# 7 Appendix

We structure the Appendix in the following way. We first provide proofs of the propositions in Section 3. Second, we give details (through the pseudo-code) of the Algorithms and subroutines that were used to find Optimal Robust Explanations: in particular we describe the *shrink* (used to improve MSA) and the Adversarial Attacks procedures (used to improve HS). We then provide details on the datasets and the architectures that we have used in the Experimental Evaluation, and finally we report many examples of interesting OREs that we were able to extract with our methods, alongside with tables that complete the comparison between MSA and HS as described in the Experimental Evaluation Section.

## 7.1 Proofs

**Proof of Prop. 2** Call $A =$ *"*$M(x)$ *is biased"* and $B =$ *"*(4) *is infeasible under* $F' \cap E = \emptyset$*"*. Let us prove first that $B \rightarrow A$. Note that $B$ can be equivalently expressed as

$$\forall E \subseteq F.(E \cap F' \neq \emptyset \vee \exists x' \in \mathcal{B}_E(t).M(x) \neq M(x'))$$

If the above holds for all $E$ then it holds also for $E = F \setminus F'$, and so it must be that $\exists x' \in \mathcal{B}_{F \setminus F'}(t).M(x) \neq M(x')$ because the first disjunct is clearly false for $E = F \setminus F'$.

We now prove $A \rightarrow B$ by showing that $\neg B \rightarrow \neg A$. Note that $\neg B$ can be expressed as

$$\exists E \subseteq F.(E \cap F' = \emptyset \wedge \forall x' \in \mathcal{B}_E(t).M(x) = M(x')), \quad (6)$$

and $\neg A$ can be expressed as

$$\forall x' \in \mathcal{B}_{F \setminus F'}(t).M(x) = M(x'). \quad (7)$$

To see that (6) implies (7), note that any $E$ that satisfies (6) must be such that $E \cap F' = \emptyset$, which implies that $E \subseteq F \setminus F'$, which in turn implies that $\mathcal{B}_{F \setminus F'}(t) \subseteq \mathcal{B}_E(t)$. By (6), the prediction is invariant for any $x'$ in $\mathcal{B}_E(t)$, and so is for any $x'$ in $\mathcal{B}_{F \setminus F'}(t)$.

**Proof of Prop. 3** A robust explanation $E \subseteq F$ guarantees prediction invariance for any $x' \in \mathcal{B}_E(t)$, i.e., for any $x'$ (in the support of $\mathcal{D}$) to which anchor $A_E$ applies.

**Proof of Prop. 4** For discrete $\mathcal{D}$ with pmf $f_{\mathcal{D}}$, we can express $\mathsf{cov}(A_E)$ as

$$\mathsf{cov}(A_E) = \sum_{x' \in supp(\mathcal{D})} f_{\mathcal{D}}(x') \cdot \mathbf{1}_{A_E(x')} = \sum_{x' \in supp(\mathcal{D})} f_{\mathcal{D}}(x') \cdot \prod_{w \in E} \mathbf{1}_{x'_w = \mathcal{E}(w)}$$

To see that, for $E' \supseteq E$, $\mathsf{cov}(A_{E'}) \leq \mathsf{cov}(A_E)$, observe that $\mathsf{cov}(A_{E'})$ can be expressed as

$$\mathsf{cov}(A_{E'}) = \sum_{x' \in supp(\mathcal{D})} f_{\mathcal{D}}(x') \cdot \prod_{w \in E'} \mathbf{1}_{x'_w = \mathcal{E}(w)} = \sum_{x' \in supp(\mathcal{D})} f_{\mathcal{D}}(x') \cdot \prod_{w \in E} \mathbf{1}_{x'_w = \mathcal{E}(w)} \cdot \prod_{w \in E \setminus E'} \mathbf{1}_{x'_w = \mathcal{E}(w)}$$

and that for any $x'$, $\prod_{w \in E \setminus E'} \mathbf{1}_{x'_w = \mathcal{E}(w)} \leq 1$.

**Proof of Prop. 1** With abuse of notation, in the following we use $C^*$ to denote both an ORE and its logical encoding.

