych warstw przetwarzane są przy użyciu tzw. **bramek**, na zasadzie przeprowadzania operacji sumowania albo mnożenia odpowiednich elementów i następnego przepuszczania ich przez pewne funkcje aktywacji.



Rys. 5: Diagram sieci LSTM

Architektura sieci LSTM z rysunku wyżej składa się z bramek, warstw typu "fully-connected" (oznaczonych jako zielone prostokąty) z funkcjami aktywacji oraz stan wewnętrzny komórki. Małe litery: x, h, c, f reprezentują wektory. Litery t oraz $t-1$ odpowiednio obecny stan czasowy oraz poprzedni stan czasowy.

W komórce LSTM występują trzy rodzaje bramek:

- **Bramka zapominająca** (ang. forget gate; f_t)
- **Bramka wejściowa** (ang. input gate; i_t) określa, ile nowej informacji ma być dodanej do obecnej wartości komórki c_t
- **Bramka wyjściowa** (ang. output gate; o_t) określa, która część komórki c_t ma być podana na wyjście (oraz jaka jej część ma służyć jako wejście do kolejnego stanu warstwy ukrytej h_t)

Każda z tych komórek działa na zasadzie używania odpowiednich wzorów matematycznych i następnie przepuszczanie uzyskanej wartości przez ustaloną funkcję aktywacji.

Sieci LSTM zdobyły uznanie w wielu zadaniach wymagających przetwarzania sekwencji danych. Innym typem sieci LSTM są sieci Bi-LSTM.

Główną ideą tej sieci jest uczenie się długoterminowych zależności od przeszłych i przyszłych stanów, w przeciwieństwie do sieci LSTM, w której brane są pod uwagę tylko przeszłe warunki. Idea ta została wprowadzona w dwukierunkowej rekurencyjnej sieci

neuronowej (BRNN), w której niektóre neurony stanu zostały podzielone na dwie części: jedną odpowiedzialną za stany przyszłe, a drugą za stany przeszłe.

W zadaniach NLP, takich jak modelowanie języka, rozumienie języka, tłumaczenie maszynowe i tagowanie POS, sieci BLSTM osiągnęły doskonałą wydajność.

Transformatory

Nie tak dawno temu wyłonił się nowy typ architektury który w niektórych zadaniach przetwarzania języka naturalnego uzyskuje lepsze rezultaty od modeli seq2seq bazujących na sieciach rekurencyjnych. Jest to tzw. **struktura transformatora**. Możemy za jej pomocą modelować globalne zależności pomiędzy sekwencjami wejściowymi a wyjściowymi.

Architektura transformatora wykorzystuje **mechanizm uwagi**, a dokładniej **samouwagi**. W naszym przypadku użycie mechanizmu uwagi oznaczałoby, że nasz model uczyłby się koncentrować na tych fragmentach sekwencji wejściowej, które mają znaczenie w przekazywaniu emocji.

Najpierw rozpatrzmy **podstawową** wersję mechanizmu samouwagi. Przyjmijmy, że dysponujemy sekwencja wejściową o długości T, a tę sekwencję wyjściową o podobnej długości. W takim przypadku celem mechanizmu uwagi w zadaniu seq2seq jest modelowanie zależności pomiędzy każdym elementem sekwencji wyjściowej, a elementami w sekwencji wyjściowej. W tym celu mechanizmy uwagi zostają podzielone na trzy etapy:

- Określanie wagi istotności na podstawie podobieństwa pomiędzy danym elementem a pozostałymi elementami sekwencji;
- Normalizowanie wag (co zazwyczaj wiąże się z użyciem funkcji softmax);
- Wykorzystanie wag wraz z odpowiednimi elementami sekwencji w celu obliczenia wartości uwagi;

A więc wynikiem mechanizmu samouwagi jest suma ważona wszystkich sekwencji wejściowych. Dla i-tego elementu wejściowego wartość na wyjściu jest obliczana następująco:

$$o^{(i)} = \sum_{j=0}^{T} W_{ij} x^{(j)}$$

Tutaj wagi W_{ij} obliczane są na podstawie podobieństwa pomiędzy bieżącym elementem wejściowym, a wszystkimi pozostałymi elementami sekwencji wejściowej. **Podobieństwo** zatem jest obliczane jako iloczyn skalarny pomiędzy bieżącym elementem wejściowym $x^{(i)}$ a innym elementem w sekwencji wejściowej, $x^{(j)}$:

$$\omega_{ij} = x^{(i)^T} x^{(j)}$$

Po obliczeniu tych wag bazujących na podobieństwie dla i-tego elementu i wszystkich elementów sekwencji (od $x^{(i)}$ do x^{T}), "nieprzetworzone" wagi są następnie normalizowane za pomocą funkcji softmax:

$$W_{ij} = \frac{exp(\omega_{ij})}{\sum_{j=0}^{T} exp(\omega_{ij})} = softmax([\omega_{ij}]_{j=0..T})$$

Suma wag w tym przypadku wynosi 1.

W bardziej zaawansowanej odmianie mechanizmu samouwagi, wprowadzane są dodatkowe trzy macierzy wag, które można w czasie uczenia modelu można dopasowywać jako jego parametr. Macierze te mają symbole U_g,U_k,U_v. a ich pomocą rzutujemy dane wejściowe w elementy sekwencji zwane **kwerendą**, **kluczem** i **wartością**:

- sekwencja kwerendy: $q^{(i)} = U_q x^{(i)}$ dla $i \in [0, T]$,
- sekwencja klucza: $k^{(i)} = U_k x^{(i)}$ dla $i \in [0, T]$,
- sekwencja wartości: $v^{(i)} = U_v x^{(i)}$ dla $i \in [0, T]$,

W tym przypadku nie obliczamy nieznormalizowanej wagi jako iloczynu skalarnego pomiędzy danym elementem sekwencji wejściowej $x^{(i)}$ a j-tym elementem sekwencji $x^{(j)}$, lecz policzymy iloczyn skalarny kwerendy i klucza:

$$\omega_{ij} = q^{(i)^T} K^{(j)}$$

Możemy wykorzystać m, a dokładniej $\frac{1}{\sqrt{m}}$, aby przeskalować wagę przed jej znormalizowanym za pomocą funkcji softmax:

$$W_{ij} = softmax(\frac{\omega_{ij}}{\sqrt{m}})$$

Dzięki przeskalowaniu wag długość euklidesowa wektorów wag będzie mieścić się w przybliżeniu w jednakowym zakresie.

Kolejną "sztuczką" usprawniającą mechanizm samouwagi jest **wieloblokowy mechanizm uwagi**. W tym przypadku każdy mechanizm samouwagi jest nazywany **blokiem** i każdy z nich jest równolegle obliczany. Każdy z **r** równoległych bloków tworzy wektor **h** o rozmiarze m. Wektory te są następnie łączone w wektor **z** o rozmiarze **r*m**. Na koniec połączony wektor jest rzutowany za pomocą macierzy wyjściowej W^o, co pozwala uzyskać wyniki:

$$o^{(i)} = W^o_{ij} z$$

Struktura transformatora między innymi też zawiera połączenie rezydualne, które dodaje wynik z warstwy do jej wejścia, czyli *x+warstwa(x)*. Blok składający się z warstwy i takie połączenie rezydualne jest nazywane blokiem rezydualnym. Również warto wspomnieć taki element jak normalizację warstwy, który dokonuje skalowania wejść i aktywacji w każdej warstwie sieci neuronowej.

A więc, na początku sekwencja wejściowa jest przekazywana do warstw MHA (składa się z wieloblokowego mechanizmu uwagi). Ponadto, te warstwy są dodawane do wyjścia warstw MHA za pomocą połączenia rezydualnego (wcześniejsze warstwy będą otrzymywać wystarczająco mocne sygnały gradientu) w czasie uczenia.

Po dodanie sekwencji wejściowych do wyjścia warstw MHA, rezultaty są normalizowane za pomocą normalizacji warstwy. Te znormalizowane sygnały przechodzą następnie przez szereg perceptronów wielowarstwowych (w pełni połączonych), w których również występuje połączenie rezydualne. Na koniec wynik bloku rezydualnego zostaje ponownie znormalizowany i zwrócony jako sekwencja wyjściowa; możemy go używać do klasyfikowania (np. analizy sentymentu) lub generowania sekwencji.

Model BERT

Kluczową innowacją techniczną **BERT** (ang. Bidirectional Encoder Representations from Transformers) jest zastosowanie dwukierunkowego uczenia (przy użyciu Masked LM) transformera do modelowania języka. Taka metoda pozwala na lepsze zrozumienie kontekstu językowego i jego "przepływu" w porównaniu do modeli jednokierunkowych.

