IMPROVING MULTI-STREAM CLASSIFICATION BY MAPPING SEQUENCE-EMBEDDING IN A HIGH DIMENSIONAL SPACE

Mohamed Bouaziz^{1,2}*, Mohamed Morchid*^{1}*, Richard Dufour*^{1}*, Georges Linarès*^{1}

^{1}LIA - University of Avignon (France)
^{2}EDD - Paris (France)
mohamed.bouaziz@alumni.univ-avignon.fr,
{firstname.lastname}@univ-avignon.fr

ABSTRACT

Most of the Natural and Spoken Language Processing tasks now employ Neural Networks (NN), allowing them to reach impressive performances. Embedding features allow the NLP systems to represent input vectors in a latent space and to improve the observed performances. In this context, Recurrent Neural Network (RNN) based architectures such as Long Short-Term Memory (LSTM) are well known for their capacity to encode sequential data into a non-sequential hidden vector representation, called sequence embedding. In this paper, we propose an LSTM-based multi-stream sequence embedding in order to encode parallel sequences by a single non-sequential latent representation vector. We then propose to map this embedding representation in a high-dimensional space using a Support Vector Machine (SVM) in order to classify the multi-stream sequences by finding out an optimal hyperplane. Multi-stream sequence embedding allowed the SVM classifier to more efficiently profit from information carried by both parallel streams and longer sequences. The system achieved the best performance, in a multi-stream sequence classification task, with a gain of 9 points in error rate compared to an SVM trained on the original input sequences.

***Index Terms*—** Sequence embedding, multi-stream classification, high dimensional space

1. INTRODUCTION

In the recent years, Neural Networks (NN) became state-of-the-art Machine Learning (ML) techniques in several application fields with impressive performances [1, 2, 3, 4]. The classification of event sequences, such as words, sentences and time series, is one of the applications that witnessed the considerable performance of NN. RNN based architectures such as Long Short-Term Memory (LSTM) [5] have gained a particular attention in sequence classification including sentence [6] and successive images [7] processing. An important advantage of RNNs is that they encode sequential data into a non-sequential hidden vector representation. This process is also known as *Sequence Embedding* [8], and more commonly as *Phrase Embedding* in the Natural Language Processing (NLP) domain [9, 10, 11]. In fact, in tasks such as computer vision and audio-visual speech processing, several event sequences are provided by parallel and maybe independent sources. However, straightforward RNN representation methods can not encode this multi-stream data. Therefore, we propose in this work to encode parallel sequences by concatenating multiple sequence embeddings in order to obtain a single non-sequential super vector representation. This latter is then used to help classifiers to behave more effectively with multi-stream sequential data. A classical Multi-Layer Perceptron (MLP) [12, 13, 14] tries to find out piecewise decision surfaces in order to separate not linearly separable input features. In fact, the ability of MLP to classify these kinds of data is limited by the number of hidden units. However, support vector machines (SVM) [15] can map the input features in a high-dimensional (and even infinite) space to find a robust hyperplane that linearly separates the data. The classification of sequence-embeddings is evaluated using these two classifiers through a telecast genre prediction task. We noticed that the error observed during the classification is lower when a hyperplane is found on the high-dimensional space than when nonlinear decision surfaces are found from the original space with MLP.

The rest of the paper is organized as follows. We present some related works in Section 2. A workflow of our approach is then shown after some basic concepts in Section 3. Finally, the conducted experiments and the results are respectively described in Sections 4 and 5 before concluding in Section 6.

2. RELATED WORKS

In this section we present some related works on sequence classification and sequence embeddings.

This work was funded by the EDD company and by the GaFes project supported by the French National Research Agency (ANR) under contract ANR-14-CE24-0022.

2.1. MLP and SVM classification

SVM and MLP are two machine ML methods that proved their efficiency in several classification tasks [12, 13, 14, 15]. A Multi-Layer Perceptron (MLP) uses a defined number of perceptrons [16] which carries out a binary classification by separating regions that are linearly separable. Using multiple hidden units (perceptrons), MLP can build piecewise decision surfaces in order to separate not linearly separable input features. However, the ability of MLP to classify these kinds of data is limited by the number of hidden units. In the other hand, SVM has the ability to map the input features in a space of infinitely high dimensions to find a robust hyperplane that linearly separates the data.

