Hedge Cue Detection in Natural Language Text

Alessio Sortino
alessio.sortino@stud.unifi.it

Vishal Sharma
vishal.sharma@stud.unifi.it

Machine Learning

University of Florence, Department of Information Engineering

February 21, 2017

Overview

1. Introduction
2. GloVe
3. LSTM
4. Wikipedia Dataset
5. The Model
6. Experimental Settings
7. Results
8. Conclusions

Introduction

- Hedge cue detection is a Natural Language Processing (NLP) task that consists of determining whether sentences contain hedges.
- These linguistic devices indicate that authors do not or cannot back up their opinions or statements with facts.
- This binary classification problem, i.e. distinguishing factual versus uncertain sentences, only recently received attention in the NLP community.

Introduction

This project aims to address two task:

1. Learning to detect sentences containing uncertainty
2. Learning to resolve the in-sentence scope of hedge cues

The model use embedding word vectors, generated by GloVe, to better describe the semantic meaning of words, and LSTM to capture the sequential nature of a sentence.

The performances achieved by our method are a bit far from today's state-of-the-art, obtaining 60% of F-score on the best binary classifier model, and 32% on the best model for in-sentence cue detection.

Related Work

Different techniques achieve state-of-the-art result on both of our tasks, in particular for the dataset we used:

- In 2010, Georgescul et al. achieved 60.2% of F-score using a Support Vector Machine classifier and employing n-grams from annotated cue phrases as features. This method achieved the top performance in the CoNLL 2010 Wikipedia hedge-detection task.

- In 2012, Frasconi et al. achieved 61.48% of F-score on classification using a kLog, a kernel-based logical and relational learning.

- In 2014, Li et al. achieved 62.9% of F-score on classification using a Hidden Markov Model (HMM).

- In 2010, Morante et al. (CoNLL 2010 Wikipedia) achieved 57.3% of F-score on in-sentence cue detection using a memory-based system that relies on information from syntactic dependencies.

Word Representation

- Vector Space Models (VSMs) represent words in a continuous space where semantically similar words are mapped to nearby points.

- For example, “monitor” is embedded ideally nearby “screen”.

- VSMs have a long history in Natural Language Processing, but all methods depend in some way how words share semantic meaning if appears in the same contexts. The different approaches that leverage this principle can be divided into two categories:
  1. *Count-Based Methods*, e.g. LSA and HAL.
  2. *Prediction-Based Methods*, e.g. word2vec (CBOW and Skip-Gram models) and ivLBL.

Count-Based Methods

The Count-Based Methods for creating word embeddings use low-rank approximations to decompose large matrices made of statistical information about all the corpus.

- In LSA (Latent Semantic Analysis), rows correspond to words and the columns correspond to different documents. The entries correspond to the number of times a given word occurs in a document.
- In HAL (Hyperspace Analogue to Language), rows and columns both correspond to words and the entries correspond to the number of times a given word occurs in the context of another given word (a *co-occurrence matrix*).

Problem: frequent words contribute a disproportionate amount to the similarity measure and so some kind of normalization is needed.

Prediction-Based Methods

Prediction-Based Methods learn word representations from local context windows instead than all the corpus. The typical model used is *word2vec* of Mikolov et al. (2013).

- It's a full neural network with a simple single-layer architecture based on the inner product between two word vectors.
- The target of this model is to maximize the log probability of the next word w_i given the context words w_k in term of a softmax function as a windows scan over the corpus.

The model is divided in two sub-models:

1. Continuous Bag of Words (CBOW): the target is to predict a word given its context. Good for *small* datasets.
2. Continuous Skip-Gram: the target is to predict a word's context given the word itself. Good for *large* datasets.

Problem: they operate on all the corpus locally.

Global Vectors for Word Representation (GloVe)

- The main idea of GloVe is to use the benefits of both methods by counting data while simultaneously capturing the meaningful linear substructures prevalent in prediction-based methods.
- GloVe is a global log-bilinear regression model for the unsupervised learning of word representations.
- It's been developed by Pennigton et al. (2015) and outperforms other models on word analogy, word similarity, and named entity recognition tasks.

The GloVe Model

- First of all, word co-occurrence statistics are collected in a word-word co-occurrence matrix X. Each element X_{ij} of such matrix represents how many times word w_i appears in context of word w_j.
- The probability that the word w_j appears near word w_i is:

$$P_{ij} = P(w_j|w_i) = \frac{X_{ij}}{\sum_k X_{ik}}$$

However, the relationship between two words can be examined by studying the ratio of the probabilities compared to another word, e.g.

$$\frac{P(solid|ice)}{P(solid|steam)} > \frac{P(gas|ice)}{P(gas|steam)}$$

- The model can be written as:

$$F(w_i, w_j, \widetilde{w}_k) = \frac{P_{ik}}{P_{jk}} \Longrightarrow w_i^T \widetilde{w}_k + b_k + \widetilde{b}_k = \log(X_{ik}),$$

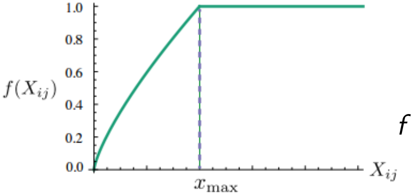
where the right member is the solution of the left equation.

The GloVe Model

- The main drawback of this model is that it weighs all co-occurrences equally, even those that happen rarely or never.
- GloVe uses a new weighted least squares regression model:

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij}) \left(w_i^T \widetilde{w}_j + b_i + \widetilde{b}_j - \log X_{ij} \right)^2$$

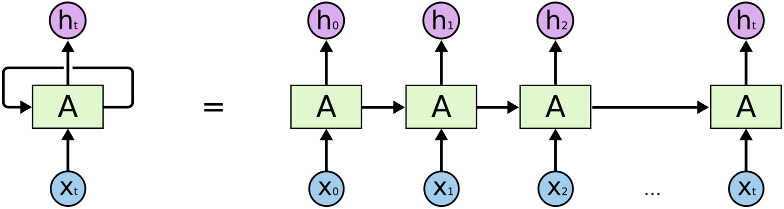
where V is the size of the vocabulary and f is the weighting function defined as:



$$f(X_{ij}) = \begin{cases} \left(\frac{X_{ij}}{x_{max}} \right)^{\alpha} & \text{if } X_{ij} < x_{max} \\ 1 & \text{otherwise} \end{cases}$$

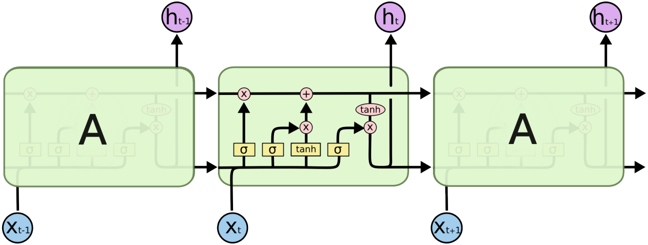
Recurrent Neural Networks (RNNs)

- When we read a text, we understand each word based on our understanding of previous words. We don't throw everything away and start thinking from scratch again. Our thoughts have persistence.
- Recurrent neural networks are networks with loops in them, allowing information to persist.
- A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.
- This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists.