Transformer, jak już zostało to omówione wcześniej, jest używany do rozpoznawania powiązania między słowami (kontekstu). W formie podstawowej transformer składa się zarówno z enkodera (do zrozumienia danych tekstowych), tak i z dekodera (do tworzenia predykcji). W przypadku BERT'a niezbędnym jest wykorzystanie jedynie enkodera.

W przeciwieństwie do modeli kierunkowych, które odczytują wprowadzany tekst w sposób sekwencyjny (od lewej strony do prawej lub od prawej do lewej), Transformer odczytuje całą sekwencję słów na raz. Dlatego uważa się, że jest dwukierunkowy, chociaż dokładniej byłoby powiedzieć, że jest bezkierunkowy. Ta cecha pozwala modelowi poznać kontekst słowa na podstawie całego jego otoczenia (po lewej i prawej stronie słowa).

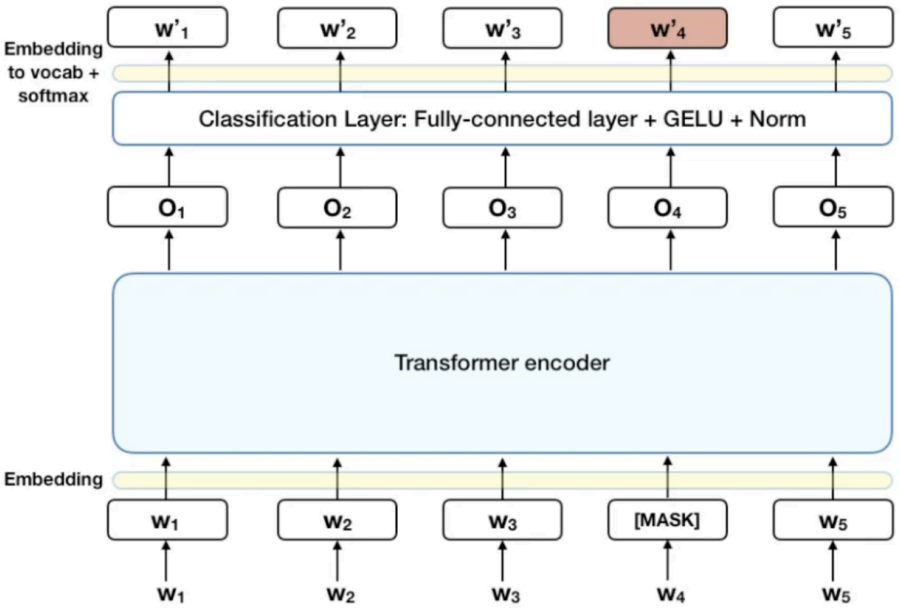
Dane wejściowe w enkoderze Transformera to ciąg znaczników, które najpierw umieszczanie są w wektorach, a następnie przetwarzane w sieci neuronowej. Dane wyjściowe natomiast są ciągiem wektorów o rozmiarze H, w którym każdy wektor odpowiada znacznikowi wejściowemu o tym samym indeksie.

W trakcie uczenia model BERT używa dwóch strategii:

Masked ML

Przed wprowadzeniem sekwencji słów do BERT, 15% słów w każdej sekwencji zostaje zastąpionych znacznikiem [MASK]. Następnie model próbuje przewidzieć pierwotną wartość zamaskowanych słów na podstawie kontekstu dostarczonego przez inne, niezamaskowane słowa w sekwencji. Z technicznego punktu widzenia przewidywanie słów wyjściowych wymaga:

- Dodanie warstwy klasyfikacji na wyjściu kodera.
- Mnożenie wektorów wyjściowych przez macierz osadzania, przekształcanie ich w "wymiar słownictwa" (ang. vocabulary dimension).
- Obliczanie prawdopodobieństwa każdego słowa w słowniku za pomocą softmax.



Rys. 6: Przepływ danych w Masked ML

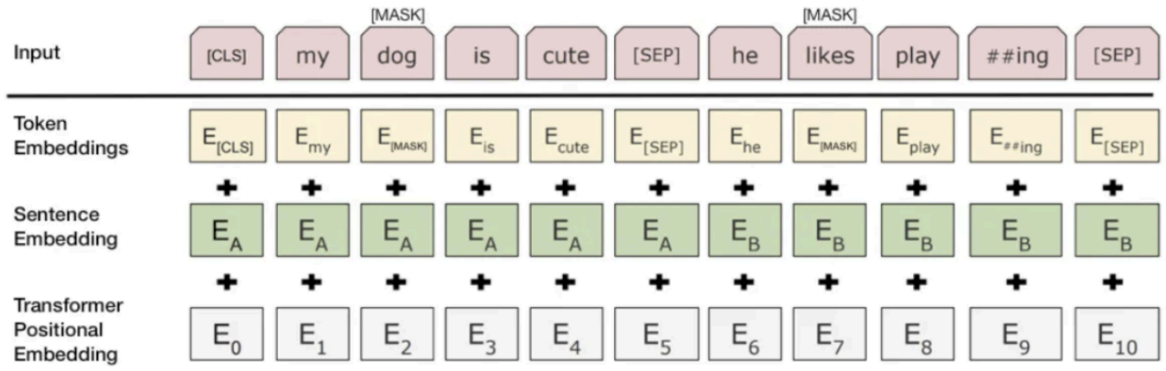
Funkcja straty BERT uwzględnia jedynie przewidywanie wartości zamaskowanych i ignoruje przewidywanie słów nie maskowanych.

Przewidywanie następnego zdania (NSP)

W procesie uczenia BERT model otrzymuje pary zdań jako dane wejściowe i uczy się przewidywać, czy drugie zdanie w parze jest kolejnym zdaniem w oryginalnym dokumencie. Podczas uczenia 50% wejść stanowi para, w której drugie zdanie jest kolejnym zdaniem w oryginalnym dokumencie, natomiast w pozostałych 50\% jako drugie zdanie wybierane jest losowe zdanie z korpusu. Zakłada się, że losowe zdanie zostanie odłączone od pierwszego zdania.

Aby pomóc modelowi rozróżnić dwa wybrane zdania podczas uczenia, dane wejściowe przed wprowadzeniem do modelu przetwarzane są w następujący sposób:

- Na początku pierwszego zdania umieszcza się token [CLS], a na końcu każdego zdania token [SEP].
- Do każdego znacznika dodawana jest dodatkowa informacja wskazująca na przynależność do Zdania A lub Zdania B.
- Do każdego znacznika dodawana jest informacja o pozycji, aby wskazać jego położenie w sekwencji.



Rys. 7: Przetwarzanie danych wejściowych w BERT

Aby przewidzieć, czy drugie zdanie rzeczywiście jest powiązane z pierwszym, wykonywane są następujące kroki:

- Cała sekwencja wejściowa przechodzi przez model transformatora.
- Dane wyjściowe tokena [CLS] są przekształcane w wektor w kształcie 2×1 przy użyciu prostej warstwy klasyfikacyjnej (wyuczone macierze wag i odchyleń).
- Obliczanie prawdopodobieństwa *IsNextSequence* za pomocą algorytmu softmax.

Podczas uczenia modelu BERT, maskowane LM i przewidywanie następnego zdania są trenowane razem, a celem jest minimalizacja funkcji straty obu strategii.

BERT może zostać użyty do różnorodnych zadań językowych, poprzez dodanie jedynie niewielkiej warstwę do modelu podstawowego:

- Zadania **klasyfikacyjne**, takie jak analiza tonacji, są wykonywane podobnie do klasyfikacji następnego zdania, poprzez dodanie warstwy klasyfikacji na wyjściu transformatora dla znacznika [CLS].
- W zadaniach dotyczących przewidywania **odpowiedzi na pytania** program otrzymuje pytanie dotyczące sekwencji tekstowej i za zadanie ma zaznaczenie odpowiedzi w sekwencji wyjściowej. Korzystając z BERT, można wytrenować model Q&A, dodając do procesu uczenia dwóch dodatkowych wektorów oznaczających początek i koniec odpowiedzi.
- W przypadku **rozpoznawania podmiotów (NER)** model otrzymuje sekwencję tekstu w której musi do zaznaczyć różne typy podmiotów (osoba, organizacja, data itp.), które pojawiają się w tekście. Za pomocą BERT można wytrenować model NER, wprowadzając wektor wyjściowy każdego znacznika do warstwy klasyfikacyjnej, która przewiduje etykietę NER.

(Dodatkowo) XAI - Explainable AI

Explainable Artificial Intelligence (XAI) to obszar badawczy i technologiczny, który skupia się na rozwijaniu metod i narzędzi, które pozwalają na zrozumienie i wyjaśnienie procesów podejmowania decyzji przez sztuczną inteligencję. W kontekście sztucznej inteligencji,

zwłaszcza w zaawansowanych modelach uczenia maszynowego, takich jak głębokie sieci neuronowe, **wyjaśnialność jest często wyzwaniem**.

