LANGUAGE PROCESSING 2, SESSION NO. 6

Exercise no. 1:
Part-of-Speech tagging using a Maximum Entropy tagger

T	**0**	**1**	**2**	**3**	**4**
WORDS	My	smartphone	worked	very	well
POS	PRP	NN	VP	MOD	RB

Suppose that we trained a Maximum Entropy tagger, which is a Multinomial Logistic Regression classifier (Logistic Regression with multiple output classes).

The features that we use are three very simple features:
- Is the first letter uppercase? (1 or 0)
- Length of the current word / observation
- Does the word end in "-ed"? (1 or 0)

Suppose that there are 10 possible POS tags:
- CONJ, DT, JJ, NN, NUM, PRP, RB, VA, VN, VP

Suppose the `x` vector is the one that will be multiplied by the weight matrix from the tagger itself. Let's try to guess how they look like.

The `x` vector represents, in this case, one single word at the sentence. We know the shape of the input vector `x`. It is (`1x3`), which means that we have 1 row and 3 columns. We have one row when we have one instance that we want to make predictions for. If we want to make predictions for more instances, the number of rows should correspond to the number of instances. We have three columns, because we have three features.

Simple exercise: Could you think which is the value of `x`, when `t=2`? `[-, -, -]`

The output that we want to produce (vector `y`) should contain 10 elements, as we want to decide which of the 10 possible POS tags is the most probable. Therefore, the output should have the shape 1x10.

We know that a Multinomial Logistic Regression makes predictions based on this formula:

$$y = \sigma(xW)$$

Which should the shape of W be?

```
-   Shape (x) = (1x3)
-   Shape (y) = (1x10)
```

Note: The softmax function (σ) does not change the shape of the vector.

SPOILER ALERT. Please don't go down until you have thought about the answers above.

Please find below an illustration of how the weights might look like in a Maxent POS tagger where we train a model with 3 features and 10 possible POS tags. The first matrix (x) is the input feature vector, which could be the feature vector for a word like “Mike”.

$$xW = \begin{pmatrix} 1 & 4 & 0 \end{pmatrix} \begin{pmatrix} 0.1 & 0.7 & 0.2 & 2.01 & 0.2 & 0.2 & 0.65 & 0.01 & 0.3 & 0.97 \\ 0.2 & 0.1 & 0.4 & 0.9 & 0.3 & 0.1 & 0.23 & 0.1 & 0.54 & 0.68 \\ 0.01 & 0.2 & 0 & 0.1 & 0.8 & 0.7 & 0.3 & 0.6 & 0.4 & 1.55 \end{pmatrix}$$
$$= \begin{pmatrix} 0.9 & 1.1 & 1.8 & 5.61 & 1.4 & 0.6 & 1.57 & 0.41 & 2.46 & 3.89 \end{pmatrix}$$

This is just an illustration, I made up the weight matrix values. Also, these values should later be normalized by using a softmax function[1], and the one that has the largest number is the index of the POS tag that we will return for this specific instance (the argmax function does that). We repeat this process for each word in the sentence. The POS tag list looks like this:

- CONJ, DT, JJ, NN, NUM, PRP, RB, VA, VN, VP

Which POS tag should we return in the specific example above?