Long Short-Term Memory (LSTM) Networks

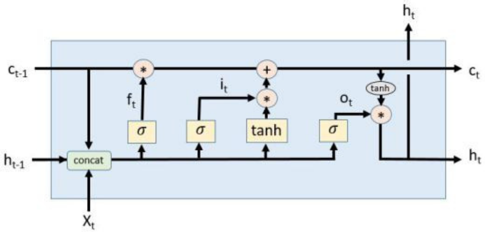
- LSTMs are a special kind of RNN, capable of learning long-term dependencies.
- LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn.
- LSTMs networks have the form of a chain of repeating modules of neural network.



The repeating module in an LSTM contains four interacting layers.

Long Short-Term Memory (LSTM) Networks

- The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.
- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.
- Gates are a way to optionally let information through.



LSTM cell.

Long Short-Term Memory (LSTM) Networks

There are three types of gates within a unit:

- The *Input Gate* i_t that conditionally decides which values from the input to update the memory state.
- The *Forget Gate* f_t that conditionally decides what information to throw away from the block.
- The *Output Gate* o_t that conditionally decides what to output based on input and the memory of the block.

All three of the gates depend on the previous output h_{t-1} on the current input x_t.

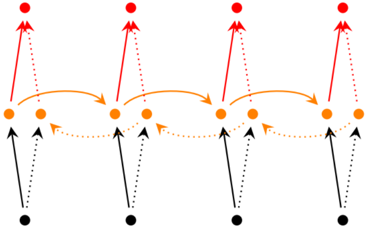
Long Short-Term Memory (LSTM) Networks

These are the functions of the LSTM cell, where $\sigma(\cdot)$ is the sigmoid function.

$$\begin{aligned}
i(t) &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
f(t) &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
c_t &= f_t * c_{t-1} + i_t * \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\
o(t) &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
h_t &= o_t * \tanh(c_t)
\end{aligned}$$

Bidirectional LSTM

- BLSTM neural network is similar to LSTM network in structure because both of them are constructed with LSTM units.
- The principle of BLSTM is to split the neurons of a regular LSTM into two directions, one for positive time direction (forward states), and another for negative time direction (backward states).
- By this structure, the output layer can get information from past and future states.
- The final output can be obtained through some operation: sum, concatenation, average, etc.



Dataset

- For our experiments, the dataset we used is the CoNLL 2010 Shared Task dataset on Wikipedia, one of the benchmark datasets for hedge cue resolution.

- The Wikipedia paragraphs were selected based on the hedge cue (called weasels in Wikipedia) tags that were added by the Wikipedia editors, which were subsequently manually annotated.

- A sentence is considered uncertain if it contains at least one weasel cue.

Certain	**Uncertain**	**Total**
8627	2484	11111

Proportion of class ratios.

Preprocessing

We used the following preprocessing of the dataset before feeding it to the NN:

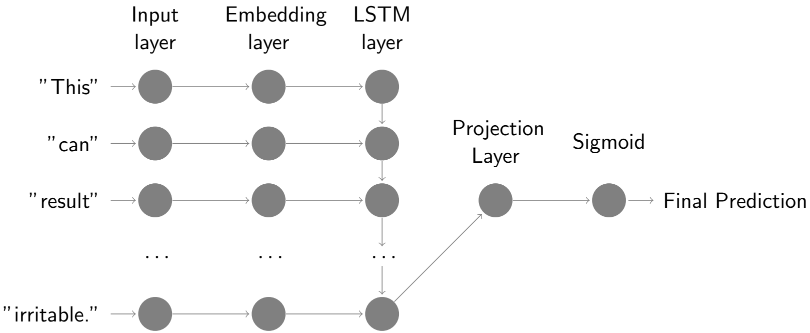
- Lowercase words.
- Removal of XML elements and punctuation.
- Tokenization.
- Construction of the vocabulary considering only the 27.000 most frequent words.

Binary Sentences Classifier Model

The model used for the classification task is structured as follows:

1. Each sentence is represented into a real vector domain.
2. An embedding layer that converts each word into the corresponding embedding vector; the initial weights are given from GloVe features, scaled in $[-1, 1]$ interval, or randomly chosen.
3. An LSTM or BLSTM layer.
4. A finally fully-connected neuron with sigmoid as activation function. This last layer takes the scalar probability $P(positive|sentence)$ given at the last timestep output of the LSTM layer.

Binary Sentences Classifier Model



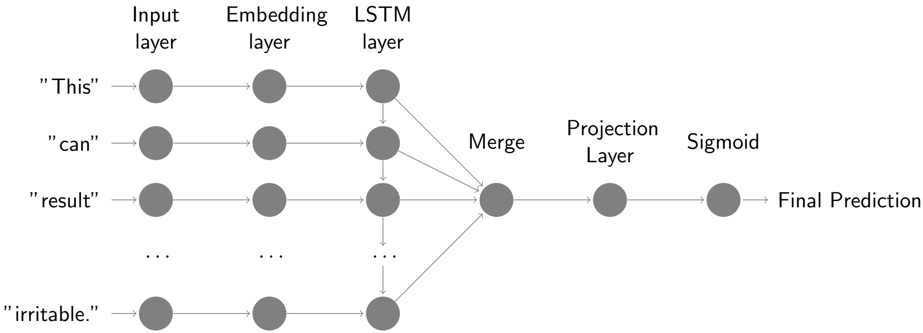
- For BLSTM version we have a Merge layer before Projection layer that receive two input, first relative to the forward propagation and the second relative to the backward propagation.