Głównym celem XAI jest uczynienie sztucznej inteligencji bardziej transparentną i zrozumiałą dla ludzi. W wielu przypadkach, zwłaszcza w zastosowaniach krytycznych, istotne jest, aby **użytkownicy, decydenci** i inne zainteresowane strony **mogły zrozumieć, dlaczego system sztucznej inteligencji podjął określone decyzje**.

W ramach XAI opracowuje się różne techniki, takie jak:

- Lokalne wyjaśnienia: Obejmuje to dostarczanie wyjaśnień dla konkretnych decyzji podjętych przez model, co umożliwia zrozumienie, dlaczego dany wynik został osiągnięty.
- Globalne wyjaśnienia: Te techniki skupiają się na zrozumieniu ogólnej struktury i działania modelu, co pozwala na uzyskanie szerszej perspektywy na jego funkcjonowanie.
- Proste modele interpretowalne: Opracowuje się modele, które są bardziej zrozumiałe dla ludzi, a jednocześnie dobrze reprezentują zależności w danych.

XAI ma znaczące znaczenie w kontekście zaufania do sztucznej inteligencji, zwłaszcza w obszarach, gdzie decyzje systemów AI mają wpływ na życie ludzi, takich jak opieka zdrowotna, finanse czy bezpieczeństwo publiczne. Uczenie maszynowe i sztuczna inteligencja stają się bardziej użyteczne i etyczne, gdy są w pełni zrozumiałe i potrafią udzielać jasnych wyjaśnień swoich działań.

W tym kontekście można wspomnieć kilka, narzędzi które można użyć mając na celu zwiększyć wyjaśnialność modeli sztucznej inteligencji:

- **InterpretML** to otwarta biblioteka w języku Python, która ma na celu dostarczanie narzędzi do interpretowalności modeli uczenia maszynowego. Biblioteka ta umożliwia użytkownikom zrozumienie i analizę prognoz generowanych przez modele. InterpretML oferuje szereg funkcji, w tym techniki interpretowalności lokalnej, pozwalając na zrozumienie decyzji modelu dla konkretnych przypadków, a także narzędzia do analizy interpretowalności globalnej, ułatwiając zrozumienie ogólnego działania modelu.

- **XAI360** to pojęcie, które może odnosić się do szeroko zakrojonej strategii lub platformy związanej z Wyjaśnialną Sztuczną Inteligencją (XAI). Mimo braku konkretów, nazwa sugeruje kompleksowe rozwiązanie obejmujące różne aspekty interpretowalności w kontekście sztucznej inteligencji.

Grafy wiedzy (Knowledge Graphs)

A guide to the Knowledge Graphs | by Mohit Mayank | Aug, 2021 | Towards Data Science | Towards Data Science

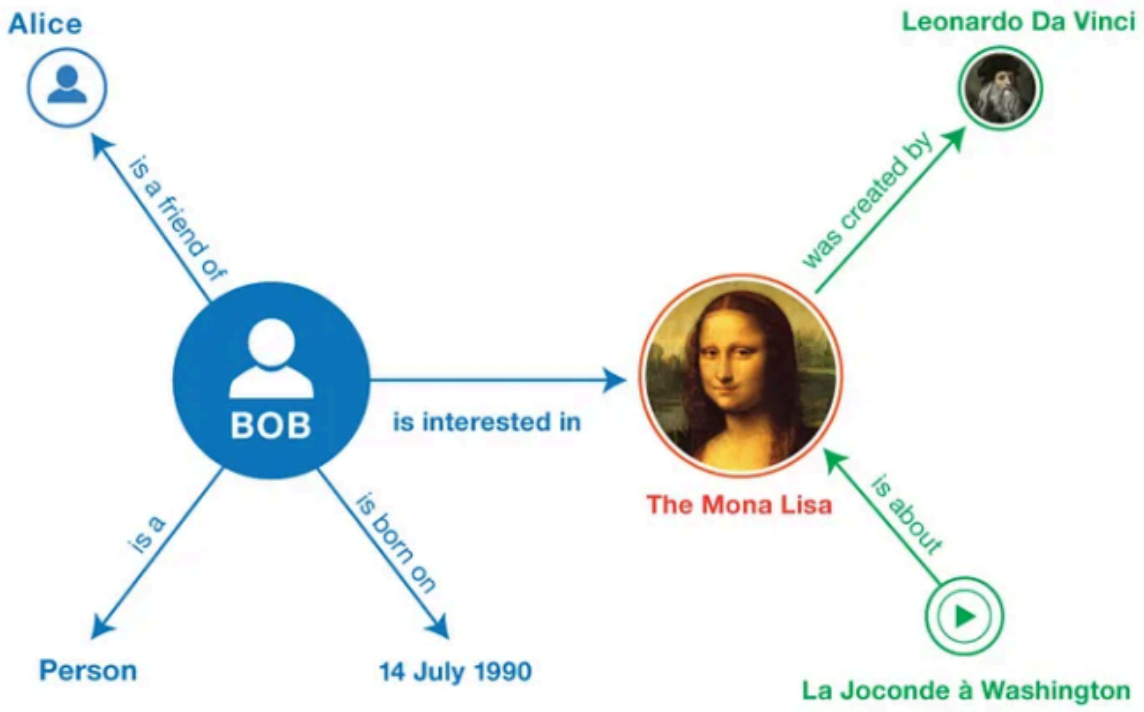
Wstęp

Aby lepiej zrozumieć Grafy Wiedzy, zacznijmy od zrozumienia jego podstawowej jednostki, czyli "faktu". Fakt stanowi najbardziej podstawowy element informacji przechowywanej w GW i może być przedstawiony jako trójka na jeden z dwóch sposobów:

- **HRT**: <głowa, relacja, ogon>
- **SPO**: <podmiot, orzeczenie, obiekt>

W przypadku HRK fakty składają się z trzech istotnych elementów (stąd nazwa trójki), co przyczynia się do intuicyjnego przedstawienia GW jako grafu:

- **Głowa lub ogon**: To są jednostki reprezentujące obiekty rzeczywistego świata lub abstrakcyjne koncepcje, widziane jako węzły.
- **Relacja**: To są połączenia między jednostkami, reprezentowane jako krawędzie.



Rys. 8: Przykład grafu wiedzy

Nie ma ograniczeń co do typu danych faktów przechowywanych w grafie wiedzy. Jak pokazano powyżej, w GW mogą być zawarte informacje o osobach (Bob, Alice, itp.), dziełach sztuki (Mona Lisa), datach itp., z każdym z tych elementów reprezentowanych jako węzły w grafie.

Grafy Wiedzy mogą być wykorzystywane do wielu zadań - czy to do **wnioskowania logicznego**, udzielania zrozumiałych rekomendacji, **prowadzenia kompleksowych analiz**, czy po prostu do lepszego przechowywania informacji.

W praktyce

Istnieje wiele małych specjalistycznych Grafów Wiedzy stworzonych prywatnie w celach wykonania ściśle określonych zadań. Również są ogromne ogólnodostępne Grafy Wiedzy, zawierające fakty wszelkich form. Kilka znanych i dostępnych (open-source) grafów wiedzy to:

- **DBpedia**: jest to projekt (crowd-sourced community-based), mający na celu ekstrakcję strukturalnych treści z informacji dostępnych w różnych projektach Wikimedia.
- **Freebase**: to ogromna, wspólnie tworzona i zmieniana na bieżąco baza połączonych danych. Znana jako „otwarta baza danych wiedzy świata". Została kupiona przez Google i używana do zasilania własnego Grafu Wiedzy.
- **OpenCyc**: narzędzie pozwalające na korzystanie z Cyc, jednego z najbardziej kompletnych na świecie ogólnych źródeł wiedzy i silników rozumowania opartego na zdrowym rozsądku.
- **Wikidata**: to darmowa,, wielojęzyczna baza danych, zbierająca dane strukturalne w celu wsparcia projektów Wikimedia.
- **YAGO**: ogromna semantyczna baza wiedzy, pochodząca z Wikipedii, WordNet i GeoNames.