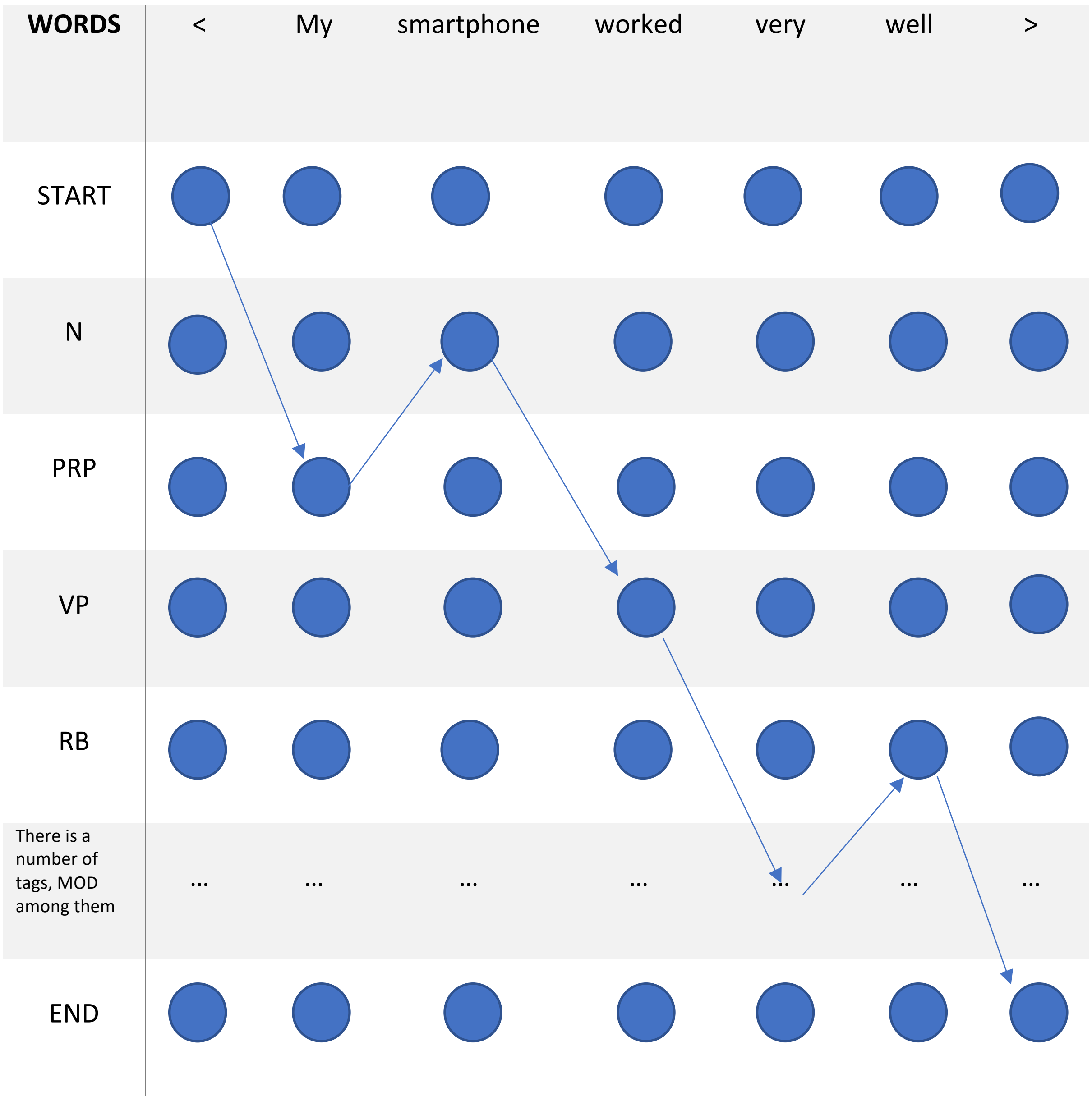
Exercise: Please write in a short sentence using your own words how the prediction of each POS tag is made.

[1] This function will convert the vector into a probability distribution.

Exercise no. 2:
The task that we will perform in this exercise is exactly the same. We want to POS-tag a given sentence.

T	**0**	**1**	**2**	**3**	**4**
WORDS	My	smartphone	worked	very	well
POS	PRP	NN	VP	MOD	RB

The difference is that in the previous example the predictions were made independently. In this case, we will use a Maximum Entropy Markov Model, which considers the previous POS tags / outputs when making predictions, like Hidden Markov Models. We can visualize these dependencies by using what it is called a trellis data structure. Please look below:



The arrows in the figure above, represent the actual POS tags that our model should return. We calculate those arrows using the Viterbi algorithm, which is a dynamic programming algorithm (like the Minimum Edit Distance algorithm).

Let us zoom in a specific node, for instance the one in row n=2 and column n=3, (starting from 1) which we will refer to $V_{N,smartphone}$. The set of all outgoing arcs from that specific node will represent a probability distribution, meaning that all probabilities coming out of that node will sum to one. These arcs are shown in the next page in the same example. This distribution is defined in the following way:

$P_{S'}(S|O) = [P1, P2, \ldots, P12]$

Where S' is the origin state, S is the new state and O is the observation. As we model probabilities in this way, the observation can be represented as a number of feature functions. In our case, to make it simple, we will have three feature functions to represent our observation, and the functions will be exactly the same functions that we used in the previous exercise: [is_uppercase, length_word, ends_in_ed]. Usually, people use more complex features than these, involving current and surrounding words and tags, but this works as an illustration.

$P_1 = P_N(\text{START} \mid w_{t=smartphone}) = P_N(\text{START} \mid [0,10,0])$
$P_2 = P_N(\text{N} \mid w_{t=smartphone}) = P_N(\text{N} \mid [0,10,0])$
$P_3 = P_N(\text{PRP} \mid w_{t=smartphone}) = P_N(\text{PRP} \mid [0,10,0])$
$P_4 = P_N(\text{VP} \mid w_{t=smartphone}) = P_N(\text{VP} \mid [0,10,0])$
…
Until P_{12}, as we have 10 possible states (POS tags) + one start and one end states.

Some of these probabilities might be zero, and if that is the case in, e.g. 5 nodes, we could say that the node has 7 outgoing arcs (or non-zero probabilities).