In-Sentence Cues Detection Model

The model used for the in-sentence cues detection task is structured as follows:

1. Each sentence is represented into a real vector domain.
2. An embedding layer that converts each word into the corresponding embedding vector; the initial weights are given from GloVe features, scaled in $[-1, 1]$ interval, or randomly chosen.
3. An LSTM or BLSTM layer.
4. A Merge layer takes the output given at each timestep of the LSTM layer and gives in output the average value.
5. A finally fully-connected neuron with sigmoid as activation function. This last layer takes the scalar probability given by previous layer.

In-Sentence Cues Detection Model



- For BLSTM version we have another Merge layer, that combines the output of the forward and the backward LSTMs.

Training

- To train the model we minimize the binary cross-entropy loss:

$$\text{Loss} = -\sum_{i=1}^{N} y^{(i)} \log \widetilde{y}^{(i)} + \left(1 - y^{(i)}\right)\left(1 - \log \widetilde{y}^{(i)}\right)$$

  where N is the number of sentences, $\widetilde{y}^{(i)}$ is the value of the sigmoid layer for the sentence i and $y^{(i)}$ is 1 if the sentence is uncertain, 0 if is certain.

- The optimizer used was Adadelta (An Adaptive Learning Rate Method) with Keras default parameters. Other optimizers, like Adam and RMSProp, have been tested for this tasks, but it seems that learning curves didn't change so much.

- In addition we used dropout before the (B)LSTM layer, and before the projection layer.

Experimental Settings

The mini-batch size was fixed to 16 and the results were obtained (for some experiments only) by performing a stratified k-fold, with $k = 5$, and taking the median value.

To overcome the unbalanced nature of the dataset, we have extended the training set by replicating uncertain sentences. The minibatches were built simply by putting in half certain and half uncertain sentences.

The tests were performed by varying the following hyperparameters:

- Number of LSTM layer's cells.
- Dropout keep probability.
- Embedding weights initialization: with or without GloVe. In the second case, we use the Wikipedia pre-trained 50-dimensional word vectors.

Results on Sentences Classifier

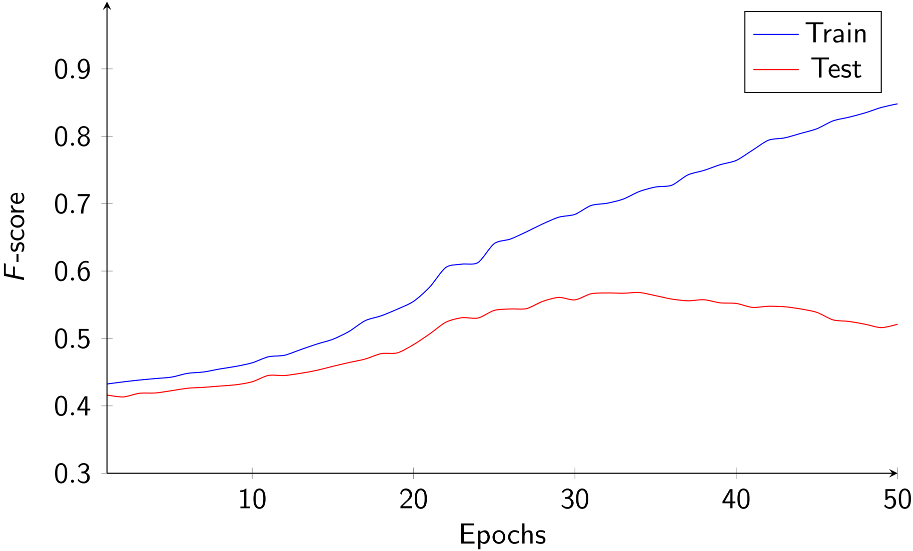
Dropout	LSTM	Loss Train	Loss Test	F-score Train	F-score Test	Epoch
0.6	20	0.32	0.51	0.74	0.54	39
0.6	25	0.29	0.50	0.76	0.55	37
0.6	30	0.34	0.49	0.72	0.56	31
0.6	35	0.36	0.49	0.70	**0.57**	32
0.7	20	0.43	0.59	0.66	0.51	49
0.7	25	0.36	0.50	0.71	0.55	47
0.7	30	0.35	0.54	0.71	0.53	47
0.7	35	0.38	0.52	0.69	**0.55**	50

Results on unidirectional model with GloVe

Comments

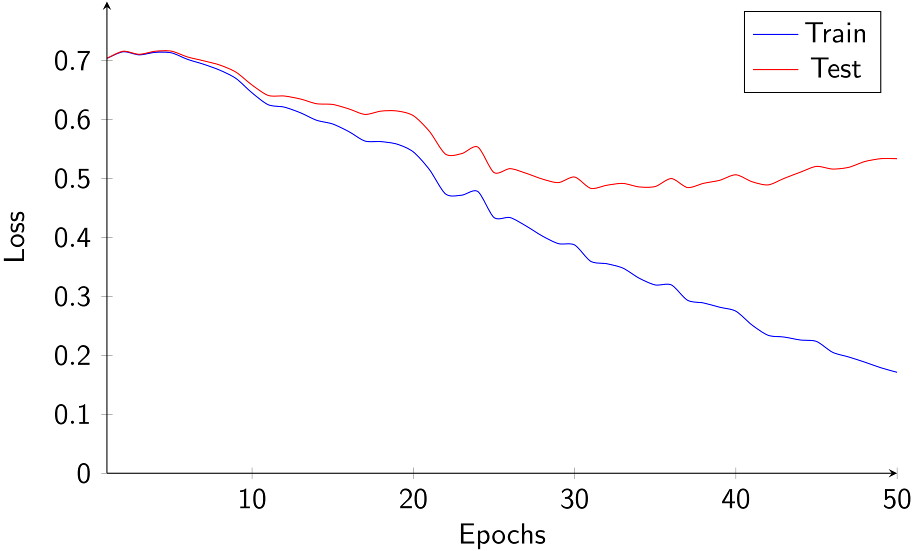
- With dropout 0.6 the model reaches the best performance in few epochs, while the number of LSTM units didn't influence very much.
- Changing the dropout keep probability doesn't improve much the F-score, but the model needs much epochs to converge.
- The best results of F-score is obtained with 35 LSTM units.