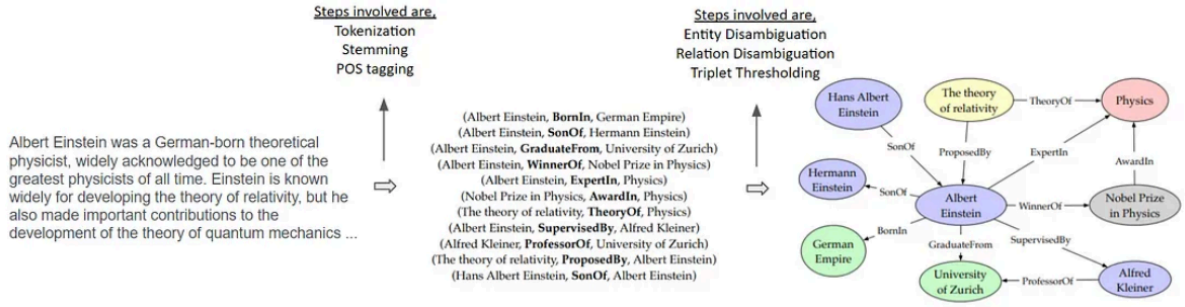
Mimo istnienia wielu otwartoźródłowych Grafów Wiedzy, możemy mieć potrzebę stworzenia Grafu Wiedzy specyficznego dla naszego obszaru zastosowań. W naszym przypadku, **dane podstawowe** (z których chcemy stworzyć Graf Wiedzy) mogą mieć różne typy - *tablicowe, graficzne lub tekstowe bloki*.

Omówmy zatem kilka kroków pokazujących jak stworzyć Graf Wiedzy *z danych nieustrukturyzowanych*, takich jak **tekst**, ponieważ jest **stosunkowo łatwiej przekształcić dane strukturalne w Graf Wiedzy**, korzystając z minimalnej wiedzy dziedzinowej i skryptów. Cały proces można podzielić na dwie główne kategorie:

1. **Tworzenie faktów**:
    a. To jest pierwszy krok, w którym *analizujemy tekst (zdanie po zdaniu*) i **wydobywamy fakty w formie trójek <H, R, T>**. Ponieważ przetwarzamy tekst, możemy *skorzystać z etapów wstępnego przetwarzania, takich jak tokenizacja, stemming lub lematyzacja,* itp., aby oczyścić tekst.
    b. Następnie chcemy **wydobyć jednostki i relacje (fakty) z tekstu.** Dla jednostek możemy użyć *algorytmów rozpoznawania jednostek nazwanych (NER).* Dla relacji możemy użyć *technik analizy składniowej zdań,* aby znaleźć związek między dowolną parą jednostek.
2. **Wybór faktów**:
    a. Po wydobyciu wielu faktów, kolejnym oczywistym krokiem może być **usunięcie duplikatów i zidentyfikowanie istotnych faktów**, które można

dodać do Grafu Wiedzy. Aby zidentyfikować duplikaty, możemy użyć *technik dysambiguacji jednostek i relacji.* Pomysłem jest skonsolidowanie tych samych faktów lub elementów faktu w przypadku powtórzeń.

b. Wreszcie, możemy mieć kompleksowy system oparty na regułach, który decyduje, który potrójnik powinien zostać dodany do Grafu Wiedzy, a który można pominąć na podstawie takich czynników jak redundantne informacje (`A → rodzeństwo → B` jest obecny, więc `B → rodzeństwo → A` jest zbędne) lub nieistotne informacje.



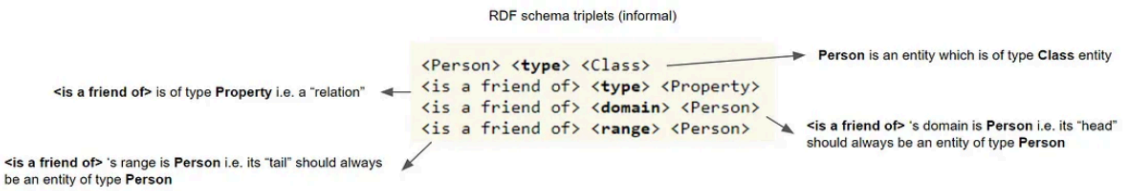
Rys. 9: Kroki niezbędne do stworzenia grafu wiedzy

Ontologia grafów wiedzy

Ontologia to model świata (praktycznie tylko jego podzbioru) który zawiera informację o rodzaju jednostek, relacjach ich łączących oraz ograniczeniach dotyczących informacji o tym jak jednostki i relacje mogą być połączone.

W pewnym sensie **ontologia definiuje zasady, według których jednostki są połączone w obrębie świata**.

Resource Description Framework (RDF) oraz **Web Ontology Language (OWL)** to niektóre z framework'ów językowych używanych do modelowania ontologii. Zapewniają one wspólną platformę do wyrażania tych informacji, dzięki czemu mogą być one *wymieniane między aplikacjami* bez utraty znaczenia.



Rys. 10: Schemat trójek używanych w RDF Schema

RDF pozwala na tworzenie ontologii przy pomocy tzw. języków.

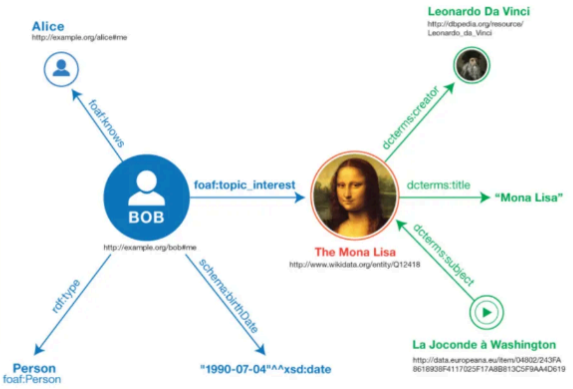
Na Rys. 11 można zobaczyć skrypt tworzenia Grafu Wiedzy (po lewej) w języku Turtle dla KG (po prawej).

```
BASE   <http://example.org/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX schema: <http://schema.org/>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX wd: <http://www.wikidata.org/entity/>

<bob#me>
    a foaf:Person ;
    foaf:knows <alice#me> ;
    schema:birthDate "1990-07-04"^^xsd:date ;
    foaf:topic_interest wd:Q12418 .

wd:Q12418
    dcterms:title "Mona Lisa" ;
    dcterms:creator <http://dbpedia.org/resource/Leonardo_da_Vinci> .

<http://data.europeana.eu/item/04802/243FA8618938F4117025F17A8B813C5F9AA4D619>
    dcterms:subject wd:Q12418 .
```



Rys. 11: Skrypt napisany w języku Turtle w celu stworzenia przykładowego grafu wiedzy

Na początku skryptu **tworzymy odniesienia do wielu predefiniowanych ontologii** (aby ułatwić proces tworzenia przykładu). Następnie, aby stworzyć fakty (lub trójki) Grafu Wiedzy, możemy postępować zgodnie z liniami komend PREFIX.

Każda jednostka i relacja mają **unikalny identyfikator** (unikalny klucz lub UID). W całym kodzie ta sama jednostka lub relacja powinna być odwoływana przez ten sam UID.

Następnie, korzystając z predefiniowanych schematów, **możemy dodawać fakty dla jednostki** (w terminach graficznych, *dodajemy łączące krawędzie*). Fakty te mogą obejmować inną jednostkę (odwołujemy się do niej za pomocą jej UID), pewien tekst, datę (w formacie DateTime), linki, itp.

Ostatecznie, ten skrypt zawiera kompletny schemat i definicję Grafu Wiedzy. Plik ten następnie może być importowany do dowolnej bazy danych Grafu Wiedzy dla wizualizacji i efektywnego odpytywania.

Istnieją dwa rodzaje baz danych, których można używać do przechowywania informacji o grafach. Pierwszym z nich są "**grafy właściwości**" takie jak **Neo4j** i **OrientDB**, które *nie obsługują plików RDF* (od razu) i *posiadają własny niestandardowy język zapytań*. Z drugiej strony mamy "składowiska trójek RDF", które obsługują pliki RDF i wspierają język zapytań tak jak **SPARQL**, powszechnie używany do zapytań w Grafach Wiedzy. Niektóre z najbardziej znanych to (z wersją open source):

- **GraphDB**: rozwiązanie firmy Ontotext, które dostarcza usługi front-end (wizualizacji) i back-end (serwera) do przeglądania i zapytywania grafów wiedzy.
- **Virtuoso**: rozwiązanie firmy OpenLinkSoftware, które dostarcza usługi backend do zapytań w przechowywanych Grafach Wiedzy. Obsługuje również zapytania w KG za pomocą kombinacji SQL i SPARQL. Na Virtuoso hostowane są również wiele otwartoźródłowych Grafów Wiedzy, takich jak DBpedia.

