Homework 2

Ana Sofia Carmo, 83810

Deep Structured Learning, 2021/22

Question 1

1.(a)

Given the formulation of John's activities/weather as a Hidden Markov Model (HMM), the *most likely sequence* of weather states can be inferred from the observed activities through the Viterbi algorithm. This algorithm approaches the likelihood problem by assuming that each state i depends only on the corresponding observed activity i (given by the emission probabilities) and on the adjacent weather states (given by the transition probabilities).

Given the implementation of the Viterbi algorithm in `hw2-q1.py`, the most likely weather for the past week was: Rainy → Sunny → Sunny → Sunny → Sunny → Sunny → Sunny. Table 1 shows the results from the forward pass of the algorithm, which was then backtracked to achieve the most likely weather sequence.

	7	8	9	10	11	12	13	14	15
	–	Videogame	Surf	Beach	Study	Beach	Study	Beach	–
Sunny	0	2.00e-02	1.20e-02	2.88e-03	1.73e-04	4.15e-05	2.49e-06	5.97e-07	1
Windy	0	3.00e-02	2.25e-02	1.12e-03	2.59e-04	1.30e-05	3.73e-06	1.87e-07	0
Rainy	1	1.50e-01	7.50e-03	4.50e-04	1.44e-04	7.20e-06	2.07e-06	1.04e-07	0
Pred.	–	Rainy	Sunny	Sunny	Sunny	Sunny	Sunny	Sunny	–

Table 1: Results from the forward pass of the Viterbi algorithm, along with the most likely sequence of weather obtained through backtracking. Note: although the computations were performed in the log-domain, the results are shown in the non-log-domain. These results were compared with fellow student Afonso Raposo.

1.(b)

Given that we want to maximize the number of days in which we make a correct prediction (rather than aiming at getting the whole sequence right), we should use minimum

risk decoding to make our weather predictions. This decoding approach makes a prediction about state i by maximizing the corresponding posterior probability.

Given the implementation of this algorithm in `hw2-q1.py`, the most likely weather for the past week was: Rainy → Windy → Sunny → Windy → Sunny → Windy → Sunny

2.

HMMs assume that John's activities are conditionally independent, therefore there is no way to accommodate this dependency between adjacent activities. However, Conditional Random Fields (CRFs) provide sort of a generalized form of HMMs that allows to include this extra piece of knowledge, by not assuming independence between activities (as illustrated by Figure 1).

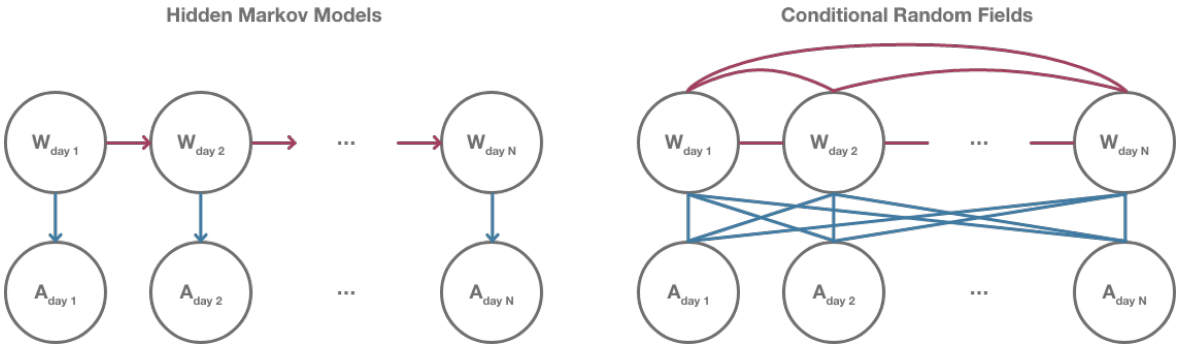


Figure 1: Illustration of HMMs and CRFs emphasizing the fact that HMMs are directed graphs, contrarily to CRFs, and the additional dependency possibilities that CRFs provide. $W_{day\ i}$: *weather of day* i; $A_{day\ i}$: *activity of day* i.