Learning Curves on Unidirectional Model



Results with dropout 0.6, best *F*-score 0.57 at epoch 32.

Learning Curves on Unidirectional Model



Results with dropout 0.6, and 35 LSTM units.

Results on Sentences Classifier (BLSTM)

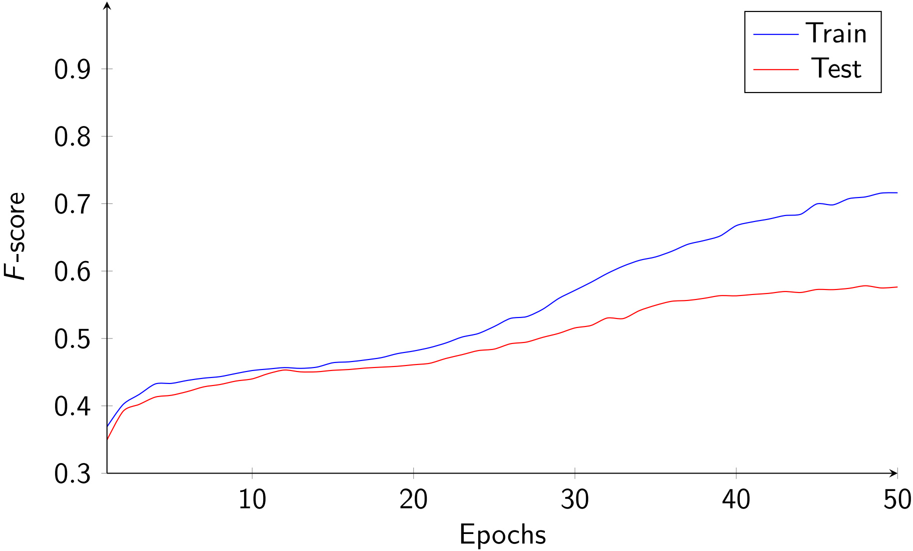
GloVe	Dropout	LSTM	Loss Train	Loss Test	F-score Train	F-score Test	Epoch
yes	1	20	0.19	0.43	0.82	0.57	8
yes	0.6	20	0.30	0.52	0.75	**0.59**	42
no	0.6	20	0.15	0.65	0.87	0.47	31
yes	0.6	25	0.30	0.43	0.72	**0.59**	5
no	0.6	25	0.29	0.53	0.76	0.52	19
yes	0.6	30	0.23	0.47	0.79	**0.58**	7
no	0.6	30	0.30	0.55	0.76	0.53	19
yes	0.6	35	0.23	0.48	0.79	**0.59**	7
no	0.6	35	0.27	0.55	0.78	0.54	21
yes	0.7	10	0.35	0.49	0.72	**0.58**	50
yes	0.7	15	0.40	0.54	0.67	0.56	50
yes	1	20	0.33	0.42	0.69	0.58	4
yes	0.7	20	0.23	0.62	0.81	**0.58**	4
no	0.7	20	0.33	0.42	0.69	0.47	40
yes	0.7	25	0.29	0.44	0.74	**0.60**	6
no	0.7	25	0.25	0.60	0.79	0.49	38
yes	0.7	30	0.29	0.45	0.73	**0.57**	5
no	0.7	30	0.30	0.60	0.76	0.51	33
yes	0.7	35	0.29	0.45	0.74	**0.58**	5
no	0.7	35	0.27	0.59	0.78	0.51	35

Results on bidirectional model

Comments

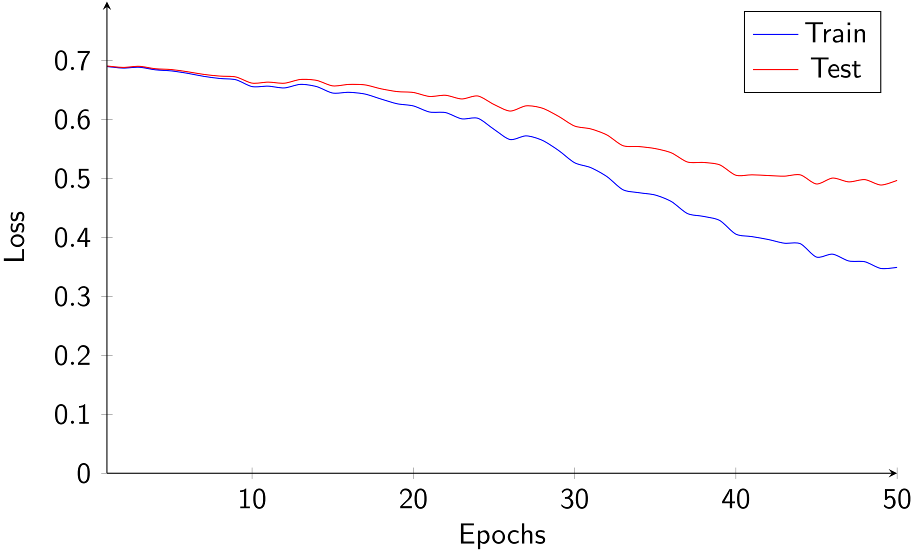
- The number of LSTM units affect much on the network learning speed, in case we have the GloVe features.
- By increasing the number of LSTM units, the network reaches maximum performance in a few epochs.
- With the same number of LSTM units the network requires a greater number of epochs to reach the maximum performance.
- BLSTM networks generally performs better than simple LSTM networks, with equal number of units for each direction, because the output layer get information from past and future states.
- As in the unidirectional model, the dropout doesn't affect much on performance.
- For models that converge very quickly, we tried to execute them without GloVe features, and we have noticed that, further the number of units LSTM, also GloVe features affecting the way of learning network, like the speed of learning and accuracy in the learning phase.