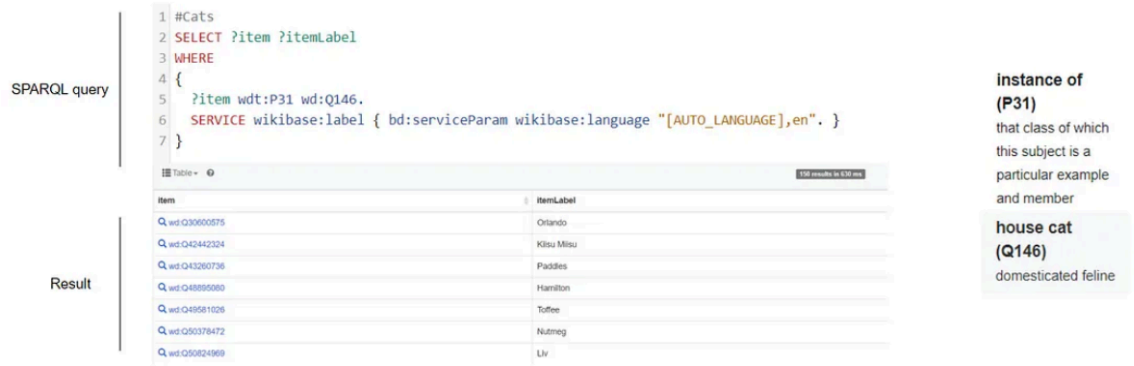
Po utworzeniu faktów jako RDF i umieszczeniu ich w bazie trójek RDF, takim jak Virtuoso, możemy je odpytać, aby uzyskać interesujące nas informacje. SPARQL to język zapytań RDF pozwalający na pobieranie i manipulowanie danymi przechowywanymi w formacie RDF.

Większość baz trójek RDF udostępnia stronę zapytań SPARQL do pobierania odpowiednich informacji.

W naszym przypadku możemy skorzystać z jednego z takich pomocników wizualnych do zapytań, udostępnionego przez Wikidane (pokazanego poniżej). Pokazane jest przykładowe zapytanie, w którym chcemy wydobyć wszystkie jednostki będące instancjami kota domowego.

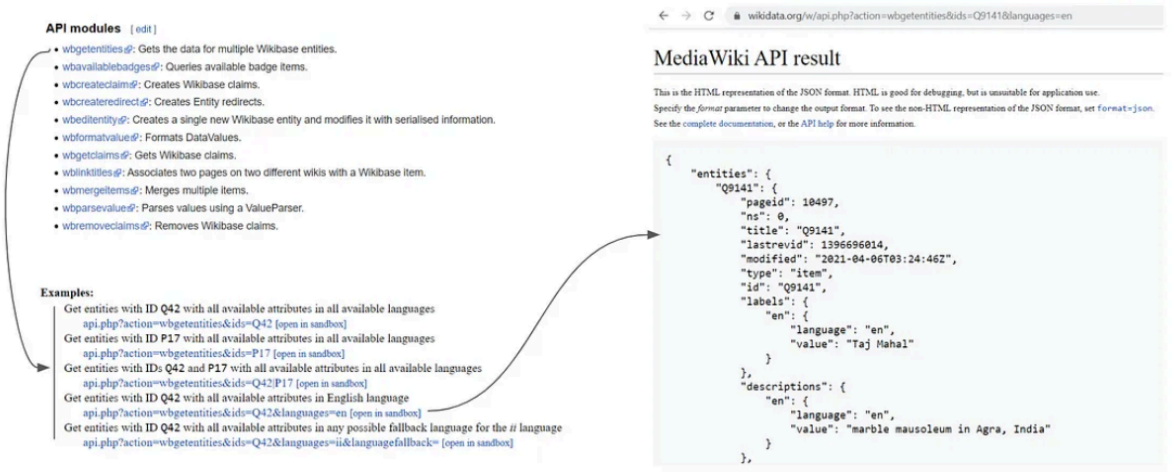
Jako że każda jednostka ma swój unikalny identyfikator (UID), relacja <instance of> jest przedstawiona jako P31, a jednostka <kot domowy> jako Q146. Zapytanie jest dość proste do zrozumienia, ponieważ najpierw staramy się przekazać, że chcemy otrzymać dowolną jednostkę będącą instancją kota domowego.

Ponieważ Wikidane zawierają dane w wielu językach, linia 6 jest potrzebna do filtrowania wyników specyficznych dla języka angielskiego. Wyniki (jednostki z ich UID i podstawowymi informacjami) są pokazane poniżej zapytania.



Rys. 12: Żądanie do grafu wiedzy przy użyciu języka SPARQL.

Otwartoźródłowe Grafy Wiedzy udostępniają również gotowe interfejsy API do często używanych zapytań. Jedno z takich API jest pokazane poniżej (dla Wikidanych), które zwraca istotne informacje dla określonej jednostki. Poniżej możemy zobaczyć wynik zapytania API wbgetentities dla jednostki Q9141, która jest UID dla Taj Mahal.



Rys. 13: Odpytywanie grafu wiedzy przy użyciu ogólnodostępnego API

Nowoczesne narzędzia do tworzenia GW

Graf to elastyczna struktura danych, a ontologia może ewoluować w miarę pojawiania się nowych danych. W rezultacie dane mogą być ciągle dodawane, aby pomóc obsługiwać nowe przypadki użycia. Do kodowania grafów wiedzy, jak już to zostało wspomniane wcześniej, najczęściej jest używany format RDF.

RDF (Framework Opisu Zasobów) to format danych, w którym każda jednostka wiedzy może być podzielona na strukturę (podmiot, predykat i obiekt) zwane trójkątem.

Dane sformatowane w RDF mogą być odpytywane przy użyciu języka zapytań, takiego jak SPARQL. Pozwala to na tworzenie aplikacji, które mogą przetwarzać grafy wiedzy, na przykład do wnioskowania na temat nowych danych lub uzyskiwania odpowiedzi na pytania, które możemy mieć dotyczące danych przechowywanych w grafie wiedzy.

Jednak RDF to tylko część tego, co potrzebujemy, aby zakodować graf. Potrzebujemy także implementacji, "konkretnej składni RDF", która pokazuje, jak **zapisać graf** sformatowany w RDF w formacie, który może być u**dostępniany między aplikacjami**. Jedną z takich składni, którą można użyć, jest **JSON-LD**. Oznacza ona JSON do Łączenia Danych.

Zacznijmy od przyjrzenia się prostemu przykładowi w formacie JSON:

```
[
	{
		"title":"Floating island",
		"author":"Unknown"
	},
	{
		"title":"Apple balloon",
		"author":"Grant Achatz"
	},
	{
```

```
            "title":"Opera",
            "author":"Cyriaque Gavillon"
      }
]
```

Jeśli nie rozpoznajemy nazw, możemy pomyśleć, że to być może tytuły książek lub filmów. Wyraźnie **brakuje kontekstu**, co prowadzi do **niejednoznaczności**. JSON-LD pozwala rozwiązać problem niejednoznaczności, umożliwiając **kodowanie danych zachowujące semantykę**. Osiąga się to poprzez dodanie obiektu @context, w którym każdy klucz jest skojarzony z identyfikatorem.

Jeśli rozszerzymy przykład JSON-a o poniższą zawartość, zarówno maszyna, jak i człowiek, mogą sprawdzić kontekst, aby dowiedzieć się, że JSON odnosi się do przepisów kulinarnych. Mówiąc inaczej, Chmurka, Balonik Jabłkowy i Opera.

```
[
      {
            "@context":"https://json-ld.org/contexts/recipe.jsonld",
            "title":"Floating island",
            "author":"Unknown"
      },
      {
            "@context":"https://json-ld.org/contexts/recipe.jsonld",
            "title":"Apple balloon",
            "author":"Grant Achatz"
      },
      {
            "@context":"https://json-ld.org/contexts/recipe.jsonld",
            "title":"Opera",
            "author":"Cyriaque Gavillon"
       }

  ]
```

Po zdefiniowaniu gramatyki (RDF) i konkretnej składni (JSON-LD) do kodowania grafów wiedzy, istnieje jeszcze jedna rzecz do dodania, aby osiągnąć interoperacyjność z innymi grafami wiedzy. Potrzebujemy **wspólnego słownika**, który określa, jakie właściwości powinna mieć jednostka - **ontologię**.

Jedną z takich ontologii, które są powszechnie używane, jest słownik **Schema.org**.

Jeśli chcemy wnioskować na temat wiedzy przechowywanej w grafie wiedzy, przydatne jest posiadanie języka, który może wyrażać właściwości i relacje dla jednostek w ontologii (słowniku), z którego składa się graf wiedzy. **OWL** (Web Ontology Language) to taki język. Zbudowany jest on na RDF (które można postrzegać jako gramatykę dla OWL) i różni się od JSON-LD, ponieważ JSON-LD opisuje jedynie składnię, którą używamy do kodowania naszego grafu wiedzy.

Przy użyciu OWL można *tworzyć aksjomaty* i *wyrażenia dla jednostek* w swojej ontologii.

```
SubClassOf( :Woman :Person )
```

Grafy wiedzy w rozwiązaniach około-telekomunikacyjnych

Grafy wiedzy również mogą być używane w ramach rozwiązań telekomunikacyjnych. W tym przypadku GW używane są do reprezentowania wymagań serwisowych wyrażonych w postaci intentów (wymagań w postaci tekstowe).