Regarding sequence classification, SVMs are well-known for their competitive performance [17, 18]. Many works dealt with sequence classification using SVM or SVM-based classifiers. In [19], SVM outperformed Naive Bayes, k-nearest neighbors, and decision tree algorithms in formal text classification. SVM performance in [20] was the best among a set of several classifiers, including decision tree and random forest algorithms, for adaptive sentiment classification on dynamic tweets.

2.2. Neural networks and Sequence embeddings

Convolutional NN [1, 21, 22] as well as Recursive NN proved their efficiency in the field of sequence classification. Due to their architecture, Recurrent NN (RNN) are even more suitable for sequential inputs. For example, Long Short-Term Memory networks are among the widely used RNN [23, 3]. [24] uses a tree-structured LSTM outperforming several sequential models and [25] proposes a hierarchical LSTM with two levels of LSTM networks that builds an embedded representation of words sequence for tweet representation. This last work guides us to broach the subject of sequence embedding that gives a real-valued vector representation to a sequence of events (words for NLP tasks). Authors in [25] use the hidden state of the last element (of the word-level LSTM) as the representation of the sequence (*i.e.* the tweet). [8] build a supervised sequence embedding by projecting n-grams into a latent lower-dimensional space through a sliding window of length n. Semi-supervised approaches, as in [26] and [10], initiate a recursive auto-encoder (RAE) with an unsupervised algorithm before fine-tuning it using the output's label. CNN are also used for sequence embedding [27] in order to generate phrase embeddings from word embeddings which learns a low dimensional real-valued vector representation of words. [28] simply defines a sentence representation with the sum of all single-word embeddings. All these approaches propose an embedding type that is oriented towards a specific task, such as sentiment classification [9], machine translation [29] and, more recently, paraphrase identification [28] and may not necessarily take the word order as the privileged information. For this reason, we chose an LSTM-based embedding in our work since it would be the most adapted architecture to encode sequences.

3. PROPOSED APPROACH

Section 3.1 describes LSTM networks that we used in order to extract sequence embeddings. The two used classifiers (MLP and SVM) are then detailed respectively in Sections 3.2 and 3.3. We finally detail our multi-stream classification process in Section 3.4.

3.1. Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) [5] networks are a special case of Recurrent Neural Networks (RNNs) [30]. The goal of this architecture is to create an internal cell state of the network which allows it to exhibit dynamic temporal behavior. This internal state allows the RNN to process arbitrary sequences of inputs such as sequences of words [6] for language modeling, time series [31]... The RNN takes as input sequence $\mathbf{x} = (x_1, x_2, \ldots, x_T)$ and computes the hidden sequence $\mathbf{h} = (h_1, h_2, \ldots, h_T)$ as well as the output vector $\mathbf{y} = (y_1, y_2, \ldots, y_T)$ by iterating from $t = 1$ to T:

$$h_t = \mathcal{H}(\mathbf{W}_{xh} x_t + \mathbf{W}_{hh} h_{t-1} + b_h) \tag{1}$$
$$y_t = \mathbf{W}_{hy} h_t + b_y \tag{2}$$

where T is the total number of sequences; $\mathbf{W}_{xh}$ are the weight matrices between the input layers $\mathbf{x}$ and $\mathbf{h}$ and so on; b is a bias vector, and $\mathcal{H}$ is the composite function. [5] shows that LSTM networks outperform RNNs for finding long range context and dependencies. The LSTM composite function $\mathcal{H}$ forming the LSTM cell with peephole connections [32] is presented in Figure 1 and defined as:

$$i_t = \sigma(\mathbf{W}_{xi} x_t + \mathbf{W}_{hi} h_{t-1} + \mathbf{W}_{ci} c_{t-1} + b_i) \tag{3}$$
$$f_t = \sigma(\mathbf{W}_{xf} x_t + \mathbf{W}_{hf} h_{t-1} + \mathbf{W}_{cf} c_{t-1} + b_f) \tag{4}$$
$$c_t = f_t c_{t-1} + i_t \tanh(\mathbf{W}_{xc} x_t + \mathbf{W}_{hc} h_{t-1} + b_c) \tag{5}$$
$$o_t = \sigma(\mathbf{W}_{xo} x_t + \mathbf{W}_{ho} h_{t-1} + \mathbf{W}_{co} c_t + b_o) \tag{6}$$
$$h_t = o_t \tanh(c_t) \tag{7}$$

where i, f and o, are respectively the input, forget and output gates, and c the cell activation vector with the same size than the hidden vector h. The weight matrices $\mathbf{W}$ from cell c to gates i, f and o, are diagonal, and thus, an element e in each gate vector receives only the element e from the cell vector. Finally, σ is the logistic sigmoid function. The h_t layer composes the sequence embedding that is employed as input features of both MLP and SVM classification approaches.