Learning Curves on Bidirectional Model



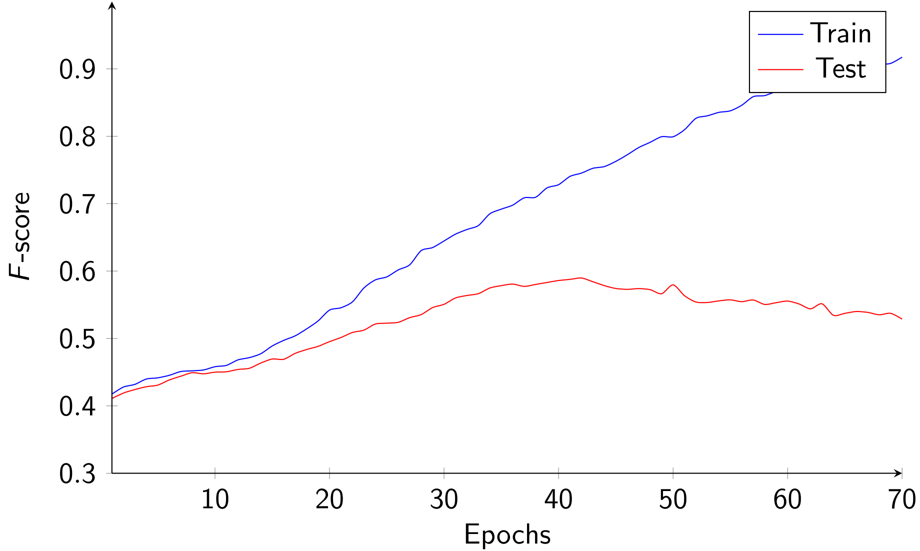
Results with dropout 0.7, 10 LSTM units, and best *F*-score 0.58 at epoch

Learning Curves on Bidirectional Model



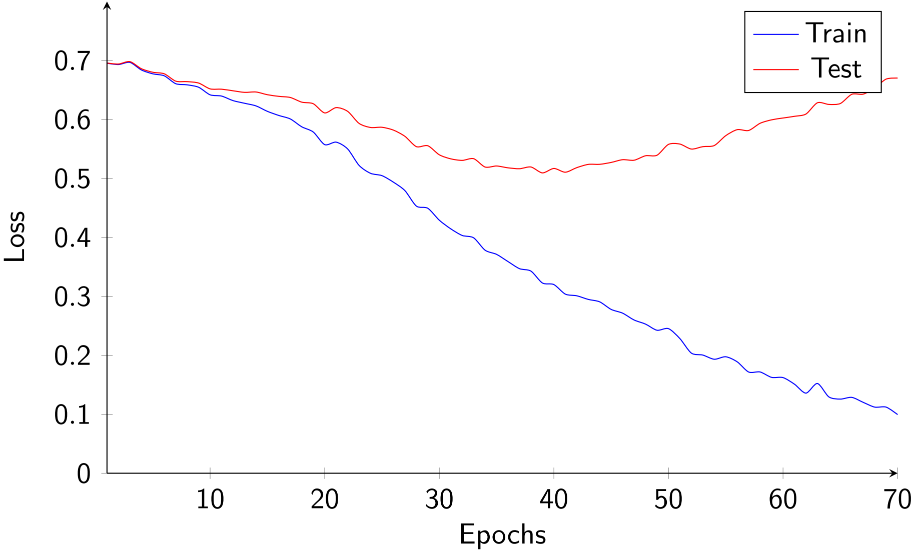
Results with dropout 0.7 and 10 LSTM units

Learning Curves on Bidirectional Model



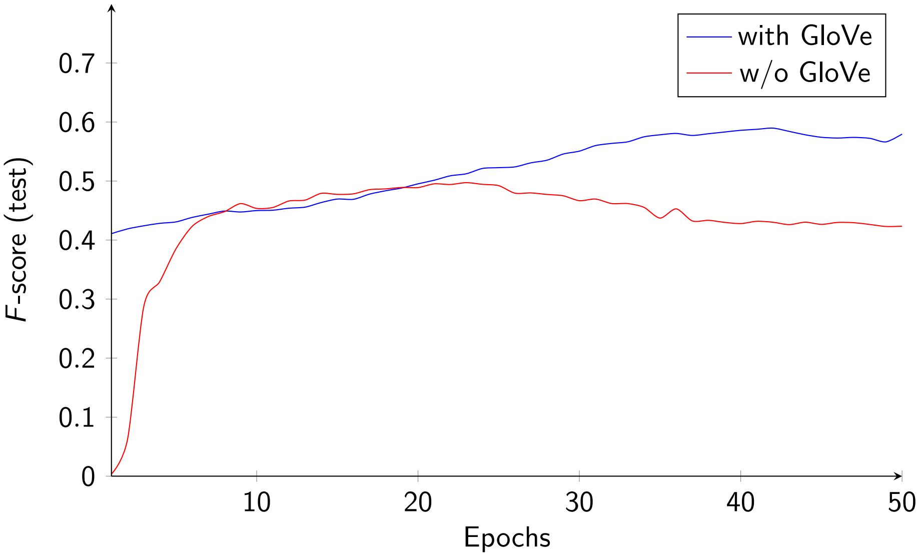
Results with dropout 0.6, 20 LSTM units, and best *F*-score 0.57 at epoch

Learning Curves on Bidirectional Model



Results with dropout 0.6 and 20 LSTM units

Learning Curves with and without GloVe



Results with dropout 0.6, 20 LSTM units, and best *F*-score 0.57 at epoch

Results for In-Sentence Cue Detection

Best Cue Threshold	Accuracy	Precision	Recall	F-score
0.31	0.89	0.19	0.59	0.28

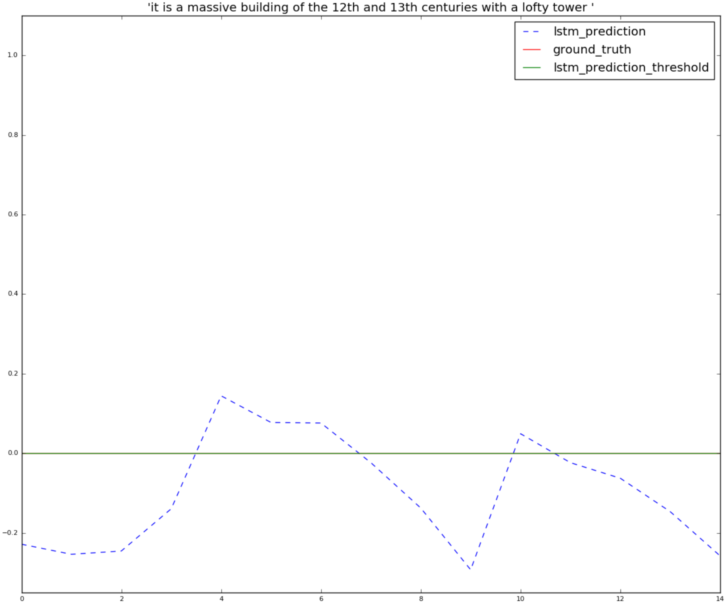
Unidirectional model with *F*-score 0.54, dropout 0.6, LSTM units 15