1. if $C^*$ is an ORE, then $\phi \equiv (B \wedge \mathcal{N}) \rightarrow \hat{y}$ is true for *any* assignment $x'$ of the features not in $C^*$. In particular, $\phi$ is trivially satisfied for any $x'$ outside the perturbation space $B$, and, by Definition 1, is satisfied for any $x'$ within the perturbation space.
2. As also explained in [Dillig *et al.*, 2012], finding an optimal $C^*$ such that $C^* \models \phi$ is equivalent to finding an MSA $C^*$ for $\phi$. We should note that $C^*$ is a special case of an MSA, because the possible assignments for the variables in $C^*$ are restricted to the subsets of the cube $C$.
3. $C^*$ is said a prime implicant of $\phi$ if $C^* \models \phi$ and there are no proper subsets $C' \subset C^*$ such that $C' \models \phi$. This holds regardless of the choice of the cost $\mathcal{C}$, as long as it is additive and assigns a positive cost to each feature as per Definition 2. Indeed, for such a cost function, any proper subset $C' \subset C^*$ would have cost strictly below that of $C^*$, meaning that $C' \not\models \phi$ (i.e., is not robust) because otherwise, $C'$ (and not $C^*$) would have been (one of) the robust explanations with minimal cost.

## 7.2 Optimal Cost Algorithms and Sub-Routines

In this Section we provide a full description and the pseudo-code of the algorithms that for reason of space we were not able to insert in the main paper. We report a line-by-line description of the HS procedure (Algorithm 1): we further describe how the adversarial attacks procedure is used to generate candidates that help the HS approach converge on hard instances, as reported in Section 4. We then describe the algorithm to compute Smallest Cost Explanations (Algorithm 4). In Algorithm 5, we finally detail the *shrink* procedure as sketched in Section 3.

**Minimal Hitting-Sets and Explanations** One way to compute optimal explanations against a cost function $C$, is through the hitting set paradigm [Ignatiev *et al.*, 2019a], that exploits the relationship between diagnoses and conflicts [Reiter, 1987]: the idea is to collect perturbations and to calculate on their indices a minimum hitting set (MHS) i.e., a minimum-cost explanation whose features are in common with all the others. We extend this framework to find a word-level explanation for non-trivial NLP models. At each iteration of Algorithm 1, a minimum hitting set $E$ is extracted (line 3) from the (initially empty, line 1) set $\Gamma$. If function *Entails* evaluates to *False* (i.e., the neural network $\mathcal{N}$ is provably safe against perturbations on the set of features identified by $F \setminus F$) the procedure terminates and $E$ is returned as an ORE. Otherwise, (at least) one feasible attack is computed on $F \setminus E$ and added to $\Gamma$ (lines 7-8): the routine then re-starts. Differently from [Ignatiev *et al.*, 2019a], as we have experienced that many OREs whose a large perturbation space - i.e. when $\epsilon$ or $k$ are large - do not terminate in a reasonable amount of time, we have extended the *vanilla* hitting set approach by introducing *SparseAttacks* function (line 7). At each iteration *SparseAttacks* introduces in the hitting set $\Gamma$ a large number of sparse adversarial attacks on the set of features $F \setminus E$: it is in fact known [Ignatiev *et al.*, 2016] that

attacks that use as few features as possible help convergence on instances that are hard (intuitively, a small set is harder to "hit" hence contributes substantially to the optimal solution compared to a longer one) *SparseAttacks* procedure is based on random search and it is inspired by recent works in image recognition and malaware detection [Croce *et al.*, 2020]: pseudo-code is reported in 2, while a detailed description follows in the next paragraph.

**Sparse Adversarial Attacks** In Algorithm 2 we present a method to generate sparse adversarial attacks against features (i.e., words) of a generic input text. $GeneratePerturbations(k, n, Q)$ (line 2) returns a random population of $n$ perturbations that succeed at changing $\mathcal{N}$'s classification: for each successful attack $p$, a subset of $k$ out of $d$ features has been perturbed through a Fast Gradient Sign attack [8] (FGSM), while it is ensured that the point lies inside a convex region $Q$ which in our case will be the $\epsilon$ hyper-cube around the embedded text. If no perturbation is found in this way (i.e., population size of the atacks is zero, as in line 3), budget is decreased (line 4) and another trial of $GeneratePerturbations(k, n, Q)$ is performed (e.g., with few features as targets and a different random seed to guide the attacks). Function $AccuracyDrop(\mathcal{N}, P)$ returns the best perturbation $a$ where $k$ is increasingly minimised (line 7). Algorithm terminates when either no attacks are possible (all the combinations of features have been explored) or after fixed number of iterations has been performed (line 1).

