# UCL Machine Reading — FNC-1 Submission

Benjamin Riedel
benjamin.riedel.09@ucl.ac.uk

Isabelle Augenstein
i.augenstein@ucl.ac.uk

George Spithourakis
g.spithourakis@cs.ucl.ac.uk

Sebastian Riedel
s.riedel@ucl.ac.uk

June 2017

## Summary

The stance detection model submitted for stage number 1 of the Fake News Challenge (FNC-1) is a single, end-to-end system consisting of lexical as well as similarity features fed through a multi-layer perceptron (MLP) with one hidden layer.

Although relatively simple in nature, the model appears to perform on par with more elaborate, ensemble-based systems submitted.

## Features

The features extracted consist of three overarching elements only:

- A bag-of-words term frequency (BoW-TF) vector of the headline
- A BoW-TF vector of the body
- The cosine similarity of term frequency-inverse document frequency (TF-IDF) vectors of the headline and body

Tokenization is performed by the standard `scikit-learn` tokenizer as part of applying `CountVectorizer`, `TfidfTransformer` and/or `TfidfVectorizer`.

The bag-of-words (BoW) used to calculate the term frequency vectors is based on the vocabulary of the training set, excluding a specified set of stop words and limited to a total of 5,000 most frequent words. The set of stop words consists of a subset of the standard `scikit-learn` stop word list for the English language.

For the TF-IDF vectors of the headline and body in the cosine similarity calculations, the BoW is based on the vocabulary of both the train and the test set. All other parameters used in the corresponding `scikit-learn` vectorizer are identical to those described for the term frequency vectors.

These three overarching elements are concatenated in a feature vector of total size 10,001 which is fed into the classifier.

## Classifier

The classifier is a MLP with one hidden layer of 100 units with rectified linear unit (ReLU) activation.

Prediction is based on the argmax of the softmax on the output of the final layer.

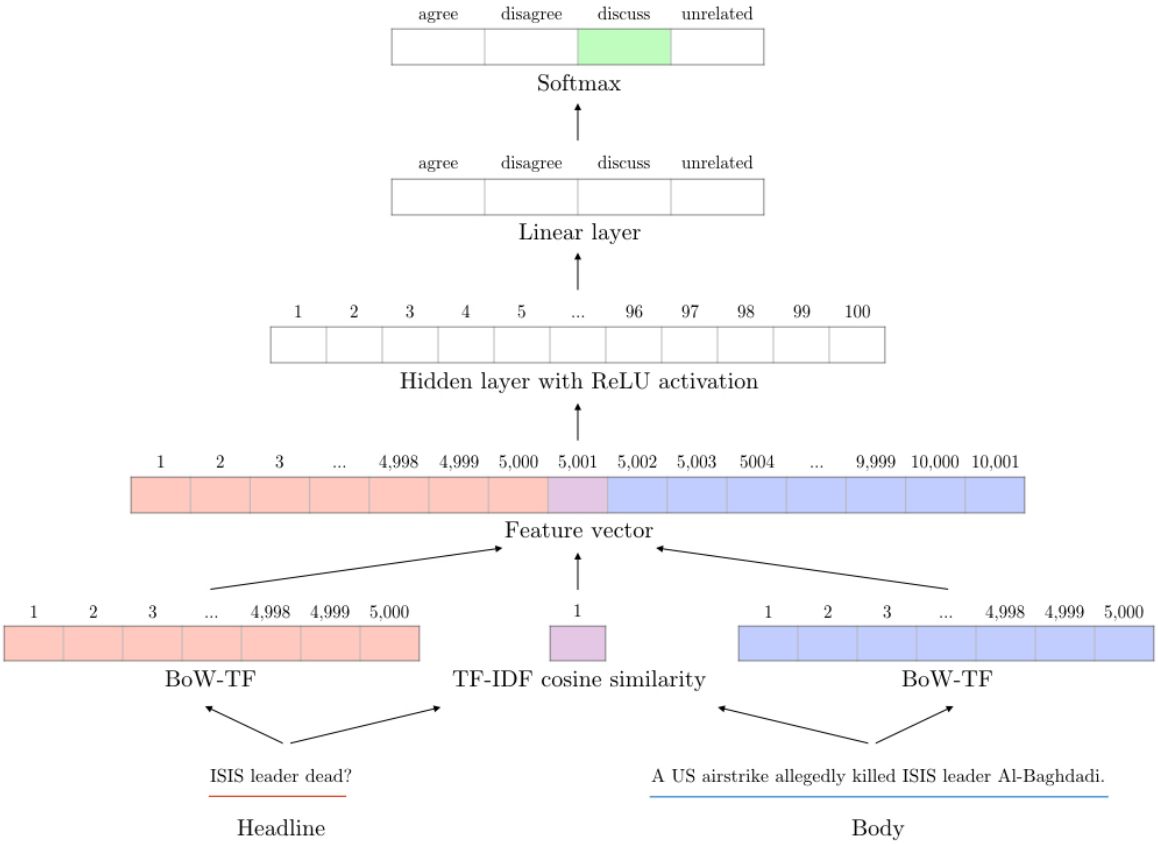


Figure: Schematic diagram of model

## Training

The loss minimized during training was the sum of the L2 loss calculated on the MLP weights and the sparse softmax cross entropy between the logits and the labels. The level of alpha used in the L2 loss calculations was set to 0.0001.