Best Cue Threshold	Accuracy	Precision	Recall	F-score
0.46	0.91	0.29	0.36	0.32

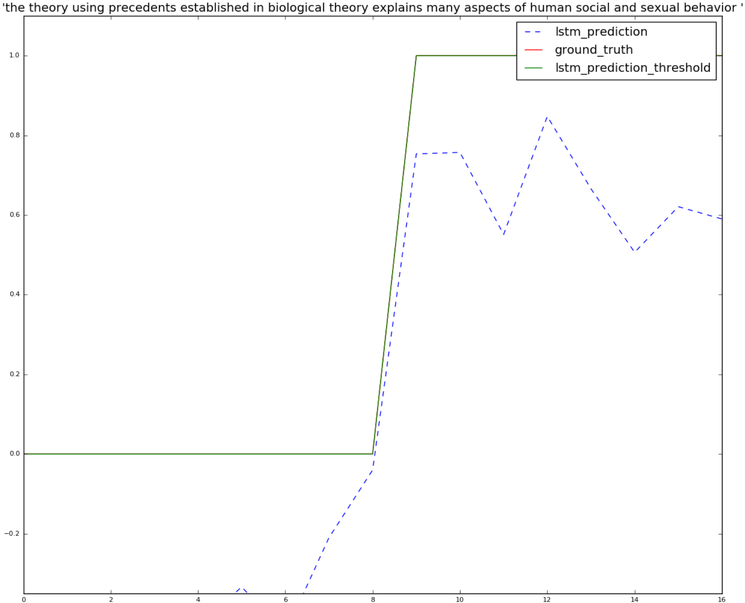
Bidirectional model with *F*-score 0.56, dropout 0.6, LSTM units 10

Bidirectional model performs much better for in-sentence cue detections, maybe with backward passes it collect some important features on syntactic information.