3.2. Multi-Layer Perceptron

A feed-forward neural network with a single hidden layer, called a Multi-Layer Perceptron (MLP), is composed of three

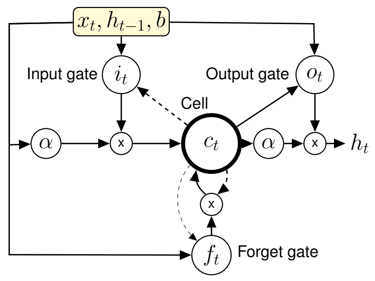

Fig. 1. Long Short-Term Memory (LSTM) cell. Dashed arrows correspond to connections with time-lag $(t-1)$. α input/output activation function is usually $\tanh$.

different components (or layers) as presented in Figure 2: input layer (x), hidden layer(s) (θ) and output layer (y). The hidden layer in the MLP containing is fully connected to input and output ones. The activation function used during the

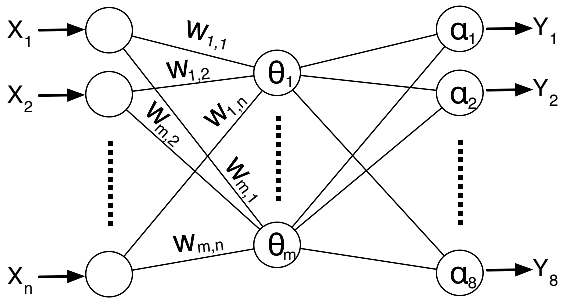

Fig. 2. A Multi-Layer Perceptron (MLP) architecture.

experiments is the classical *hyperbolic-tangent* function:

$$\alpha(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{8}$$

The feed-forward algorithm of the MLP is composed of 3 steps: forward, learning and update phases:

Forward phase
Let N_l be the number of neurons contained into the layer l ($1 \leq l \leq M$) and M the number of layers of the MLP. b_n^l is the bias of the neuron n ($1 \leq n \leq N_l$) from the layer l. Given a set of P input patterns x_p ($1 \leq p \leq P$) and a set of labels y_p associated to each x_p, the output γ_n^l ($\gamma_n^0 = x_p^n$) of the neuron n from the layer l is given by:

$$\gamma_n^l = \alpha(S_n^l)$$
$$\text{with } S_n^l = \sum_{m=0}^{N_{l-1}} w_{nm}^l \times \gamma_m^{l-1} + b_n^l \tag{9}$$

Learning phase
The error e observed between the expected outcome y and the result of the forward phase γ is then evaluated as follows:

$$e_n^l = y_n - \gamma_n^l \tag{10}$$

for the output layer ($l = M$), and

$$e_n^l = \sum_{h=1}^{N_{l+1}} w_{h,n}^{l+1} \times \delta_h^{l+1} \,, \tag{11}$$

for the hidden layer l ($1 < l < M$). The gradient δ is computed with:

$$\delta_n^l = e_n^l \times \frac{\partial\alpha(S_n^l)}{\partial S_n^l} \text{ where } \frac{\partial\alpha(S_n^l)}{\partial S_n^l} = \alpha(S_n^l)(1-\alpha(S_n^l)) \tag{12}$$

Update phase
When errors between the expected outcome and the result are computed, the weights $w_{n,m}^l$ and the bias b_n^l have to be respectively updated to $w_{n,m}^{l\star}$ and $b_n^{l\star}$:

$$w_{n,m}^{l\star} = w_{n,m}^l + \epsilon\delta_n^l \times \alpha(S_n^l) \tag{13}$$
$$b_n^{l\star} = b_n^l + \epsilon\delta_n^l \,. \tag{14}$$

3.3. High-dimensional hyperplane from a SVM

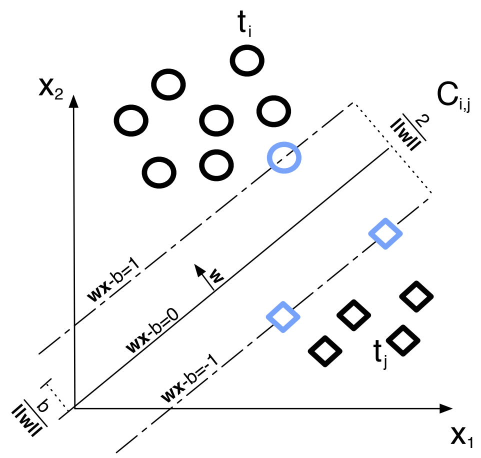

Fig. 3. Maximum-margin hyperplane and margins for a SVM trained with samples from two classes. Samples on the margin are called the support vectors.

