Efficient Text Classification with Echo State Networks

Jérémie Cabessa[1,2,3], Hugo Hernault[1], Heechang Kim[1], Yves Lamonato[1], Yariv Z. Levy[1]

[1]Playtika Research
Lausanne, Switzerland

[2]Laboratory of Mathematical Economics and Applied Microeconomics (LEMMA)
University Paris 2 Panthéon-Assas, Paris, France

[3]Institute of Computer Science of the Czech Academy of Sciences
Prague, Czech Republic

{jeremiec, hugoh, heechangk, yvesl, yarivl}@playtika.com, jeremie.cabessa@u-paris2.fr

***Abstract*—We consider echo state networks (ESNs) for text classification. More specifically, we investigate the learning capabilities of ESNs with pre-trained word embedding as input features, trained on the IMDb and TREC sentiment and question classification datasets, respectively. First, we introduce a customized training paradigm for the processing of multiple input time series (the inputs texts) associated with categorical targets (their corresponding classes). For sentiment tasks, we use an additional frozen attention mechanism which is based on an external lexicon, and hence requires only negligible computational cost. Within this paradigm, ESNs can be trained in tens of seconds on a GPU. We show that ESNs significantly outperform their Ridge regression baselines provided with the same embedded features. ESNs also compete with classical Bi-LSTM networks while keeping a training time of up to 23 times faster. These results show that ESNs can be considered as robust, efficient and fast candidates for text classification tasks. Overall, this study falls within the context of light and fast-to-train models for NLP.**

***Index Terms*—reservoir computing, echo state networks, natural language processing, text classification**

I. Introduction

Recurrent neural networks (RNNs) refer to the class of artificial neural networks that contain recurrently interconnected units. This architecture endows the networks with valuable memory capabilities, making them ideal candidates for the learning of sequential data. Nowadays, recurrent neural networks are mainly used in the form of LSTM or GRU-like architectures and achieve remarkable learning capabilities in many domains [1]. However, the performance of these networks comes at a certain price: RNNs are hard to train [2].

Echo State Networks (ESNs) are specific types of recurrent neural networks that are fast-to-train and particularly well-suited for temporal tasks [3–6].[1] An ESN consists of an input layer projecting onto a sparse, recurrent and random *reservoir* of neurons, itself projecting onto an output layer (cf. Figure 1). During training, the input and reservoir weights remain fixed, and only the output weights are learned – generally via simple regression-like methods. In this architecture, the recurrent reservoir first performs a non-linear transformation of the sequential inputs, and the output layer then computes a simple mapping of the transformed inputs. Thus, ESNs can be considered as a *temporal kernel method* [9]. ESNs are easily and quickly trainable models that have been successfully applied to a wide variety of machine learning problems [9, 10]. Recently, they have been expanded into deeper architectures [11].

In the field of natural language processing (NLP), text classification tasks represent a major topic with numerous applications in business intelligence, marketing, finance, and politics, among others [12, 13]. In this domain like in many other applicative areas of machine learning, classical methods like linear Support Vector Machines [14] have nowadays been mainly supplanted by highly efficient deep learning approaches [15]. But deep methods often require a large amount of training data and can be slow to train.

Due to their recurrent nature as well as their simplicity of training, and because input texts are specific kinds of sequential data, ESNs represent a relevant fit for natural language processing problems. In fact, ESNs have already been used in the context of NLP. For instance, in an early seminal work, a first RNN-based language model not far from the reservoir computing approach was introduced by Elman [16]. More recently, a series of studies considering ESNs in the context of grammatical inference with applications in human-robot interaction has been proposed [17–21]. Furthermore, ESNs have been used in the context of named-entity recognition (NER), thus involving word-level rather than sentence-level classification features [23]. ESNs have also been used for text classification [22, 24], yet with different approaches from ours, resulting in significantly slower training times [24]. A more detailed discussion of these methods is provided in Section IV.

