# Projekt (40pkt) - NLP, semestr zimowy 2019/2020

**Termin oddawania projektu: 10 lutego 2020r.**

Link do zbioru danych i embeddingów znajduje się tutaj.

Zaimplementuj sieć neuronową, która rozwiązuje zadanie Natural Language Inference na podzbiorze Stanford Natural Language Inference (SNLI). Zadanie polega na klasyfikacji pary zdań (przesłanka, hipoteza) do jednej z 3 kategorii („entailment" - z pierwszego zdania wynika drugie, „contradiction" - drugie zdanie przeczy pierwszemu, „neutral" - zdania są niezależne).

Zadania:

- (8pkt) Preprocessing (Zarówno podzbiór SNLI jak i embeddingi znajdują się na google drive - link powyżej):
  - Ze zbiorów SNLI wybierz tylko te zdania, w których większość adnotatorów wybrała jedną klasę (czyli gold_label ≠ „-"). Wypisz 10 zdań wyrzuconych.
  - Stokenizuj zdania i zamień wszystkie litery na małe (ponieważ embeddingi słów zakładają, że to zrobiliśmy). Wypisz liczbę różnych słów, które zostały.
  - Jako enkodera słów użyj embeddingów wytrenowanych metodą GloVe na zbiorze Wikipedia 2014 (6B tokenów, 400k vocabulary, 200D) (ściągamy te embeddingi, uczenie ich zajęłoby zbyt dużo czasu). Do macierzy embeddingów wybierz tylko te słowa, które występują jednocześnie w GloVe i SNLI.
  - Słowa, które występują w SNLI, ale nie znajdują się w GloVe, traktuj jako <unk>, dla którego wektor zainicjalizuj losowo. Podobnie stwórz dwa dodatkowe wektory, <bos>, <eos>, które reprezentują początek i koniec zdania (zmodyfikuj zbiór SNLI w taki sposób, żeby każde zdanie zaczynało się od tokenu <bos>, kończyło tokenem <eos>, a dla nieznanych słów zawierało token <unk>). Wypisz liczbę wszystkich słów oraz słów zamienionych na <unk>.
- (14pkt) Zdefiniuj model i wytrenuj go (monitorując co epokę accuracy na zbiorze treningowym oraz walidacyjnym), przykładowa architektura:
  - Słowa powinny być enkodowane za pomocą enkodera słów (używamy word embeddingów jak powyżej, nie uczymy)
  - Zdania powinny być zaenkodowane za pomocą enkodera zdań (LSTM, który jest uczony podczas treningu - można użyć jednego LSTM, aby zaenkodować przesłankę i hipotezę, osobno) - używamy hidden size równy 200.
  - W wyniku powyższego enkodowania dostaniemy 2 wektory, które są reprezentacjami zdań (dla przesłanki, dla hipotezy). Te wektory konkatenujemy i przepuszczamy przez dwie warstwy sieci neuronowej (pierwsza z hidden size 200 i funkcją aktywacji relu, następnie warstwa softmaxowa)
  - Używamy batchów (przykładowo batch size 32)
  - Proszę dodać wykres zmian lossów i accuracy per epoch oraz wypisać po 3 przykłady dobrze i źle zaklasyfikowane (1 per klasa)
- (8pkt) Reprodukowalność:
  - Napisz funkcję, która zapisuje wagi najlepszego modelu po każdej epoce treningu (i jej używaj).
  - Napisz funkcję, która ładuje model z zapisanych wag. Na koniec treningu wybierz model z najlepszym wynikiem na zbiorze walidacyjnym i zapisz go (będzie potrzebny przy oddawaniu projektu).
- (10pkt) Wynik na zbiorze testowym:
  - >68% - 10pkt, >65.5% - 8pkt, >63% - 6pkt, >60.5% - 4pkt, >58% - 2pkt

W pierwszej linijce rozwiązania proszę o oświadczenie, że zadanie zostało wykonane samodzielnie.

Wskazówki:

- Nieoczywistym problemem jest używanie batchów w sieciach rekurencyjnych. Problem polega na tym, że zdania są często różnej długości, a torchowy tensor potrzebuje, aby wymiar długości zdania był ustalony. Jednym z rozwiązań jest użycie paddingu, czyli dodawania tokenów <pad> na końcu zdań krótszych po to, aby ostatecznie wszystkie zdania miały długość równą długości najdłuższego zdania w batchu. Pod tym linkiem można znaleźć funkcje używane do batchowania w RNNach. Prawdopodobnie najłatwiej jest użyć pad_sequence. Warto pamiętać, że przy tej funkcji defaultowo zostaje zakładane, że token <pad> ma w naszym słowniku (word2idx) index 0.
- Czasami zostaje nam przydzielona karta, która jest w dużej części zajęta, wtedy może nam zabraknąć pamięci. Gdy tak się stanie, należy zrestartować notebooka. Poniższy kod może pomóc w sprawdzeniu ilości pamięci na GPU.

```
! ln -sf /opt/bin/nvidia-smi /usr/bin/nvidia-smi
! pip install gputil
! pip install psutil
! pip install humanize

import psutil
import humanize
import os
import GPUtil as GPU
GPUs = GPU. getGPUs ( )
# XXX: o n l y one GPU on Colab and isn't guaranteed
gpu = GPUs [ 0 ]

def printm ( ) :
    process = psutil.Process(os.getpid( ))
    print ("GenRAM Free: " + humanize.naturalsize(psutil.virtual_memory(). available),
    print ('GPU RAM Free : { 0 : . 0f}MB | Used : { 1 : . 0f }MB | Util { 2 : 3 . 0 f }% | Total{ 3 : . 0 f }MB'

printm ( )
```

- W razie problemów z optymalizacją lub chęcią osiągnięcia lepszego accuracy można eksperymentować z optimizerem (np. Adam, RMSProp), dropoutem, batch normalization, lr schedulerem.

Powodzenia!