Question 2

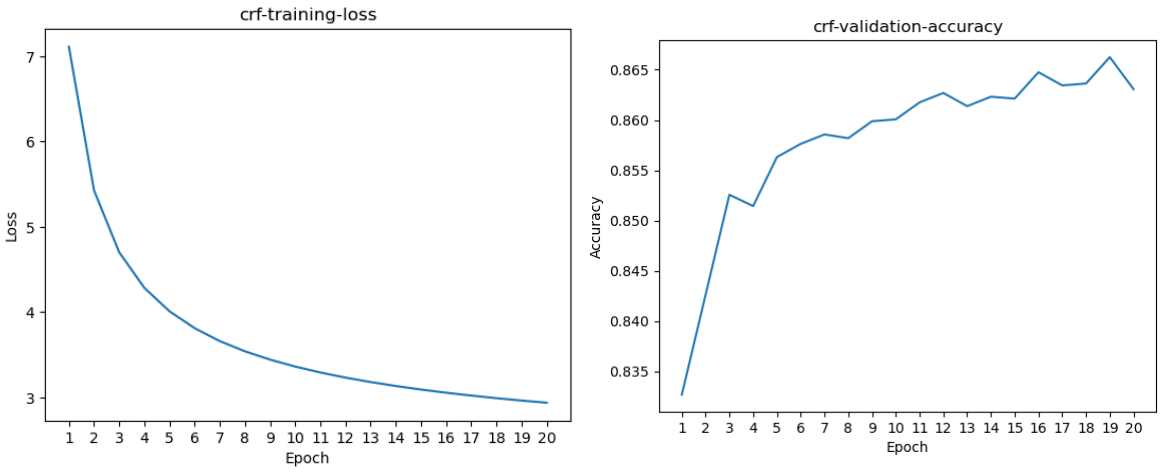
Linear CRFs assume that the dependency between the labels only exists between adjacent elements, greatly simplifying the generalized approach of standard CRFs. This dependency if characterized by a chosen feature function (in this case, $\psi_{i,i-1}(y_i, y_{i-1}) := exp(w \cdot \phi_{i,i-1}(x)))$ and is modeled within the CRF by W_{big}. On the other hand, the relation between a label and the corresponding pixel representation of the character is modeled by W_{unig}. As such, the CRF models the conditional probability of a sequence of labels given a sequence of pixel-representation of characters by:

$$P_W(labels|pixel\ characters) = \frac{1}{Z(W,x)} exp(\sum_i w_{unig(y_i)} \cdot \phi_i(x) + \sum_i w_{big(y_i,y_{i-1})} \cdot \phi_{i,i-1}(x))$$

where $Z(W, x)$ is a normalization factor.

Within training of this model, we estimate the values of the weights W_{big} and W_{unig} by minimizing the negative log-likelihood ($-P_W(labels|pixel\ characters)$), as implemented in `hw2_linear_crf.py`[1]. In order to infer the sequence of labels from a given sequence of pixel representation of characters, we can use the Viterbi algorithm, already explained in the previous question (also implemented in `hw2_linear_crf.py`).

For the sake of simplicity, training of performed using epochs of a single sample, avoiding the need to include masks. On the test set, the model achieved an accuracy of 0.8573. Figures 2(A) and 2(B) provide the loss over training epochs and the corresponding accuracy on the validation set, respectively.



(A) (B)

Figure 2: Results from the implementation of the CRF model. (A) Loss over training epochs and (B) the corresponding accuracy on the validation set.

Question 3

1.

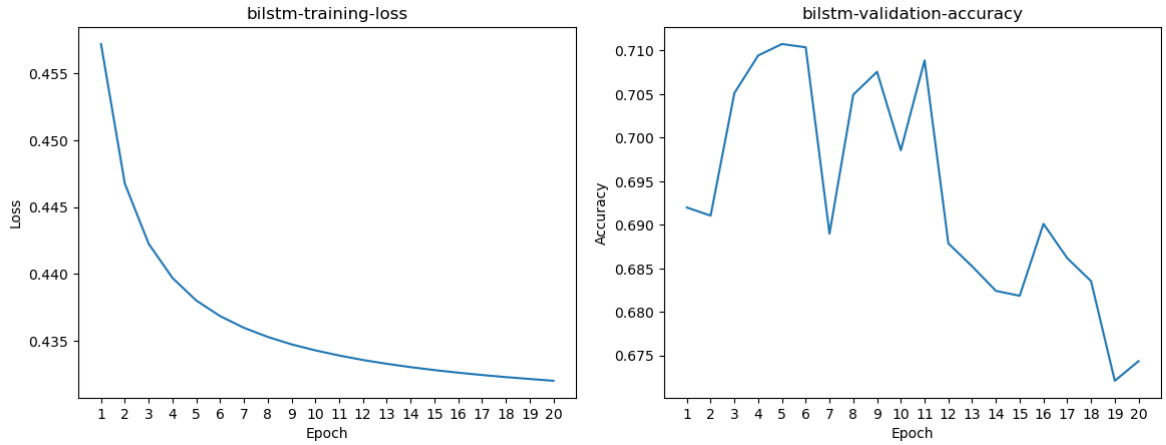
The Bidirectional Long-Short Term Memory (BILSTM) layer consists of two LSTMs: one receiving the input in the forward direction, and the other in the backward direction. A linear layer is then needed to map the output of the BILSTM into a tensor that has the length corresponding to the number of classes.