Here, we pursue the study of ESNs for text classification tasks. More specifically, we investigate the learning capabilities of ESNs with pre-trained word embedding as input features, trained on the IMDb and TREC sentiment and question classification datasets, respectively. First, we introduce a customized training paradigm for the processing of multiple input time series (the successive embedded texts) associated with

[1]A similar bio-inspired approach was introduced independently under the name of *Liquid State Machines (LSMs)* [7, 8].

categorical targets (their corresponding classes). For sentiment tasks, we make use of an external lexicon in order to build a straightforward frozen attention mechanism that requires only negligible computational cost. Within this paradigm, ESNs can be trained in tens of seconds on a GPU. We show that ESNs significantly outperform their Ridge regression baselines provided with the same embedded features. ESNs also compete with classical Bi-LSTM networks while keeping a training time of up to 23 times faster, depending on he dataset. These results show that ESNs can be considered as robust, efficient and fast candidates for text classification tasks. Overall, this study falls within the context of light and fast-to-train models for NLP.

II. Echo State Networks

A. Definition

An *echo state network (ESN)* is a recurrent neural network composed of N_u input units, N_x hidden units composing the so-called *reservoir*, and N_y output units (cf. Figure 1). The input units project onto the reservoir, which is itself recurrently connected, and projects onto the output units. The input-to-reservoir, reservoir-to-reservoir and reservoir-to-output connections are respectively given by the following weight matrices

$$\begin{aligned}\mathbf{W}_{\text{in}} &\in \mathbb{R}^{N_x \times (1+N_u)} \\ \mathbf{W}_{\text{res}} &\in \mathbb{R}^{N_x \times N_x} \\ \mathbf{W}_{\text{out}} &\in \mathbb{R}^{N_y \times (1+N_x)}\end{aligned}$$

where $\mathbf{W}(j,i)$ is the weight of the directed connection from cell i to cell j, for $\mathbf{W} \in \{\mathbf{W}_{\text{in}}, \mathbf{W}_{\text{res}}, \mathbf{W}_{\text{out}}\}$, and the first columns of $\mathbf{W}_{\text{in}}$ and $\mathbf{W}_{\text{out}}$ represent the biases of the reservoir and output cells, respectively.

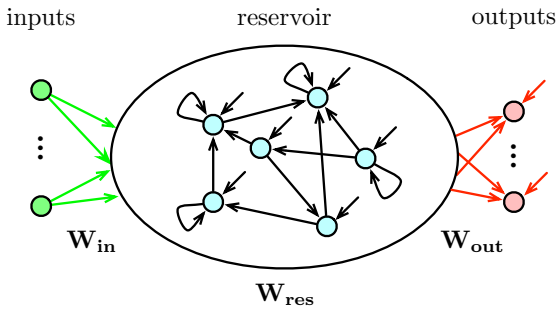

Fig. 1: An echo state network. The network consists of an input layer, a reservoir and an output layer connected together by the weights matrices $\mathbf{W}_{\text{in}}$, $\mathbf{W}_{\text{res}}$ and $\mathbf{W}_{\text{out}}$. The weights from $\mathbf{W}_{\text{in}}$ and $\mathbf{W}_{\text{res}}$ are fixed, while those from $\mathbf{W}_{\text{out}}$ (in red) are trainable.

In this work, we consider *Leaky Integrator ESNs*. The inputs, reservoir state and outputs of the network at time $t > 0$ are denoted by $\mathbf{u}(t) \in \mathbb{R}^{N_u}$, $\mathbf{x}(t) \in \mathbb{R}^{N_x}$ and $\mathbf{y}(t) \in \mathbb{R}^{N_y}$, respectively. The state $\mathbf{x}(0)$ is the *initial state*. The dynamics of the network is then given by the following equations:

$$\tilde{\mathbf{x}}(t+1) = f_{\text{res}}\left(\mathbf{W}_{\text{in}}[\mathbf{1}, \mathbf{u}(t+1)] + \mathbf{W}_{\text{res}}\mathbf{x}(t)\right) \quad (1)$$

$$\mathbf{x}(t+1) = (1-\alpha)\mathbf{x}(t) + \alpha\tilde{\mathbf{x}}(t+1) \quad (2)$$

$$\mathbf{y}(t+1) = f_{\text{out}}\left(\mathbf{W}_{\text{out}}[\mathbf{1}, \mathbf{x}(t+1)]\right) \quad (3)$$

where $[\mathbf{a}, \mathbf{b}]$ denotes the concatenation of $\mathbf{a}$ and $\mathbf{b}$, $\mathbf{x}(0)$ is the *initial state*, f_{res} and f_{out} are the *activation functions* of the reservoir and output cells (applied component-wise), and α is the *leaking rate* ($0 \leq \alpha \leq 1$). The leaking rate controls the update speed of the reservoir dynamics: larger leaking rates mean faster reacting reservoirs, since the contribution of the previous state in the updating process is reduced [10].