**Algorithm 1:** ORE computation via implicit hitting sets and sparse attacks

**Data:** a network $\mathcal{N}$, the input text $t$, the initial set of features F, a network prediction $\hat{y}$ , a cost function $\mathcal{C}$ against which the explanation is minimised
**Result:** an optimal ORE E

```
1  Γ = ∅
2  while true do
3      E = MinimumHS(Γ, C)
4      if Entails(E, (N ∧ B_{F\E}(t)) → ŷ) then
5          return E
6      else
7          A = SparseAttacks(E, N)
8          Γ = Γ ∪ {A}
```

**Minimum Satisfying Assignment Explanations** This approach, based on the method presented in [Dillig *et al.*, 2012], finds an explanation in the form of an MSA, for which in turn a maximum universal subset (MUS) is required. For a given cost function $\mathcal{C}$ and text $t$, an MUS is a universal subset $t'$ of words that maximises $\mathcal{C}(t')$. An MSA of the network $M$ w.r.t the text is precisely a satisfying assignment of the formula $\forall_{w \in t'}.M \rightarrow \hat{y}$ for some MUS $t'$. In other words, an MSA is $t \setminus t'$. The inputs to the MSA algorithm are: $\mathcal{N}$ which represents the network $M$ in constraint form; text $t$; cost function $\mathcal{C}$ and prediction $\hat{y}$ for the input $t$. The algorithm first uses the reversed sort function for the text $t$ to optimize the search tree. The text is sorted by the cost of each word. then uses the recursive MUS algorithm to compute an MUS $t'$. Finally, the optimal explanation ($t \setminus t'$) is returned.

[8] https://www.tensorflow.org/tutorials/generative/adversarial_fgsm

**Algorithm 2:** Computing a perturbation that is successful and minimises the number of features that are perturbed.

**Data:** $\mathcal{N}$ - neural network model, $F$ - input text from feature space; $k \in \mathbb{N}^{+}_{\setminus\{0\}}$ - number of perturbations initially tested; $Q \subseteq F$ - (sub)set of features where perturbations are found; $n \in \mathbb{N}^{+}_{\setminus\{0\}}$ - number of elements generated at each iteration of the algorithm; budget - number of iterations allowed before stopping.

```
1  while k > 0 ∧ budget > 0 do
2      P ← GeneratePerturbations(k, n, Q)
3      if length(P) == 0 then
4          budget ← budget − 1
5          continue
6      end
7      a ← arg max_{p∈P} AccuracyDrop(M, P)
8      k ← k − 1, budget ← budget − 1
9  end
10 return a
```

The inputs of the $mus$ algorithm are: a set of candidate words $cW$ that an MUS should be calculated for (equal to $t$ in the first recursive call), a set of bounded words $bW$ that may be part of an MUS, where $\forall_{w \in bW}$, $w$ may be limited by $\epsilon$-ball or $k$-NN box clousure, a lower bound $L$, the network $\mathcal{N}$, a cost function $\mathcal{C}$, and a network prediction $\hat{y}$. It returns a maximum-cost universal set for the network $\mathcal{N}$ with respect to $t$, which is a subset of $cW$ with a cost greater than $L$, or the empty set when no such subset exists. The lower bound allows us to cut off the search when the current best result cannot be improved. During each recursive call, if the lower bound cannot be improved, the empty set is returned (line 1). Otherwise, a word $w$ is chosen from the set of candidate words $cW$ and it is determined whether the cost of the universal subset containing word $w$ is higher than the cost of the universal subset without it (lines 5-12). Before definitively adding word $w$ to $bW$, we test whether the result is still satisfiable with $Entails$ (line 5) i.e. still an explanation. The $shrink$ method helps to reduce the set of candidate words by iterating through current candidates and checking using $Entails$ whether they are necessary. This speeds-up the algorithm (as there are fewer overall calls to $Entails$). The recursive call at line 6 computes the maximum universal subset of $\forall_{w \in bW} \mathcal{N} \rightarrow \hat{y}$, with adjusted $cW$ and $L$ as necessary. Finally within this $if$ block, we compute the cost of the universal subset involving word $w$, and if it is higher than the previous bound $L$, we set the new lower bound to cost (lines 7-11). Lines 11-12 considers the cost of the universal subset *not* containing word $w$, in case it has higher cost, and if so, updates $best$. Once one optimal explanation has been found,