This classifier modifies the representation of data, which are mapped into a space of higher dimension. More formally, a support vector machine (SVM) constructs a hyperplane or a set of hyperplanes in a high (or infinite) dimensional space, which can be used for classification (see [33] for further details). In fact, if data is not linearly separable, the separation between input features vectors are impossible. Therefore, the hyperplanes (or set of hyperplanes) have to be found in a higher even infinite space. Intuitively, a good separation is achieved by the hyperplane that has the largest distance $\frac{2}{||\mathbf{w}||}$ to the nearest training data point $\mathbf{w}$ of any class (so-called functional margin), since, in general, the larger the margin the lower the generalization error of the classifier (see Figure 3). This hyperplane is defined as the set of points x_i labeled as y_i that minimize the norm $||w||$ of the normal vector as described in Figure 3 subject to:

$$w \times x_i + b \geq y_i \tag{15}$$

3.4. Multi-stream classification using sequence-embedding

LSTM is known to perform quite well in sequential inputs classification. However, this representation method can not encode multi-stream data provided by independent sources. In fact, simply concatenating the parallel input data is not relevant. LSTM is actually supposed to encode recurrent relations to rather homogeneous input data. Thus, giving concatenated data to an LSTM layer is theoretically incorrect.

Therefore, our approach, presented in Figure 4, consists in training a different LSTM-based network on each input stream as in ***a)***. Building this supervised sequence embedding would allow us to make a latent representation of the sequential data, in the form of non-sequential vectors. Afterwards, we concatenate, as in ***b)***, the vectors produced by LSTM encoding in a single super-vector performing a multi-stream latent representation. We think this representation could indeed help state-of-the-art classifiers to behave more effectively with multi-stream sequential data.

Regarding the classification process, we compared two state-of-the-art methods, namely MLP and SVM trained on the produced sequence-embedding representation (as in ***c)***). SVMs find a hyperplane on the high-dimensional space while MLPs define nonlinear decision surfaces in the original space.

4. EXPERIMENTAL PROTOCOL

We will evaluate our multi-stream sequence-embedding representation through a telecast genre prediction task. In this section, we describe the evaluation task, the used dataset and the models settings.

4.1. Genre prediction task

Sequence classification is evaluated through an automatic TV show genre labeling task on the Electronic Program Guide (EPG) corpus of the University of Avignon. For a given input history sequence (composed of the n previous telecast genres), a genre label representing the next telecast is output. The size of the genre sequences (n) varies from 1 to 19. Multi-stream experiments consists in using the history of multiple channels in order to predict the next telecast genre for a specific channel.

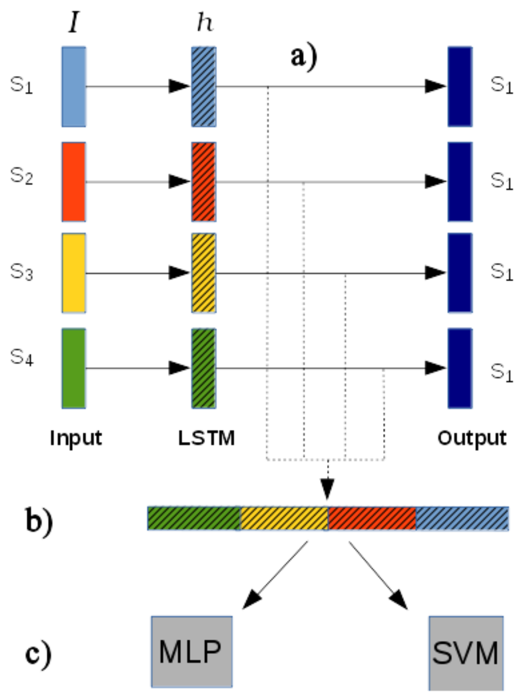

Fig. 4. Multi-stream classification process using multi-source LSTM-based latent representation. S_n: the n^{th} stream.