Optimization was carried out using Adam with a learning rate of 0.01 and gradient clipping by a global norm clip ratio of 5.

Moreover, the setup was regularized using a dropout of 0.4 on the output of both perceptron layers.

Batch training of size 500 was thus performed on the entire training set and stopped early after 90 epochs.

The criterion used for early-stopping was a qualitative combination of the plateau of the loss on the training set and mean performance of the model on 50 random splits of the data into training and hold-out sets as defined in the official baseline setup.

## Hyperparameters

The hyperparameters of the model were optimized during development using random search on a grid of combinations and (cross-validation on) various splits of the data.

The full set of hyperparameter labels, their description, the values considered and corresponding optimized values are provided below.

- `lim_unigram`: number of most frequent words considered in `CountVectorizer` and `TfidfVectorizer`
  - values: 1,000; 2,000; 2,500; 4,000; 5,000; 7,500; 10,000
  - optimized: 5,000
- `hidden_size`: number of hidden units in hidden layer of MLP
  - values: 50; 75; 100; 200; 300; 400; 500; 600
  - optimized: 100
- `train_keep_prob`: 1 - dropout on output of both layers of MLP
  - values: 0.5; 0.6; 0.7; 0.8; 0.9; 1.0
  - optimized: 0.6
- `l2_alpha`: alpha level used in L2 loss calculations
  - values: 0.1; 0.01; 0.001; 0.0001; 0.00001; 0.000001; 0.0000001
  - optimized: 0.0001
- `learn_rate`: learning rate of Adam
  - values: 0.1; 0.01; 0.001
  - optimized: 0.01

- `clip_ratio`: global norm clip ratio
  - values: 1; 2; 4; 5; 6; 8; 10
  - optimized: 5
- `batch_size`: size of batch
  - values: 250; 500; 750; 1,000
  - optimized: 500
- `epochs`: number of epochs
  - max: 1,000
  - optimized: 90

# Reproducibility

Rather than providing seed values and requiring the model to be retrained, the public `GitHub` repository provided contains relevant scripts and the `TensorFlow` model trained as part of the submission.

The submission can easily be reproduced by loading this model using the `pred.py` script to make the predictions on the relevant test set.

Alternatively, as suggested by the organizers of the competition, the validity of the submission can be checked by also using the `pred.py` script to train the model with different seeds and evaluating the mean performance of the system.

# Getting started

To get started, simply download the files in the public `GitHub` repository provided to a local directory.

## Prerequisites

The model was developed, trained and tested using the following:

- `Python==3.5.2`
- `NumPy==1.11.3`
- `scikit-learn==0.18.1`
- `TensorFlow==0.12.1`

Please note that compatibility of the saved model with newer versions of `TensorFlow` has not been checked. Accordingly, please use the `TensorFlow` version listed above.

### Installing

Other than ensuring the dependencies are in place, no separate installation is required.

Simply execute the `pred.py` file once the `GitHub` repository has been saved locally.

## Reproducing the submission

The `pred.py` script can be run in two different modes: 'load' or 'train'. Upon running the `pred.py` file, the user is requested to input the desired mode.

Execution of the `pred.py` file in 'load' mode entails the following:

- The train set will be loaded from `train_stances.csv` and `train_bodies.csv` using the corresponding `FNCData` class defined in `util.py`.
- The test set will be loaded from `test_stances_unlabeled.csv` and `train_bodies.csv` using the same `FNCData` class. Please note that `test_stances_unlabeled.csv` corresponds to the second, amended release of the file.
- The train and test sets are then respectively processed by the `pipeline_train` and `pipeline_test` functions defined in `util.py`.
- The `TensorFlow` model saved in the `model` directory is then loaded in place of the model definition in `pred.py`. The associated `load_model` function can be found in `util.py`.
- The model is then used to predict the labels on the processed test set.
- The predictions are then saved in a `predictions_test.csv` file in the top level of the local directory. The corresponding `save_predictions` function is defined in `util.py`. The predictions made are equivalent to those submitted during the competition.

Execution of the `pred.py` file in 'train' mode encompasses steps identical to those outlined above with the exception of the model being trained as opposed to loaded from file. In this case, the predictions will obviously not be identical to those submitted during the competition.

The file name for the predictions can be changed in section '`Set file names`' at the top of `pred.py`, if required.

Please note that the predictions are saved in chronological order with respect to the `test_stances_unlabeled.csv` file, however, only the predictions are saved and not combined with the `Headline` and `Body ID` fields of the source file.