**Algorithm 3:** MUS computation, $mus(bW, \mathcal{N}, cW, t, \mathcal{C}, L, \hat{y})$

**Data:** a list of bounded words $bW$, a network $\mathcal{N}$, a set of candidate words $cW$, the input text $t$, a cost function $\mathcal{C}$ against which the ORE is minimised, a lower bound for MUS $L$, a prediction $\hat{y}$ for the input
**Result:** a Maximum Universal Subset with respect to input text $t$

```
if cW = ∅ or C(cW) ⩽ L then return ∅
best = ∅
choose w ∈ cW
bW = bW ∪ {w}, constW = cW \ {w}
if Entails(constW, (N ∧ B_{F\E}(constW)) → ŷ) then
    Y = mus(bW, N, shrink(N, bW, cW \ {w}), t, C, L − C(w), ŷ)
    cost = C(Y) + C(w)
    if cost > L then
        best = Y ∪ {w}
        L = cost
Y = mus(bW \ {w}, N, cW \ {w}, t, C, L, ŷ)
if C(Y) > L then best = Y
return best
```

it is possible to compute *all combinations* of the input that match that cost, and then use $Entails$ on each to keep only those that are also explanations.

**Comparing MHS and MSA** The MSA-based approach uses MUS algorithm to find maximum universal subset and then finds a MSA for that MUS. MUS is a recursive branch-and-bound algorithm [Dillig *et al.*, 2012] that explores a binary tree structure. The tree consists of all the word appearing in the input cube. The MUS algorithm possibly explores an exponential number of universal subsets, however, the recursion can be cut by using right words ordering (i.e. words for which robustness query will answer false, consider words with the highest cost first) or with shrink method. MUS starts to work with a full set of candidate words, whereas the HS approach starts with an empty set of fixed words and tries to find an attack for a full set of bounded words. In each iteration step, the HS approach increases the set of fixed words and tries to find an attack. It is because a subset $t' \subseteq t$ is an MSA for a classifier $M$ with respect to input text $t$ iff $t'$ is a minimal hitting set of minimum falsifying set (see [Ignatiev *et al.*, 2016] for details). To speed up the MSA algorithm, we use $shrink$ procedure which reduces the set of candidate words, and for non-uniform cost function, words ordering (words with the highest cost are considered as the first candidates), while HS-based approach uses $SparseAttacks$ routine to increase the hitting set faster.

**Excluding words from MSA** To exclude specific words from a smallest explanation we add one extra argument to the MSA algorithm input: the $bW$ which represents bounded words. In this case the set $cW = t \setminus bW$. From now on the procedure is the standard one.

**Algorithm 4:** Computing smallest cost explanation

**Data:** a network $\mathcal{N}$, an input text $t$, a cost function $\mathcal{C}$ for the input $C$, a prediction $\hat{y}$,
**Result:** A smallest cost explanation for network $\mathcal{N}$ w.r.t. input text $t$

```
bW = ∅, cW = C, sce = ∅
textSortedByCost = sort(t)
maxus = mus(bW, N, cW, textSortedByCost, C, 0, ŷ)
foreach c ∈ t do
    if c ∉ maxus then
        sce = sce ∪ c
    end
end
return sce
```

**Algorithm 5:** $shrink$ algorithm $shrink(bW, \mathcal{N}, cW, C, \mathcal{C}, L, \hat{y})$

**Data:** a list of bounded words $bW$, a network $\mathcal{N}$, a set of candidate words $cW$, a text $t$, a cost function $\mathcal{C}$, a lower bound $L$, a prediction $\hat{y}$ for the input
**Result:** A set of the essential candidate words eW

```
eW = cW
foreach word ∈ cW do
    eW = eW \ {word}
    bW = bW ∪ {word}
    constW = C \ bw
    if Entails(constW, (N ∧ B_{F\E}(cW)) → ŷ) then
        eW = eW ∪ {word}
    end
    bW = bW \ {word}
end
return eW
```

## 7.3 Details on the Experimental Results

**Datasets and Test Bed**

As mentioned in the Experimental Evaluation Section, we have tested MSA and HS approaches for finding optimal cost explanations respectively on the SST, Twitter and IMDB datasets. For each task, we have selected a sample of 40 input texts that maintain classes balanced (i.e., half of the examples are *negative*, half are *positive*). Moreover, we inserted inputs whose polarity was exacerbated (either very *negative* or very *positive*) as well as more challenging examples that machines usually misclassifies, like *double negations* or mixed sentiments etc. Further details in Table 1.