4.2. EPG Dataset

The EPG dataset is extracted from 4 French TV channels (M6, TF1, France 5 and TV5 Monde) for 3 years, from January 2013 to December 2015. M6 channel is used in our experiments as the output stream. Data from *2013* and *2014* are merged and split into the *training* (70%) and *validation* (30%) datasets using a *stratified shuffle split* [34] in order to preserve the same percentage of samples of each class in the output of both folds, while the *2015* dataset is kept for testing. In order to guarantee a clean experimental environment, labels (*i.e.* genres) that are absent at least in one of the three folds were removed. Doing so allows us to have equivalent datasets in terms of labels vocabulary.

4.3. Models setups

The LSTM sequence embedding is built using, for each channel, an input layer with a size varying from 1 to 19 according to the sequence length, the LSTM hidden layer $\mathbf{h}$ of size 80 and an output layer with a size equals to the number of different possible TV genres (11). Regarding the classifiers based on sequence embedding, the MLP network is composed of a single hidden layer containing 400 nodes. The Keras library [35], based on Theano [36] for fast tensor manipulation and CUDA-based GPU acceleration, has been employed to train neural networks on a Nvidia GeForce GTX TITAN X GPU card. For our second classifier, the SVM one-against-one method is chosen with an RBF kernel [37]. This method gives a better accuracy than the one-against-rest [38]. In this multi-class problem, T denotes the number of genres and $t_i, i = 1...T$ denotes the T genres. A binary classifier is then used for every pair of distinct genres. As a result, $T(T-1)/2$ binary classifiers are constructed all together. The binary classifier $C_{i,j}$ is trained from example data where t_i is a positive class ($w \times x + b = 1$) and t_j a negative ($w \times x + b = -1$) one ($i \neq j$). For a vector representation of an unseen input s, if $C_{i,j}$ means that s is in the label t_i, then the vote for the class t_i is added by one. Otherwise, the vote for the genre t_j is increased by one. After the vote of all classifiers, the input s is assigned to the label having the highest number of votes.

5. RESULTS AND DISCUSSION

After detailing the experimental protocol, we expose the results along with the relative discussions in two parts. We present experiments using raw input data followed by those using LSTM-based sequence embedding. The obtained results are compared with baseline mono-stream and multi-stream experiments detailed in this section.

5.1. Classification using raw data

As a baseline mono-stream experiment, we carry out genre prediction using the raw input sequences. A first mono-channel experiment is conducted using M6 history sequences with an SVM (**SVM-I**$_{M6}$) and an MLP (**MLP-I**$_{M6}$) classifier. As depicted in Figure 5, **SVM-I**$_{M6}$ performance improves using bigger sequence size until reaching the best error rate (ER) among the two classifiers (30%) with sequences containing genres of the last 6 telecasts. However, when sequence size is too big (greater than 6), SVM classifier in **SVM-I**$_{M6}$ fails more and more at predicting the next genre as long as the input size increases. We can explain this behavior by the fact that SVM is known to have some weaknesses when dealing with relatively high *input size* by *training examples* ratio.

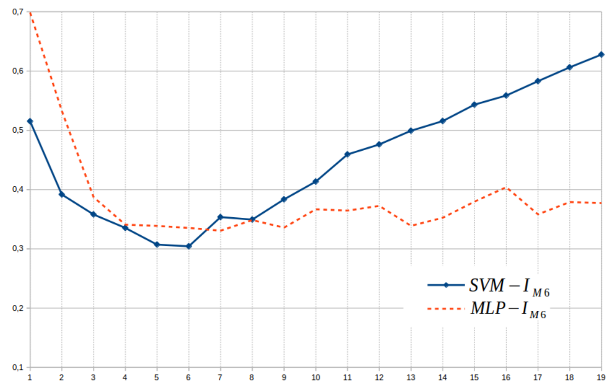

Fig. 5. Performances (ER) of Mono-stream experiments : SVM vs Multi-Layer Perceptron (MLP)

Considering that we evaluate our multi-stream classification approach in the context of telecast genre labeling, we want to verify if the history of other channels is useful to predict the genre of a single channel. For this purpose, we use separately each of the 4 channels history as input sequences (**SVM-I**$_{M6}$, **SVM-I**$_{TF1}$, **SVM-I**$_{Fr5}$ and **SVM-I**$_{TV5}$). Figure 6 shows that it is possible to predict the next M6 telecast genre using other streams. In fact, with TF1 channel input, the system correctly classifies almost half of the test examples.