Autorzy artykułu [1] zaproponowali użycie grafów wiedzy do:

- **Implementacji ontologii i modelu opisu intencji** poprzez rozszerzenie znormalizowanego modelu zaproponowanego przez TMForum (zdefiniowanie modelu intencji, który jest neutralny dla domeny i zapewnia wystarczającą elastyczność dla różnych przypadków użycia aplikacji);
- **Rozszerzenie modeli ontologii i opisu usługi** do **reprezentacji usług krytycznych** dla misji i niekrytycznych dla misji przy użyciu standaryzowanych wskaźników wydajności i celów poziomu usługi (SLOs).

Aby zapewnić działanie takiego rozwiązania, został też zaproponowany oparty na grafie wiedzy framework IBN do przetwarzania i tłumaczenia intencji na dziedzinowo-sprecyzowane żądania usług oraz ich wdrożenia.

Autorzy wspomnianego artykułu założyli, że intenty można również przedstawić jako obiekty grafu wiedzy. Dane założenie narzuca w danym przypadku ograniczenia co do postaci używanego intentu. To oznacza, że oczekiwania konsumenta muszą być przekazywane w sposób znormalizowany. A zatem każda intencja musi zawierać informacje o wszystkich:

- **Wymaganiach funkcjonalnych**: obejmują rodzaj usługi do dostarczenia wraz z oczekiwaną funkcjonalnością dla użytkowników;

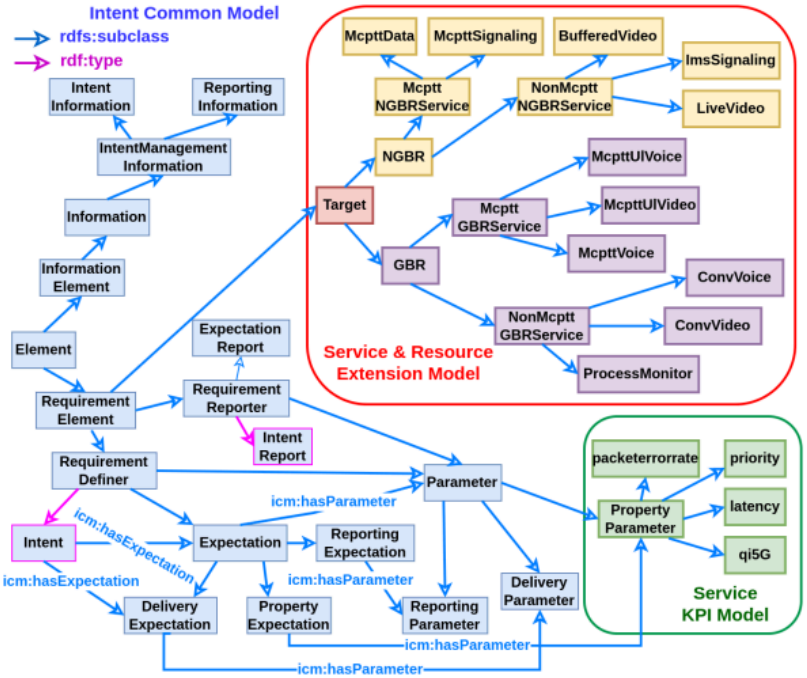
- **Wymaganiach nie-funkcjonalnych:** obejmują informacje potrzebne do wyjaśnienia oczekiwań dotyczących wymaganej usługi, np. kluczowe wskaźniki wydajności (KPI) i powiązane metryki.

Reprezentacja wiedzy o intencji musi podążać za strukturalnym podejściem, a grafy wiedzy stanowią doskonałe narzędzie do organizacji informacji (Rys. 14).

Zaproponowany model zapewnia kompatybilność między różnymi elementami przez uzupełnianie kilku klas informacyjnych, w tym *icm:Expectation, icm:Parameter, icm:Target* z modelu ICM. Klasy informacyjne wykorzystywane są do uwzględniania *danych specyficznych dla domeny* dotyczących usług i zasobów dla wdrożenia intencji w sieci. Klasy RDF w modelach rozszerzeń przedstawionych na Rys. 1 opisane są następująco:

- **Model Usługi** (Service) **i Zasobów** (Resource): Model zasobów jest rozszerzony za pomocą klasy icm:Target. Jest to zamierzony wynik niezbędny dla wdrożenia intencji (np. usługi). W przedstawionym frameworku jest on wykorzystywany do reprezentacji niezbędnego rodzaju zasobu (tj. wymagających zasobów z gwarantowaną przepływnością - *targetResource:GBR i* bez gwarancji przepływności *targetResource:NGBR*) dla określonej kategorii usług (np. service:McpttGBRService i service:NonMcpttNGBRService);

- **Model KPI Usługi**: Wykorzystujemy parametry KPI takie jak opóźnienie, packet error rate, priorytet i identyfikator QoS 5G (5GQI) jako parametry KPI usługi zawarte w podklasie icm:PropertyParameter modelu RDF ICM. Te parametry zawierają wartości jako termy literałów RDF dla różnych usług wymagających zasobów GBR i NGBR w podklasach kpi:latency, kpi:packeterrorrate, met:priority i met:qi5G, odpowiednio. Modele rozszerzenia KPI dla usług MC i NMC, przedstawione na Rys. 14 zostały stworzone na podstawie KPI z artykułu 3GPP [2].



Rys. 14: Wykorzystywana w artykule [1] struktura intentu w postaci GW, Model ICM

Informacje o wymaganiach następnie można pobierać przy pomocy żądań SPARQL.

Wydobywanie ontologii z języka naturalnego

Grafy wiedzy

Grafy wiedzy (KG) odgrywają kluczową rolę w wielu nowoczesnych aplikacjach. Jednak **konstrukcja KG na podstawie tekstu w języku naturalnym** stanowi **wyzwanie** ze względu na **złożoną strukturę tekstu**. Ostatnio zaproponowano wiele podejść do przekształcania tekstu w języku naturalnym na trójki w celu uzyskania KG. Takie podejścia jeszcze nie dostarczyły efektywnych rezultatów dla mapowania wyodrębnionych elementów trójek, zwłaszcza predykatu, na ich równoważne elementy w KG. **Mapowanie predykatów jest istotne**, ponieważ może zmniejszyć heterogeniczność danych i zwiększyć możliwość wyszukiwania w KG.

Autorzy artykułu [3] zaproponowali *T2KG*, **automatyczny framework do tworzenia KG dla tekstu w języku naturalnym**, aby bardziej efektywnie mapować tekst na predykaty. W niniejszym frameworku przedstawiono hybrydowe połączenie podejścia opartego na

regułach i podejścia opartego na podobieństwie do mapowania predykatu na odpowiadający mu predykat w KG.

Tworzenie GW głównie zawiera trzy kroki: ekstrakcję wiedzy, mapowanie jednostek i integrację danych. Natomiast rozwiązanie, jak to zostało pokazane na Rys. 15, T2GK proponuje 5 kroków (3 z nich dokonywane są równolegle): mapowanie jednostek, rozwiązywanie koreferencji, ekstrakcja trójek, integracja trójek, mapowanie predykatów.

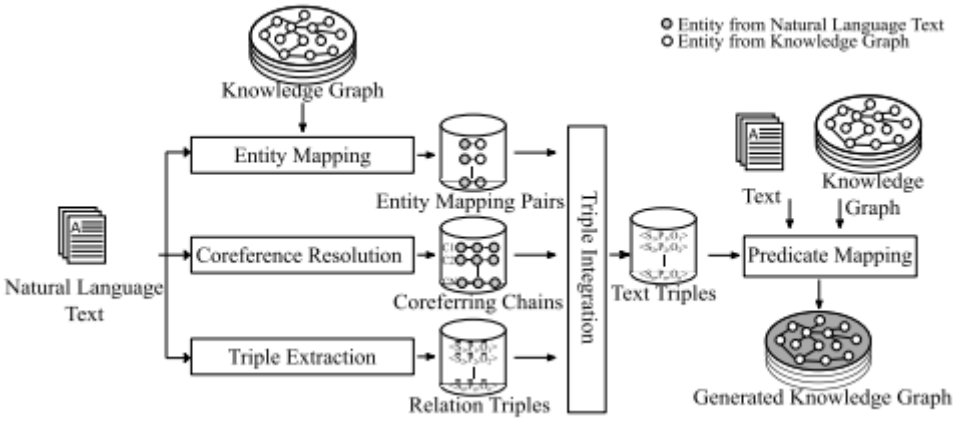


Fig. 15: Architektura framework'u T2GK

Przy **mapowaniu jednostek** komponent ma na celu odwzorowanie jednostki w tekście naturalnym na unikalny identyfikator zasobu (URI). Jeśli jednostka w tekście może być odwzorowana na identyczną jednostkę w grafie wiedzy (KG), używany jest URI z KG jako reprezentant wyodrębnionej jednostki. W przeciwnym razie przypisywane jest nowe URI.