The input weights $\mathbf{W}_{\text{in}}$ are initialized randomly from a uniform distribution $\mathcal{U}(-a, a)$, where a is the *input scaling*. The input scaling determines the extent of nonlinearity of the reservoir response: in fact, larger input scalings will drive the reservoir units into larger activation values, where f_{res} operates in a more non-linear regime [10]. The input weights $\mathbf{W}_{\text{in}}$ are kept fixed during the whole training process.

The reservoir weights $\mathbf{W}_{\text{res}}$ are drawn from the uniform distribution $\mathcal{U}(-1, 1)$, and then randomly set to 0 with a sparsity rate of 99%. Afterwards, $\mathbf{W}_{\text{res}}$ is rescaled such that the *spectral radius*[2] of the matrix $\mathbf{W} = (1-\alpha)\mathbf{I} + \alpha\mathbf{W}_{\text{res}}$, denoted by $\rho(\mathbf{W})$, is equal to some desired value $\rho < 1$. Formally, setting

$$\mathbf{W}_{\text{res}} := \frac{\rho - (1-\alpha)}{\alpha\rho(\mathbf{W}_{\text{res}})}\mathbf{W}_{\text{res}}$$

ensures that $\rho(\mathbf{W}) = (1-\alpha) + \alpha\rho(\mathbf{W}_{\text{res}}) = \rho < 1$. The reservoir weights $\mathbf{W}_{\text{res}}$ are also kept fixed during training.

The rescaling of $\mathbf{W}_{\text{res}}$ is performed with the aim to satisfy the *echo state property (ESP)* – a set of mathematical conditions under which a consistent learning can be achieved [3, 25, 26]. Intuitively, the ESP states that, as the network processes its successive inputs, the induced reservoir states should depend less and less on the initial conditions and more and more on the input history – until asymptotically acting as an "echo" of the past inputs only. In practice, choosing some spectral radius $\rho < 1$ and rescaling $\mathbf{W}_{\text{res}}$ such that $\rho((1-\alpha)\mathbf{I} + \alpha\mathbf{W}_{\text{res}}) = \rho$ ensures that the echo state property is satisfied in most situations [9, 10]. The spectral radius ρ modulates the effect of the past inputs on the successive reservoir states: larger spectral radii correspond to longer input memories [10].

In an ESN, only the output weights $\mathbf{W}_{\text{out}}$ are trained. The training process can be described as follows. Consider some training set $\mathcal{S}$ composed of temporal inputs and associated targets, i.e.,

$$\mathcal{S} = \left\{\left(\mathbf{u}(t), \mathbf{y}^{\text{target}}(t)\right) : t = 1, \ldots, T\right\}.$$

Let $\mathbf{x}(1), \ldots, \mathbf{x}(T)$ and $\mathbf{y}(1), \ldots, \mathbf{y}(T)$ be the successive reservoir states and predictions obtained when running the ENS on inputs $\mathbf{u}(1), \ldots, \mathbf{u}(T)$, respectively (cf. Equations (1)–(3)). Then, the output weights $\mathbf{W}_{\text{out}}$ are computed by minimizing some cost function $\mathcal{L}$ of the predictions and targets via any desired learning algorithm. Usually, some initial transient of the ESN dynamics is used as a warm-up of the reservoir, and $\mathbf{W}_{\text{out}}$ is computed on the basis of

[2]The spectral radius of a matrix $\mathbf{W}$, denoted by $\rho(\mathbf{W})$, is the largest absolute value of the eigenvalues of $\mathbf{W}$.

the remaining suffix of collected states, predictions and and targets [10]. Note that an ESN with N_x reservoir units contains only $|\mathbf{W}_{\text{out}}| = N_y \times (1 + N_x)$ learning parameters (e.g., 2002 parameters for an ESN of size 1000 used for binary classification).

In this study, we simply let $\mathbf{W}_{\text{out}}$ be given by the closed-form solution of a Ridge regression, i.e.,

$$\mathbf{W}_{\text{out}}{}^T = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}^{\text{target}}$$

where $\mathbf{X}$ and $\mathbf{y}^{\text{target}}$ are the row-wise concatenations of reservoir states and targets, respectively, and $\lambda \in \mathbb{R}^+$ is the regularization parameter.