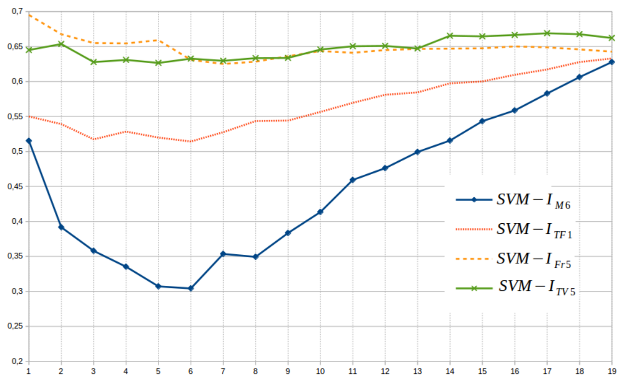

Fig. 6. Performance (ER) of Mono-stream SVM on each channel

The next step is to carry out a baseline multi-stream experiment using the concatenation of all the raw input sequences still using SVM (**SVM-I**$_{Concat}$) and MLP (**MLP-I**$_{Concat}$) classifiers. As show in Figure 7, **SVM-I**$_{Concat}$ outperforms both the SVM mono-stream classifier (**SVM-I**$_{M6}$) and **MLP-I**$_{Concat}$ with 26% of ER using sequences of size 2. Nonetheless the performance more rapidly and drastically falls with longer sequences. We can notice that even if **MLP-I**$_{Concat}$ performs better with big sequence sizes, its performance also decreases as long as the sequence size increases. It becomes obvious that combining the different streams in this way (using raw data), is not the best solution in order to use the information provided by parallel streams.

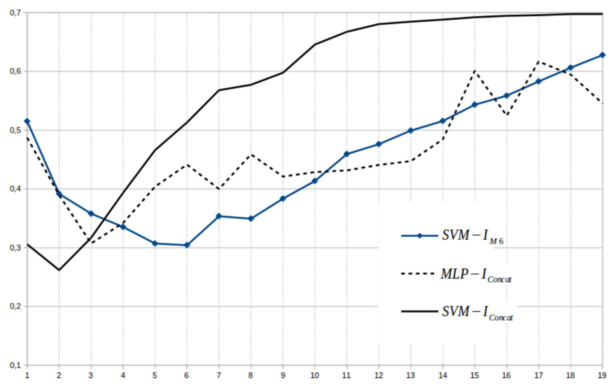

Fig. 7. Performances (ER) of Multi-stream experiments: SVM vs MLP

5.2. Classification using sequence-embedding

We noticed in the previous section that multi-stream classification using raw data behave badly when dealing with relatively long sequences. In order to tackle this problem, we train, as previously shown in Section 3.4, a separate LSTM-based latent representation for each stream as sequence embedding for our classifiers (cf. part ***a)*** in Figure 4). At first, we visualize the individual impact of the LSTM encoding through the mono-stream LSTM experiment carried out on M6 (**LSTM-I**$_{M6}$). At this point, as depicted in Figure 8, this model already outperforms **SVM-I**$_{M6}$ in almost all input configurations with an ER of about 24% at several points.

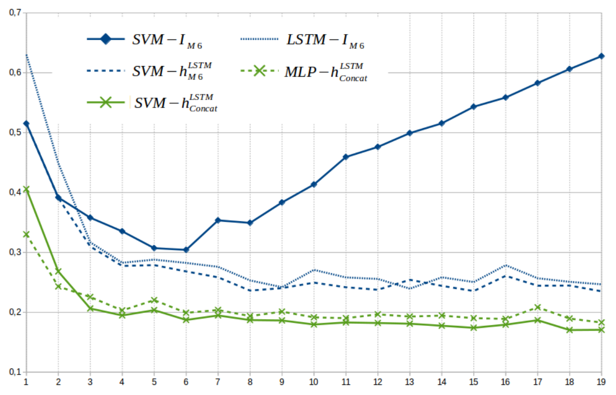

Fig. 8. Performances (ER) of Mono-stream and multi-stream experiments with LSTM output: SVM vs MLP

Continuing at the mono-stream level, an SVM trained over the output of the mono-stream LSTM layer (**SVM-h**$_{M6}^{LSTM}$) is even slightly better. It is worth emphasizing that these two experiments (cf. Figure 8 and Table 1) suffer no more with relatively big history sequence sizes. This proves at this level that the LSTM-based latent representation is more robust and helps a classifier to better behave with sequential data.