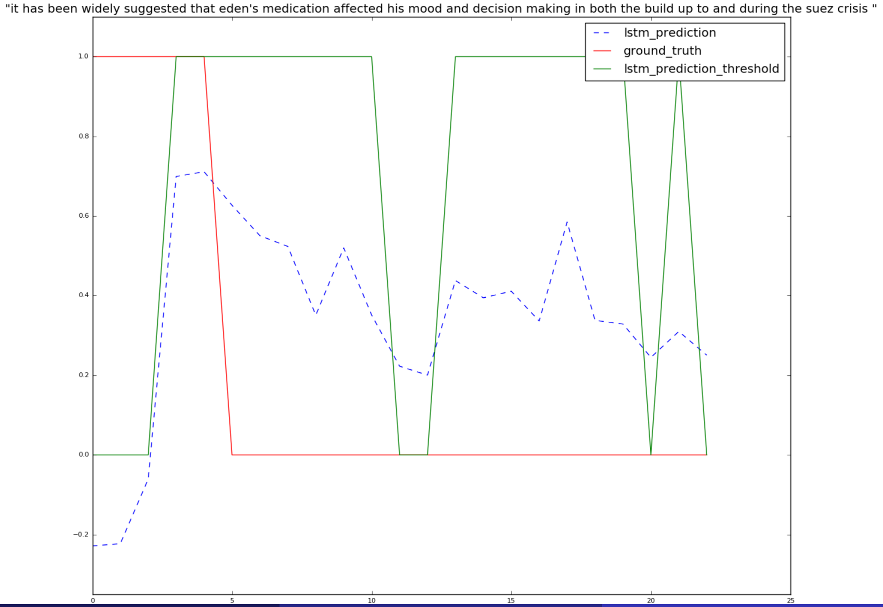
Results for In-Sentence Cue Detection



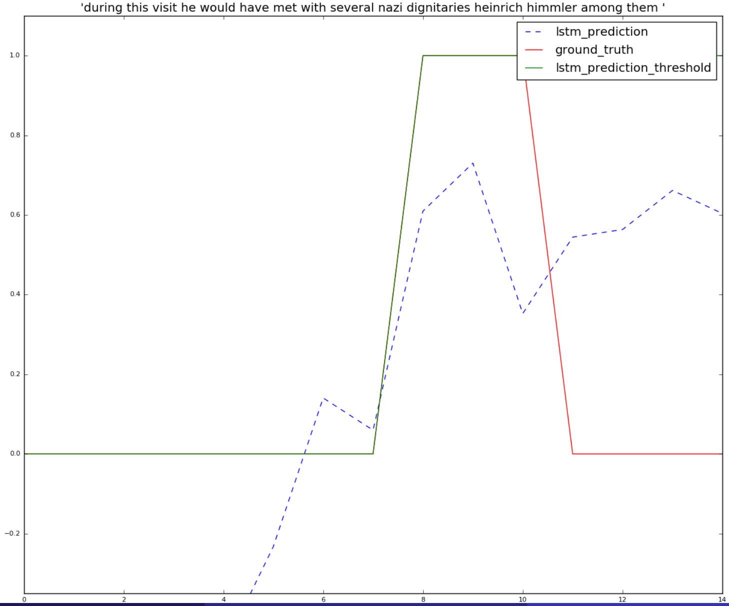
Results for In-Sentence Cue Detection



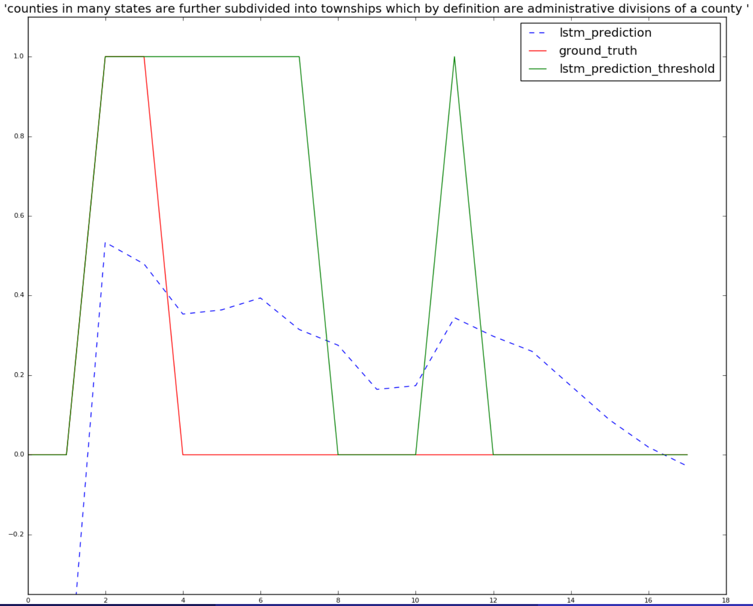
Results for In-Sentence Cue Detection


"it has been widely suggested that eden's medication affected his mood and decision making in both the build up to and during the suez crisis "