Podczas **rozwiązywania koreferencji** odpowiedni komponent ma na celu wykrycie łańcuchów koreferencyjnych jednostek w tekście i ich grupowanie. Ten krok pozwala na identyfikację działań dla identycznych jednostek używających różnych wyrażeń.

Komponent odpowiadający za **ekstrakcję trójek** odnajduje jednostki w tekście naturalnym. W tym przypadku też definiujemy relację trójki, czyli predykat trójki oraz związane z nią argumenty: podmiot i obiekt.

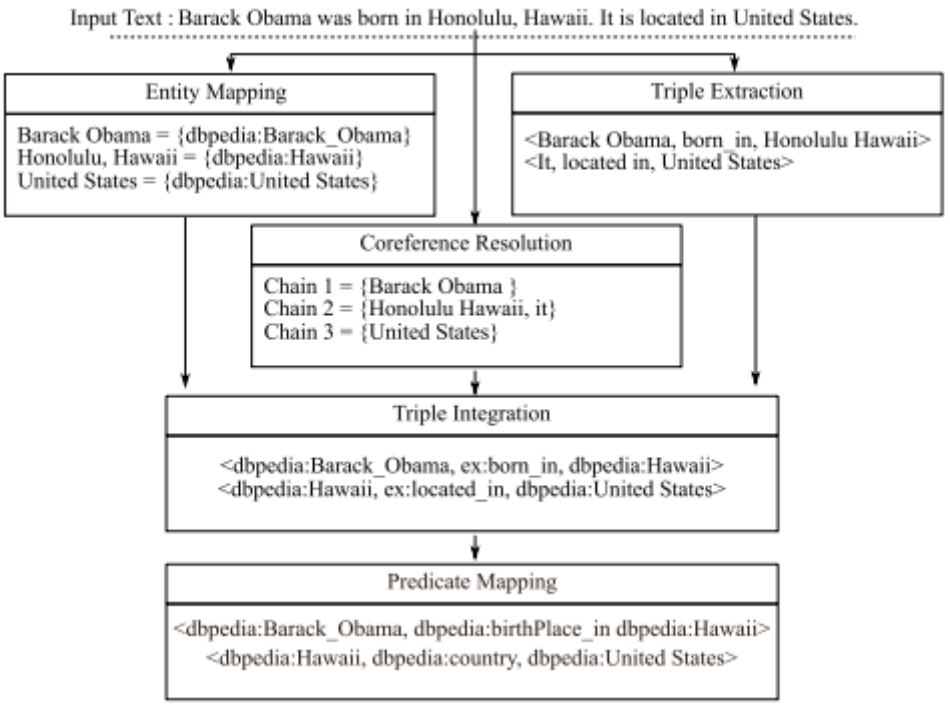


Fig. 16: Przykładowy przepływ danych w framework'u T2GK

Integracja trójki polega również na generowaniu tekstowych trójek z wyjść z komponentów mapowania jednostek i rozwiązywaniu koreferencji. Konieczna w tym procesie jest transformacja relacji trójki, aby dostosować ją do standardów KG.

W przypadku **mapowania predykatów**, próbujemy odwzorować predykat tekstowej trójki na identyczny predykat w KG. Tutaj jest wykorzystywane podejście hybrydowe: oparte na regułach oraz oparte na podobieństwie, używając metryki podobieństwa wektorowego.

Tak skonstruowane grafy wiedzy mogą być używane do następującego połączenia jednostek (głównie słów kluczowych) wyrażonych w ramach pewnego przekazywanego intent'u z ich odpowiednikami na innych poziomach abstrakcji (przykładowo, połączenie słów kluczowych na poziomie biznesowym z ich odpowiednikami na poziomie serwisów).

Niemniej jednak jest ważne, żeby pamiętać że takiego rodzaju podejścia (rozbudowane, składające się z wielu niezależnych jednostek) cechują się dość niskim poziomem poprawności otrzymywanych wyników (głównie na poziomie 50-60%).

Modele językowe

Zanim użyjemy grafów wiedzy, zawierających ontologię implementowanych serwisów, a przynajmniej ustrukturyzowane wymagania funkcjonalne i niefunkcjonalne, intent wyrażony w postaci języka naturalnego powinien zostać przetłumaczony na postać bardziej zwartą i ustrukturyzowaną.

W tym celu mogą zostać użyte różnego rodzaju modele językowe. Z jednej strony można próbować tworzyć takiego rodzaju modele, jednak to wymaga dogłębnego zrozumienia działania takich modeli: wpływu różnych typów warstw transformatorów na wynik

działania takich modeli, sposoby doboru tysięcy parametrów takich modeli i duże zasoby obliczeniowe do wytrenowania modeli, aby je można było potem używać.

Kolejnym wyjściem jest **użycie przetrenowanych modeli LLM** i następne ich **dostrajanie** na podstawie posiadanych wąsko-dziedzinowych danych albo użycie modeli **generatywnych** typu GPT-3 i tworzenie tzw. promptów, pasujących do predefiniowanych zadań.

W pierwszym przypadku możemy użyć modeli BERT, udostępnionych przez Hugging Face. Między innymi, dana platforma oferuje takie narzędzia jak:

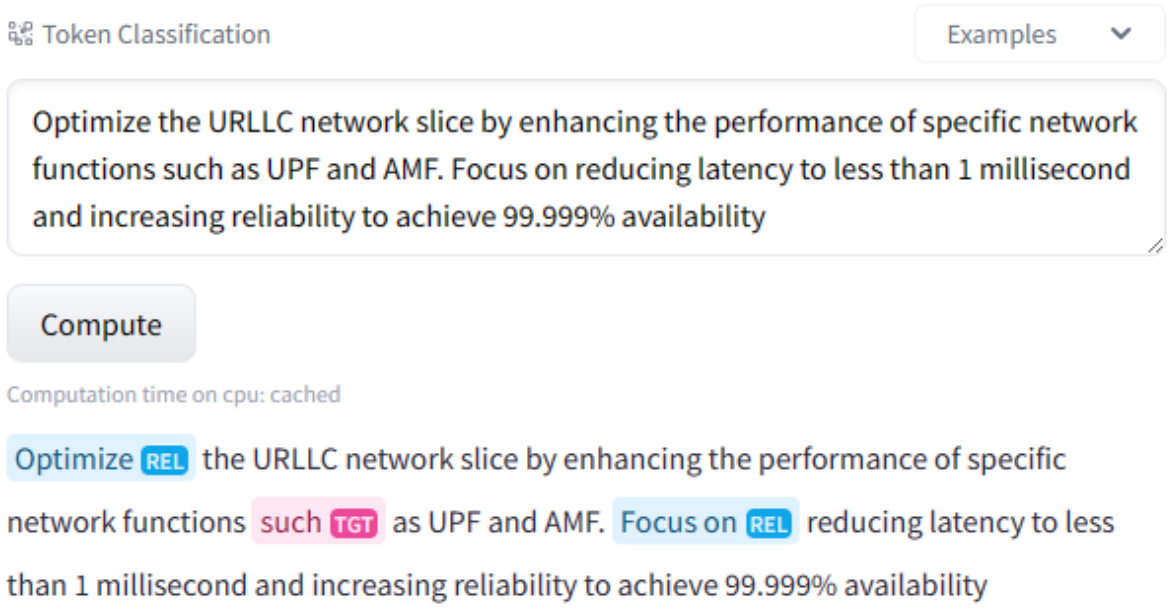
- **Transformers Library**: Hugging Face jest znane z biblioteki o nazwie "Transformers", która zawiera implementacje i pre-trenowane modele różnych architektur, takich jak **BERT**, **GPT**, **RoBERTa** itp. Ta biblioteka umożliwia łatwe korzystanie z tych zaawansowanych modeli w projektach NLP.
- **Model Hub**: Hugging Face oferuje model hub, gdzie społeczność może dzielić się, **pobierać i dostosowywać pre-trenowane modele**. Umożliwia to łatwy dostęp do zaawansowanych modeli do zastosowań w dziedzinie przetwarzania języka naturalnego.
- **Tokenizers**: Hugging Face dostarcza również narzędzia do **tokenizacji tekstu**, co jest istotnym elementem w przetwarzaniu języka naturalnego.
- **Trening i Fine-Tuning**: Platforma umożliwia również **trening i dostosowywanie modeli** do konkretnych zastosowań.
- **Hugging Face AI Research** (Hugging Face Badania nad SI): Firma prowadzi również badania nad sztuczną inteligencją i publikuje różne artykuły naukowe w tej dziedzinie.

Przykładowym rozwiązaniem, przydatnym w ramach mojej pracy magisterskiej może być przetrenowany model BERT (vishnun/knowledge-graph-nlp), który oferuje funkcjonalność ekstrakcji jednostek i znalezienia relacji między nimi.