Back to our multi-stream experiment workflow, it is time now to see how the multi-stream sequence embedding could help at dealing with parallel sequential input data. Results shown in Figure 8 reveal that classifiers based on the concatenated latent representations (cf. parts ***b)*** and ***c)*** in Figure 4), which consists in MLP (**MLP-h**$_{Concat}^{LSTM}$) and SVM (**SVM-h**$_{Concat}^{LSTM}$) classifiers, outperform the mono-stream LSTM-based experiments (**LSTM-I**$_{M6}$ and **SVM-h**$_{M6}^{LSTM}$) by between 5 and 10 points. Furthermore, relying on the results of multi-stream experiments shown in Table 2, it is proven that sequence-embedding multi-stream classifiers (**MLP-h**$_{Concat}^{LSTM}$ and **SVM-h**$_{Concat}^{LSTM}$) do a lot better than simply concatenating raw data sequences in **MLP-I**$_{Concat}$ and **SVM-I**$_{Concat}$. Not only are the error rates lower in almost all the history sizes, but also these models overtake the problem faced with long sequences of history input. It is now rather possible to fully profit from the additional information carried by long sequences with the best performances using 19 and 18 sized history sequences for respectively **MLP-h**$_{Concat}^{LSTM}$ and **SVM-h**$_{Concat}^{LSTM}$. In this configuration, **MLP-h**$_{Concat}^{LSTM}$ reaches an ER of 18% against 17% for **SVM-h**$_{Concat}^{LSTM}$.

Architecture	ER(%)	Sequence size
SVM-I$_{Fr5}$	62.51	7
SVM-I$_{TV5}$	62.66	5
SVM-I$_{TF1}$	51.43	6
SVM-I$_{M6}$	30.43	6
MLP-I$_{M6}$	33.06	7
LSTM-I$_{M6}$	23.95	13
SVM-h$_{M6}^{LSTM}$	**23.51**	19

Table 1. Best performance for each mono-stream experiment

Architecture	ER(%)	Sequence size
MLP-I$_{Concat}$	30.74	3
SVM-I$_{Concat}$	26.19	2
MLP-h$_{Concat}^{LSTM}$	18.30	19
SVM-h$_{Concat}^{LSTM}$	**17.04**	18

Table 2. Best performance for each multi-stream experiment

6. CONCLUSION

In this paper, we presented a solution for classifying parallel sequences using an LSTM-based multi-stream sequence embedding. This representation helps classifiers not only to predict more precisely the correct output of sequences but also to profit more efficiently from information carried by both parallel streams and longer sequences. In this context, SVM performed better than the used MLP architecture. From our knowledge, this work is the first that tries to build an embedding from input sequences provided by multiple streams. In future works, we will apply our proposed method in other tasks such as sentiment analysis. We intend also to extend the architecture in order to treat additional NLP applications especially sequence labeling tasks such as Part-Of-Speech tagging and Named Entities labeling.

7. REFERENCES

[1] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom, "A convolutional neural network for modelling sentences," *ACL*, 2014.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, "Efficient estimation of word representations in vector space," *ICLR*, 2013.

[3] Duyu Tang, Bing Qin, and Ting Liu, "Document modeling with gated recurrent neural network for sentiment classification," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1422–1432.

[4] Tomávs Mikolov, "Statistical language models based on neural networks," *Presentation at Google, Mountain View, 2nd April*, 2012.

[5] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney, "Lstm neural networks for language modeling.," in *INTERSPEECH*, 2012, pp. 194–197.

[7] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3156–3164.

[8] Dmitriy Bespalov, Yanjun Qi, Bing Bai, and Ali Shokoufandeh, "Sentiment classification with supervised sequence embedding," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2012, pp. 159–174.

[9] Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning, "Semi-supervised recursive autoencoders for predicting sentiment distributions," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2011, pp. 151–161.