On the test set, the model achieved an accuracy of 0.6733. Figures 3(A) and 3(B) provide the loss over training epochs and the corresponding accuracy on the validation set,

[1]The Python implementation of the required segments of code were inspired in the post by Marcos Treviso on Towards Data Science, "Implementing a linear-chain Conditional Random Field (CRF) in PyTorch".

respectively. As seen from Figure (B), however, one can hypothesize that the model has overfit the training data, having decreased testing performance after some epoch.

As a solution, using a third (validation) set of data, regularization could be optimized to avoid this phenomenon (either through the dropout rate or the l2-regularization).



(A) (B)

Figure 3: Results from the implementation of the BILSTM model. (A) Loss over training epochs and (B) the corresponding accuracy on the validation set.

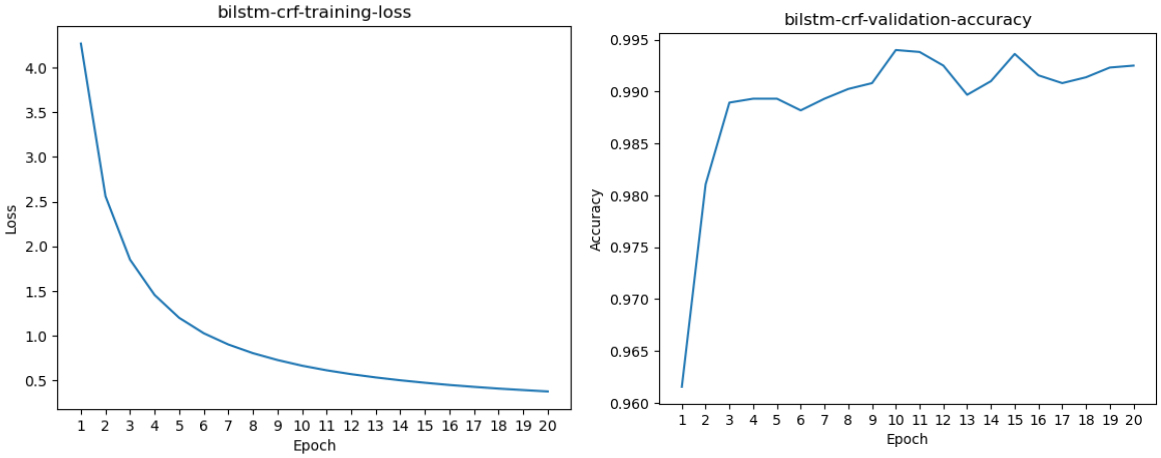
2.

While linear CRFs explicitly model the dependencies between adjacent labels (through transition scores), they can only do so in the forward direction. Conversely, the BILSTM architecture yields the context of a label in both the forward and backward directions by combining the output of two LSTM networks (one receiving the input in the forward direction, and the other in the backward direction).

As such, BILSTMs get a more complete insight into the context of a particular label than CRFs. However, a BILSTM does not explicitly model the dependencies between labels, loosing the insight that CRFs have into the internal logic of labeling.

3.

The proposed architecture consists of using a BILSTM layer to convey the bidirectional context of each label into a feature tensor that is then used as the input to a CRF, thus combining the advantages of both architectures. Figures 4(A) and 4(B) provide the loss over training epochs and the corresponding accuracy on the validation set, respectively, and Table 2 provides the performances of both approaches.



(A) (B)

Figure 4: Results from the implementation of the BILSTM-CRF model. (A) Loss over training epochs and (B) the corresponding accuracy on the validation set.

Table 2: Results obtained for the BILSTM and BILSTM-CRF models, namely accuracy on the test set and training time.

	Final Accuracy	**Training Time**
CRF	0.8573	823.29 s
BILSTM	0.6733	256.43 s
BILSTM-CRF	0.9901	1105.26 s

The improvement in performance by incorporating the CRF on top of the Long-Short Term Memory (LSTM) is apparent when compared to both CRF and BILSTM alone, illustrating the phenomenon explained (however at the cost of a much longer training time).