**Models Setup**

We performed our experiments on FC and CNNs with up to 6 layers and $20K$ parameters. FC are constituted by a stack of *Dense* layers, while CNNs additionally employ *Convolutional* and *MaxPool* layers: for both CNNs and FC the decision is taken through a *softmax* layer, with *Dropout* that is addedd after each layer to improve generalization during the training phase. As regards the embeddings that the models equip, we experienced that the best trade-off between the ac-

'# I've seen Foxy Brown, Coffy Friday Foster Bucktown, and Black Mama White Mama of these this is Pam Griers worst movie poor acting bad script boring action scenes theres just nothing there avoid this and rent Friday Foster Coffy or Foxy Brown instead' (**IMDB, predicted as *negative***)

'# I gave this a 2 and it only avoided a 1 because of the occasional unintentional laugh the film is excruciatingly. Boring and incredibly cheap its even worse if you know anything at all about the Fantastic Four.', (***IMDB, predicted as negative***)

'# a few words for the people here in cine club the worst crap ever seen on this honorable cinema a very poor script a very bad actors and a very bad movie dont waste your time looking this movie see the very good or any movie have been good commented by me say no more' (**IMDB, predicted as *negative***)

Figure 8: Examples of Optimal Robust Explanations - highlighted in blue -. OREs were extracted using kNN boxes with 25 neighbors per-word: fixing words in an ORE guarantees the model to be locally robust. The examples come from the IMDB dataset, model employed is a FC network with 100 input words (accuracy 0.81).

curacy of the network and the formal guarantees that we need to provide is reached with low-dimensional embeddings, thus we employed optimized vectors of dimension 5 for each word in the embedding space: this is in line with the experimental evaluations conducted in [Patel and Bhattacharyya, 2017], where for low-order tasks such as sentiment analysis, compact embedding vectors allow to obtain good performances, as shown in Table 1. We note that techniques such as retrofitting [Faruqui *et al.*, 2014] could allow using more complex representations and might help with high-order tasks such as multi-class classification, where the quality of the embedding plays a crucial role in terms of accuracy. We will consider this as a future extension of the work. We report in Table 1 an overview of the models we used.

## 7.4 Additional Results

In this Section we provide a few interesting results that we couldn't add to the main paper.

### Additional kNN Results

As discussed in Section 5, we have found a few instances where distilling an ORE from an $\epsilon$-bounded input was computationally infeasible, thus motivating us to develop and use the kNN-boxes technique for the majority of the results in this paper. In Figure 10 we compare how OREs grow for increasing values of $\epsilon$ and k (i.e., the control parameters of respectively $\epsilon$-boxes and kNN-boxes). Finally, in Figure 8 we report a few examples of IMDB reviews that we could solve for an FC with 100 input words: those examples show OREs for the largest model - in terms of both input size and parameters - that we could solve by means of HS or MSA, eventually improved with the *Adversarial Attacks* routine.

### $\epsilon$-ball Results

With a perturbation method defined as an $\epsilon$-ball around each input vector (see section 3), Table 2 shows a comparison of ORE length and execution time for both the MSA and HS methods.

Figure 9 shows how using adversarial attacks speeds up convergence.

Below is an example of calculating all possible OREs for a given input and $\epsilon$, and an example of decision bias.

**Example 1** Calculating *all* of the smallest explanations for

| | INPUT INSTANCE | EXECUTION TIME [s] (HS *Vanilla*, HS + Adversarial Attacks, MSA) |
|---|---|---|
| **ε = 0.05** | 'insanely hilarious!' | **87.66, 8.67, 47.56** |
| **ε = 0.05** | 'this one is not nearly as dreadful as expected' | **114.99, 10.49, 0.44** |
| **ε = 0.05** | 'this one is baaaaad movie!' | **Timeout, 79.2, 0.79** |
| **ε = 0.1** | 'so your entire day was spent doing chores ay??!!' [...] | **Timeout, 1520.80, 0.44** |
| **ε = 0.05** | 'I just seen ur tweetz plz write bak' [...] | **Timeout, 159.11, 930.83** |

Figure 9: Examples of explanations that were enabled by the adversarial attacks routine. Timeout was set to 2 hours.