[10] Peng Li, Yang Liu, and Maosong Sun, "Recursive autoencoders for itg-based translation.," in *EMNLP*, 2013, pp. 567–577.

[11] T Mikolov and J Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, 2013.

[12] Dennis W Ruck, Steven K Rogers, Matthew Kabrisky, Mark E Oxley, and Bruce W Suter, "The multilayer perceptron as an approximation to a bayes optimal discriminant function," *IEEE Transactions on Neural Networks*, vol. 1, no. 4, pp. 296–298, 1990.

[13] Sankar K Pal and Sushmita Mitra, "Multilayer perceptron, fuzzy sets, and classification," *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 683–697, 1992.

[14] Nelson Morgan and Herve Bourlard, "Continuous speech recognition using multilayer perceptrons with hidden markov models," in *Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference on*. IEEE, 1990, pp. 413–416.

[15] Tony Mullen and Nigel Collier, "Sentiment analysis using support vector machines with diverse information sources.," in *EMNLP*, 2004, vol. 4, pp. 412–418.

[16] Frank Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, pp. 386, 1958.

[17] Rohini S Rahate and M Emmanuel, "Feature selection for sentiment analysis by using svm," *International Journal of Computer Applications*, vol. 84, no. 5, 2013.

[18] I Hemalatha, GP Saradhi Varma, and A Govardhan, "Preprocessing the informal text for efficient sentiment analysis," *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, vol. 1, no. 2, 2012.

[19] Apoorv Agarwal, Boyi Xie, Ilia Vovsha, Owen Rambow, and Rebecca Passonneau, "Sentiment analysis of twitter data," in *Proceedings of the workshop on languages in social media*. Association for Computational Linguistics, 2011, pp. 30–38.

[20] Shenghua Liu, Xueqi Cheng, Fuxin Li, and Fangtao Li, "Tasc: topic-adaptive sentiment classification on dynamic tweets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 6, pp. 1696–1709, 2015.

[21] Yoon Kim, "Convolutional neural networks for sentence classification.," in *EMNLP*, 2014, vol. 4, pp. 1746–1751.

[22] Aliaksei Severyn and Alessandro Moschitti, "Twitter sentiment analysis with deep convolutional neural networks," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2015, pp. 959–962.

[23] Xin Wang, Yuanchao Liu, Chengjie Sun, Baoxun Wang, and Xiaolong Wang, "Predicting polarities of tweets by composing word embeddings with long short-term memory," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, 2015, vol. 1, pp. 1343–1353.

[24] Kai Sheng Tai, Richard Socher, and Christopher D Manning, "Improved semantic representations from tree-structured long short-term memory networks," in *ACL*, 2015.

[25] Minlie Huang, Yujie Cao, and Chao Dong, "Modeling rich contexts for sentiment classification with lstm," *CoRR*, vol. abs/1605.01478, 2016.

[26] Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng, "Parsing with compositional vector grammars.," in *ACL (1)*, 2013, pp. 455–465.

[27] Nal Kalchbrenner and Phil Blunsom, "Recurrent continuous translation models.," in *EMNLP*, 2013, vol. 3, p. 413.

[28] Wenpeng Yin and Hinrich Schütze, "Discriminative phrase embedding for paraphrase identification," *NAACL-HLT*, 2016.

[29] Jiajun Zhang, Shujie Liu, Mu Li, Ming Zhou, Chengqing Zong, et al., "Bilingually-constrained phrase embeddings for machine translation.," in *ACL (1)*, 2014, pp. 111–121.

[30] Jeffrey L Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[31] Felix A Gers, Douglas Eck, and Jürgen Schmidhuber, "Applying lstm to time series predictable through time-window approaches," in *Artificial Neural NetworksICANN 2001*, pp. 669–676. Springer, 2001.

[32] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber, "Learning precise timing with lstm recurrent networks," *The Journal of Machine Learning Research*, vol. 3, pp. 115–143, 2003.

[33] Corinna Cortes and Vladimir Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[34] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al., "Scikit-learn: Machine learning in python," *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[35] Franois Chollet, "keras," `https://github.com/fchollet/keras`, 2015.

[36] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio, "Theano: new features and speed improvements," Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[37] Johan AK Suykens and Joos Vandewalle, "Least squares support vector machine classifiers," *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.

[38] Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin, "Recent advances of large-scale linear classification," *Proceedings of the IEEE*, vol. 100, no. 9, pp. 2584–2603, 2012.