Przykładowo, dla intentu o brzmieniu:

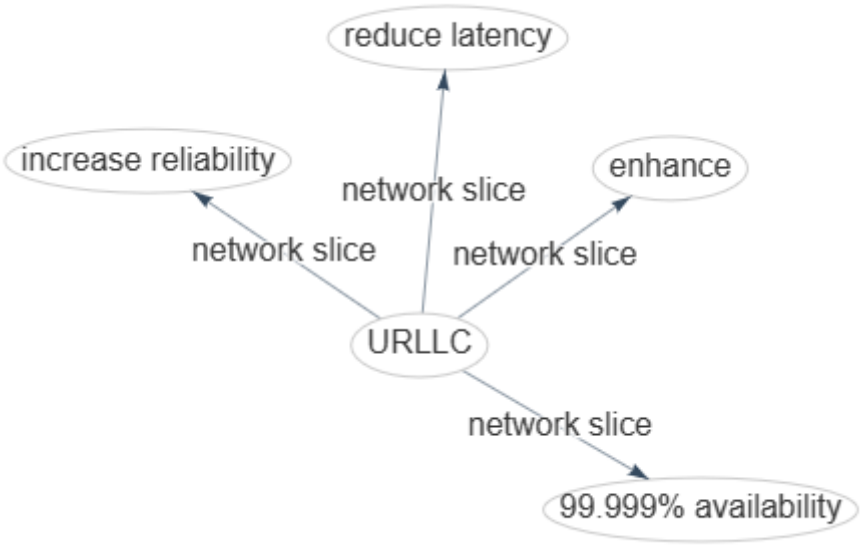
> “Optimize the URLLC network slice by enhancing the performance of specific network functions such as UPF and AMF. Focus on reducing latency to less than 1 millisecond and increasing reliability to achieve 99.999% availability”

Encje i relacje rozpoznane przez taki model zostały przedstawione na Rys. 17. Oczywiście, taki model nie posiada informacji o technicznej terminologii używane w telekomunikacji, niemniej jednak może on zostać wytrenowany w taki sposób, aby je rozpoznawać.



Rys. 17: Działanie przetrenowanego modelu BERT

Inny narzędzie o nazwie GraphGPT, udostępnione jako rozwiązanie oparte na GPT (które wykorzystuje inżynierię zapytań), również umożliwia ekstrakcję jednostek i relacji między nimi (i ilustruje utworzony graf). Dla identycznego intentu, zdefiniowanego wyżej, wynik pokazany na Rys. 18 został stworzony.



Rys. 18: Działanie GraphGPT opartego na modelu Davinci

Zebrane “intenty”

1. **Business Intent**: Provide Enhanced Mobile Broadband (eMBB) Services for High-Speed Internet Access
    a. **Service Intent**:
        i. Network Functions: gNB (gNodeB), AMF (Access and Mobility Management Function), UPF (User Plane Function)
        ii. Configurations: Optimize gNB for high data rates, configure AMF for mobility management, ensure UPF supports high-speed data transfer.
2. **Business Intent**: Enable Massive Internet of Things (IoT) Connectivity
    a. **Service Intent**:
        i. Network Functions: NRF (Nodal Function), SMF (Session Management Function), UPF
        ii. Configurations: Optimize NRF for device discovery, configure SMF for session management, ensure UPF supports massive device connections.
3. **Business Intent**: Ensure Ultra-Reliable Low Latency Communications (URLLC) for Mission-Critical Applications
    a. **Service Intent**:
        i. Network Functions: AMF, SMF, UPF
        ii. Configurations: Optimize AMF for quick access setup, configure SMF for low-latency sessions, ensure UPF provides reliable and low-latency data transfer.
4. **Business Intent**: Provide Network Slicing for Customized Services
    a. **Service Intent**:
        i. Network Functions: NSSF (Network Slice Selection Function), SMF, UPF
        ii. Configurations: Configure NSSF for dynamic slice selection, optimize SMF for diverse service sessions, ensure UPF supports customized data handling.
5. **Business Intent**: Ensure Network Security and Privacy
    a. **Service Intent**:
        i. Network Functions: AUSF (Authentication Server Function), NEF (Network Exposure Function), SMF
        ii. Configurations: Strengthen AUSF for secure authentication, configure NEF for controlled data exposure, optimize SMF for secure session management.
6. **Business Intent**: Optimize Network Resource Utilization for Cost Efficiency
    a. **Service Intent**:
        i. Network Functions: NRF, UPF, AUSF
        ii. Configurations: Optimize NRF for resource discovery, ensure UPF utilizes resources efficiently, configure AUSF for efficient authentication.
7. **Business Intent**: Implement Network Slicing for Diverse Industries (e.g., Manufacturing, Healthcare)
    a. **Service Intent**:
        i. Network Functions: NSSF, AMF, SMF

        ii. Configurations: Configure NSSF for industry-specific slices, optimize AMF and SMF for varied service requirements.
8. **Business Intent**: Offer Low-Cost IoT Connectivity for Agriculture
    a. **Service Intent**:
        i. Network Functions: NRF, SMF, UPF
        ii. Configurations: Optimize NRF for agriculture-specific devices, configure SMF for low-cost sessions, ensure UPF supports affordable connectivity.
9. **Business Intent**: Provide Seamless Handovers for Mobile Users
    a. Service Intent:
        i. Network Functions: AMF, UPF, SMF
        ii. Configurations: Optimize AMF for quick handover decisions, configure UPF for seamless user mobility, ensure SMF manages smooth session transitions.
10. **Business Intent**: Enhance Network Monitoring and Analytics
    a. Service Intent:
        i. Network Functions: NEF, AMF, NRF
        ii. Configurations: Configure NEF for real-time analytics, optimize AMF for network monitoring, ensure NRF provides valuable data insights.

Podsumowanie

W tym semestrze udało mi się przeanalizować literaturę i wyrafinować dalszy plan badań w kierunku tłumaczenia intentu (wyrażonego w postaci języka naturalnego, na poziomie biznesowym) na postać bardziej ustrukturyzowaną.

Zapoznałam się z istotą i metodami tworzenia grafów wiedzy w celu następnego przechowywania w nich wiedzy dziedzinowej i/albo ontologię wykorzystywanych intentów (prawdopodobnie na poziomie serwisowym).

Udało mi się zapoznać z metodami tworzenia LLM (LArge LAnguage Models), które mogą być wykorzystywane do analizy intentu w postaci języka naturalnego, wyodrębnieniu informacji o wymaganiach i jej ukształtowaniu do następnego użycia w ramach zapytań SPARQL do odpytywania grafów wiedzy w celu uzyskania “przetłumaczenia” intentu z jednego poziomu abstrakcji do kolejnego.

W tym celu chciałabym skorzystać z grafów wiedzy, a właściwie framework’u RDF w połączeniu z modelami sztucznej inteligencji do:

1. Stworzenia grafu wiedzy na podstawie wiedzy dziedzinowej (między innymi, szczegółowej informacji o blokach 5G Core);

2. Utworzenia i następnego wytrenowania modelu spanBERT do odnajdywania słów/wyrażeń kluczowych w ramach intento biznesowych, które definiują wdrażane usługi. Następnie takie słowa kluczowe zostaną odpowiednio podzielone według ich znaczenia/roli i na ich podstawie zostanie sporządzone zapytanie SPARQL;

3. Używając wygenerowanych zapytań SPARQL następnie odpytany graf wiedzy, który w odpowiedzi wyśle informację o zestawie funkcji sieciowych, które pozwalałyby na wdrożenie intentu biznesowego;

Plan na kolejny semestr

Cel: planowanie (projektowanie) struktury serwisu (usługi 5G core na podstawie intentu biznesowego;

Plan (projekt) struktury serwisu: minimalny zestaw funkcji 5G core spełniających intent

1. Szczegółowa analiza funkcjonalna wszystkich bloków 5G core;
2. Propozycja metodyki tworzenia grafu wiedzy dla takiej sieci/usług; podstawą jest nasza wiedza domenowa (domain knowledge) dotycząca sieci 5G;
3. Propozycja grafu wiedzy i ilustracja jego wykorzystania na wybranych przykładach intentów;

Bibliografia

[1] Mehmood, Kashif, Katina Kralevska, and David Palma “Knowledge-based Intent Modeling for Next Generation Cellular Networks” arXiv:2302.08544v2, July. 2023, https://arxiv.org/pdf/2302.08544.pdf.

[2] 3GPP, “5g; system architecture for the 5g system (5gs),” Technical Specification 23.501, 3GPP, 2022.

[3] Natthawut Kertkeidakachorn, and Ryutaro Ichise, “An Automatic Knowledge Graph Creation Framework from Natural Language Text” - Special Section on Semantic Web and Linked Data,, IEICE trans. Inf. & Syst., vol.E101–D, no.1 Jan 2018