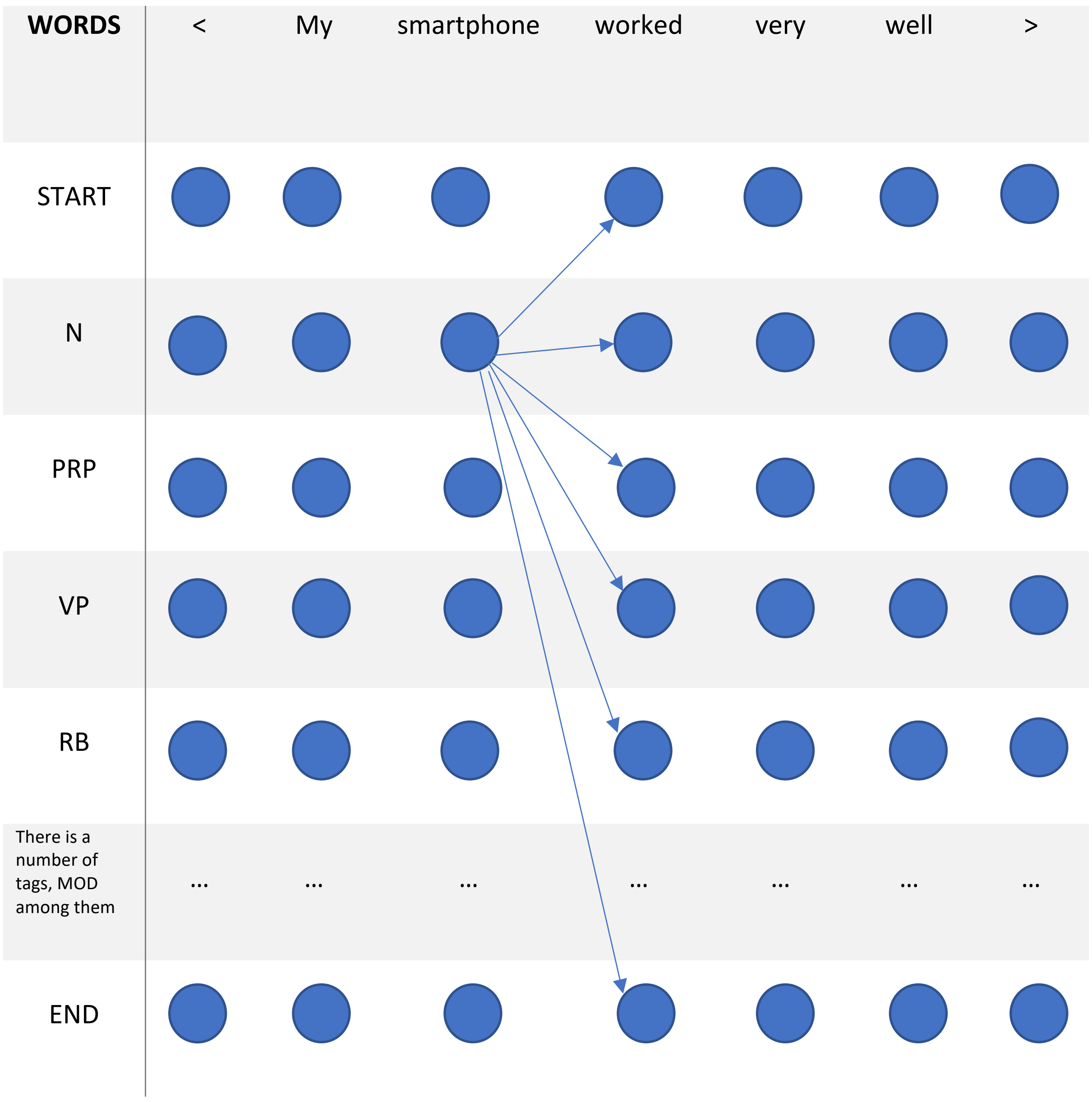
Now, I would like you to analyze the Viterbi algorithm, as shown in Jurafsky and Martin (2008, p. 220). We also include a snapshot of that algorithm below[2]. Please try to find out where we should fit the type of probabilities that we described above. We are not asking you to fit the specific P_N(START | …) above, but the general formula $P_{S'}(S|O)$. Where does it fit in the algorithm?

[2] https://web.stanford.edu/~jurafsky/slp3/A.pdf

```
function VITERBI(observations of len T,state-graph of len N) returns best-path, path-prob

create a path probability matrix viterbi[N,T]
for each state s from 1 to N do                        ; initialization step
      $viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
      $backpointer[s,1] \leftarrow 0$
for each time step t from 2 to T do                    ; recursion step
   for each state s from 1 to N do
      $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
      $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$                 ; termination step
$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T]$             ; termination step
$bestpath \leftarrow$ the path starting at state bestpathpointer, that follows backpointer[] to states back in time
return bestpath, bestpathprob
```

Figure A.9 Viterbi algorithm for finding optimal sequence of hidden states. Given an observation sequence and an HMM $\lambda = (A, B)$, the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence.



Exercise no. 3:

What happens if we have three nodes, the first one with 12 outgoing arcs, a second one with 7 outgoing arcs and a last one with only 2 outgoing arcs? How do those probabilities look like? May that affect to predictions that the model will make?

Exercise no. 4:
Yes! You identified the problem! This problem is called the Label Bias problem, and as you may have guessed, it refers to the fact that the nodes with less outgoing arcs will have higher probabilities to happen than others with more.

Now, you should think more. How would you solve this problem?