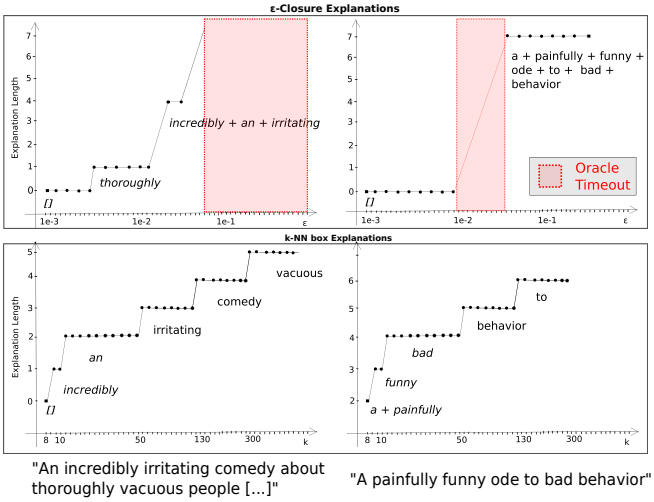


Figure 10: How an explanation grows when either $\epsilon$ (top) or k (bottom) is increased. Model considered is a fully connected with 50 input words on SST dataset (0.89 accuracy). On the left a *negative* review that is correctly classified, on the right a *positive* review that is misclassified (i.e., the model's prediction is *negative*). For specific ranges of $\epsilon$ the Oracle cannot extract an explanation (timeout, highlighted in red).

Table 1.1: Training

| | TWITTER | SST | IMDB |
|---|---|---|---|
| **Inputs (Train, Test)** | $1.55M, 50K$ | $117.22K, 1.82K$ | $25K, 25K$ |
| **Output Classes** | 2 | 2 | 2 |
| **Input Length (max, max. used)** | 88, 50 | 52, 50 | 2315, 100 |
| **Neural Network Models** | FC, CNN | FC, CNN | FC, CNN |
| **Neural Network Layers (min,max)** | 3,6 | 3,6 | 3,6 |
| **Accuracy on Test Set (min, max)** | 0.77, 0.81 | 0.82, 0.89 | 0.69, 0.81 |
| **Number of Networks Parameters (min,max)** | $3K, 18K$ | $1.3K, 10K$ | $5K, 17K$ |

Table 1.2: Explanations

| | TWITTER | SST | IMDB |
|---|---|---|---|
| **Sample Size** | 40 | 40 | 40 |
| **Review Length (min-max)** | 10, 50 | 10, 50 | 25, 100 |

Table 1: Datasets used for training/testing and extracting explanations. We report various metrics concerning the networks and the training phase (included accuracy on Test set), while in Table 1.2 we report the number of texts for which we have extracted explanations along with the number of words considered when calculating OREs: samples were chosen to reflect the variety of the original datasets, i.e., a mix of long/short inputs equally divided into positive and negative instances.

| $\epsilon$ | Explanation Length | MSA Execution Time | HS Execution Time |
|---|---|---|---|
| **0.01** | $5 \pm 5$ | $8.08 \pm 7.9$ | $63.70 \pm 63.69$ |
| **0.05** | $5.5 \pm 4.5$ | $176.22 \pm 175.92$ | $339.96 \pm 334.66$ |
| **0.1** | $7.5 \pm 2.5$ | $2539.75 \pm 2539.14$ | $3563.4 \pm 3535.84$ |

Table 2: Comparison between MSA and HS in terms of execution time for different values of $\epsilon$, and the corresponding explanation length.

an input ($\epsilon = 0.05$, FC network, 10 input words, 5 dimensional embedding, SST dataset):

```
Input: ['strange', 'funny', 'twisted', 'brilliant',
     'and', 'macabre', '<PAD>', '<PAD>', '<PAD>',
    '<PAD>']

 Explanations (5 smallest, len=6.0): ['strange', '
     funny', 'twisted', 'brilliant', '<PAD>', '<PAD
     >'] ['strange', 'funny', 'twisted', '<PAD>',
     '<PAD>', '<PAD>'] ['strange', 'twisted', '
     brilliant', '<PAD>', '<PAD>', '<PAD>'] ['
     strange', 'twisted', 'and', '<PAD>', '<PAD>',
     '<PAD>'] ['strange', 'twisted', 'macabre', '<
     PAD>', '<PAD>', '<PAD>']
```

**Example 2** Decision bias, as *Derrida* cannot be excluded ($\epsilon = 0.05$, FC network, 10 input words, 5 dimensional embedding, SST dataset):

```
Input: ['Whether', 'or', 'not', 'you', 'are', '
    enlightened', 'by', 'any', 'of', 'Derrida']

Exclude: ['Derrida']

Explanation: ['Whether', 'or', 'are', 'enlightened
    ', 'by', 'any', 'Derrida']
```