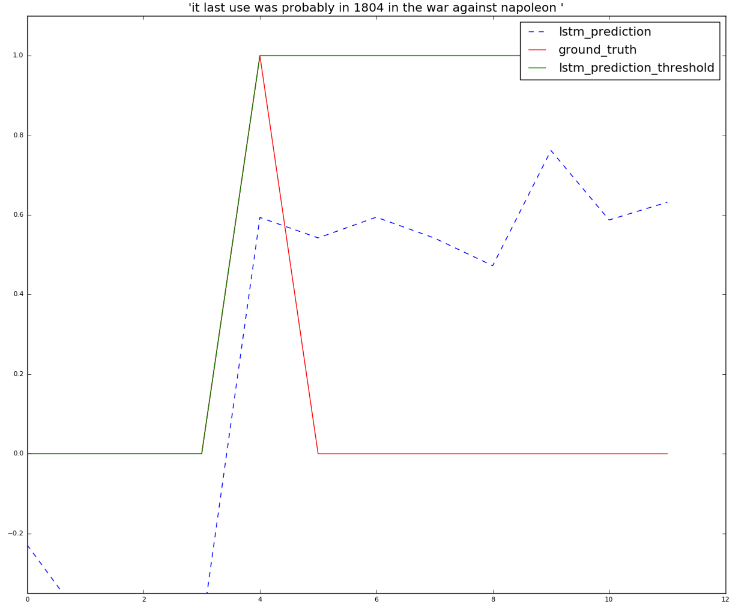
Results for In-Sentence Cue Detection



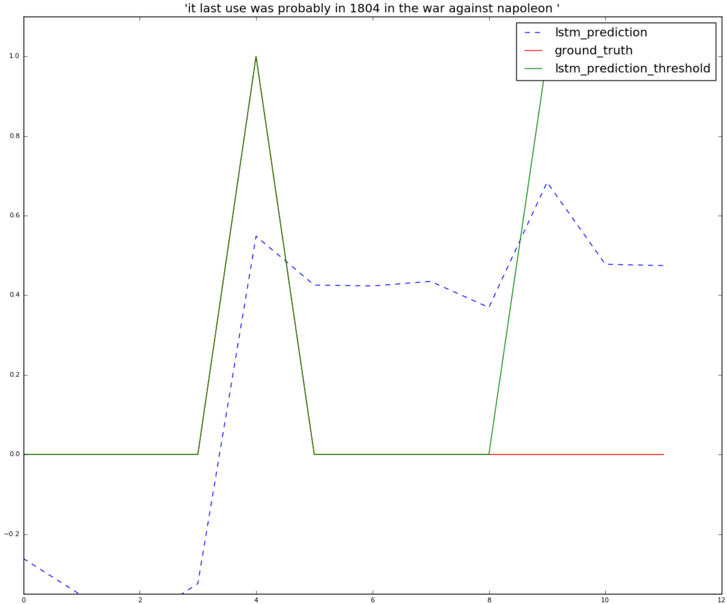
Results for In-Sentence Cue Detection



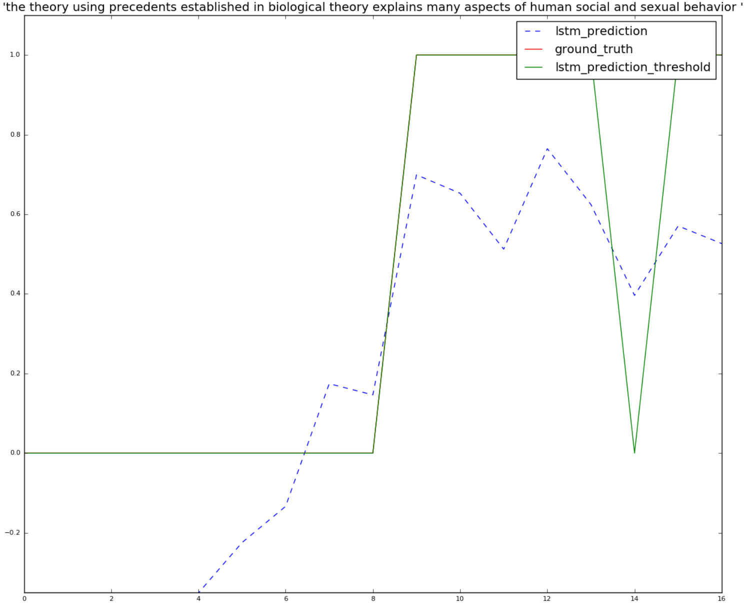
Results for In-Sentence Cue Detection



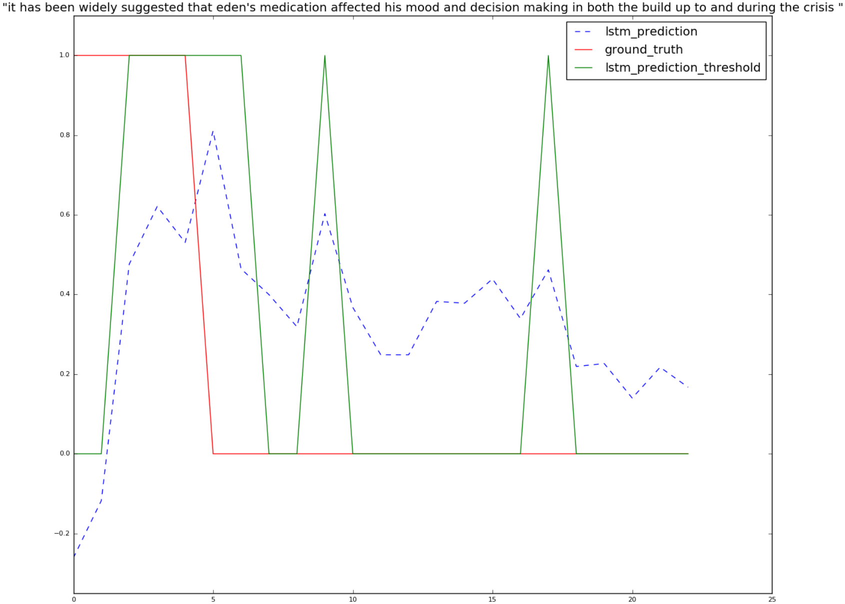
Results for In-Sentence Cue Detection (BLSTM)



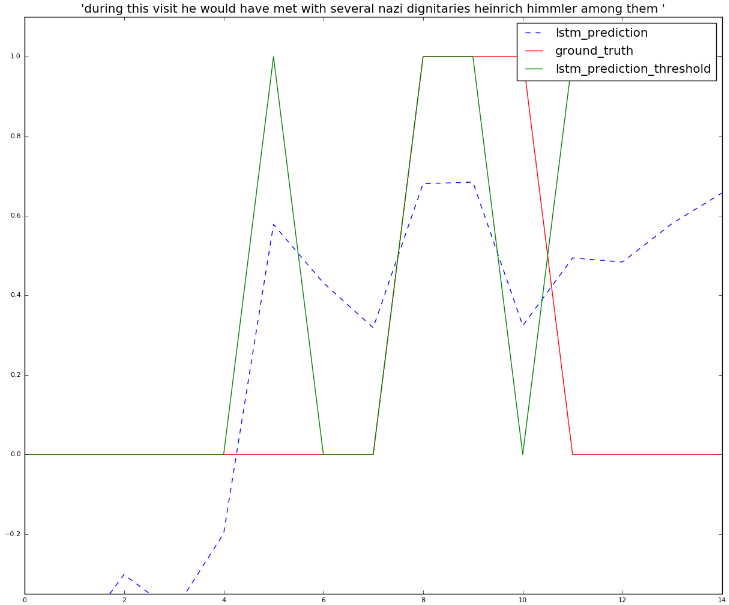
Results for In-Sentence Cue Detection (BLSTM)



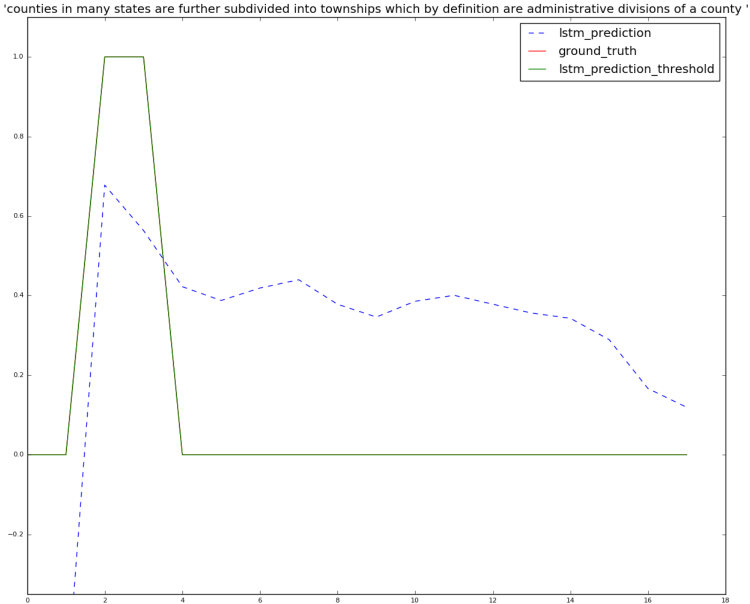
Results for In-Sentence Cue Detection (BLSTM)



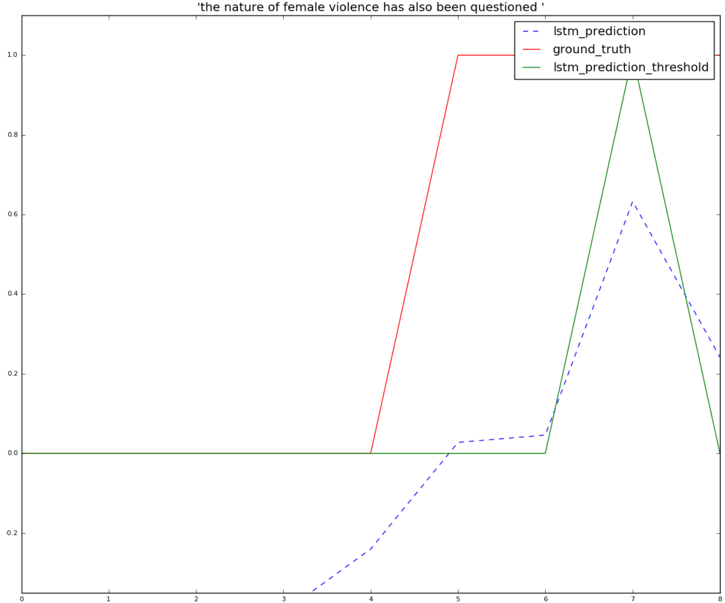
Results for In-Sentence Cue Detection (BLSTM)



Results for In-Sentence Cue Detection (BLSTM)



Results for In-Sentence Cue Detection (BLSTM)



Conclusions and Further Work

- In this project we have shown how RNNs, particularly LSTMs, can be used to classify sentences with satisfying results.
- Moreover, embedding initial weights, learned by GloVe vectors, can mildly improve the learning speed and *F*-score of our model.
- Bidirectional model generally improve performances w.r.t. unidirectional, but the training is more slow.
- Contrary to intuition, the bidirectional model is able to capture information even looking the sentence in reverse.

Future developments can be:

- Test the model on a different dataset.
- Optimize hyperparameters using a *k*-fold cross-validation on all cases.
- Variable threshold for in-sentence cue detection based on LSTM's output weights.