B. Many-to-one training paradigm

Classical temporal tasks involve time series where each point is associated with a corresponding target. By contrast, in the present case, the task comprises *multiple* time series as inputs – the successive embedded texts – each of which being associated with only *one* output target – its corresponding class. We propose a customized training process targeted at this *many-to-one* paradigm.

Consider some training set composed of input texts $(\tau_i)_{i=1}^T$ associated with classes $(\mathbf{y}_{\text{i}}^{\text{target}})_{i=1}^T$:

$$\mathcal{S} = \Big\{ \big(\tau_i, \mathbf{y}_i^{\text{target}}\big) : i = 1, \ldots, T \Big\}.$$

Let $\mathbf{E}$ be some *word embedding* of dimensions D which maps every text τ_i onto the matrix $\mathbf{E}_{\tau_i} \in \mathbb{R}^{|\tau_i| \times D}$, where the rows of $\mathbf{E}_{\tau_i}$ are the successive embedded words of τ_i. The many-to-one training procedure, illustrated in Figure 2, is described as follows:

1) **Warm up:** Run the ESN freely on a sufficiently large embedded text and record the final reservoir state $\mathbf{x}_{\text{warm}}$, referred to as the *warm state*.
2) **Embedding + ESN + merging:** For each text τ_i:
   a) Initialize the reservoir state to $\mathbf{x}_{\text{warm}}$.
   b) Embed text τ_i into the matrix $\mathbf{E}_{\tau_i} \in \mathbb{R}^{|\tau_i| \times D}$.
   c) Run the ESN on the sucessive inputs $\mathbf{E}_{\tau_i} \in \mathbb{R}^{|\tau_i| \times D}$ and collect the corresponding reservoir states $\mathbf{X}_{\tau_i} \in \mathbb{R}^{|\tau_i| \times N_x}$. The rows of $\mathbf{X}_{\tau_i}$ are the reservoir states obtained by running the ESN on the rows of $\mathbf{E}_{\tau_i}$.
   d) Apply a *merging strategy* $\text{ms} : \mathbb{R}^{|\tau_i| \times N_x} \to \mathbb{R}^{N_x}$ in order to transform the sequence of reservoir states $\mathbf{X}_{\tau_i}$ into a single *merged state* $\mathbf{x}_{\tau_i} = \text{ms}(\mathbf{X}_{\tau_i})$.
3) **Training:** Consider the new training set composed of merged states $(\mathbf{x}_{\tau_i})_{i=1}^T$ and associated targets $(\mathbf{y}_{\text{i}}^{\text{target}})_{i=1}^T$:

$$\mathcal{S}' = \Big\{ \big(\mathbf{x}_{\tau_i}, \mathbf{y}_i^{\text{target}}\big) : i = 1, \ldots, T \Big\},$$

   and compute $\mathbf{W}_{\text{out}}$ as the closed-form solution of a Ridge regression for this supervised learning problem.

Two main merging strategies are considered:

- *Mean:* each merged state $\mathbf{x}_{\tau_i}$ is simply the mean of the reservoir states (rows) of $\mathbf{X}_{\tau_i}$, i.e., $\mathbf{x}_{\tau_i} = \frac{1}{|\tau_i|}\mathbf{1}^T\mathbf{X}_{\tau_i}$.
- *Lexicon mean:* each merged state $\mathbf{x}_{\tau_i}$ is a weighted mean of the reservoir states of $\mathbf{X}_{\tau_i}$. The chosen weights are not learned but fixed. They are determined on the basis of the pre-annotated lexicon of the Semantic Orientation CALculator (SO-CAL), which associates each word x from the corpus with a polar sentiment score $\text{sent}(x) \in [-5, 5]$ [27]. On that basis, each state (row) $\mathbf{X}_{\tau_i,k}$ of $\mathbf{X}_{\tau_i}$ is associated with a positive weight[3] $w_{\tau_i,k}$ given by

$$w_{\tau_i,k} = \begin{cases} 1 + 2|\text{sent}(\tau_{i.k})| & \text{if the } k\text{-th word } \tau_{i.k} \text{ of } \tau_i \text{ is in the lexicon,} \\ 1 & \text{otherwise} \end{cases}$$

  for $k = 1, \ldots, |\tau_i|$. The merged state $\mathbf{x}_{\tau_i}$ is then given by the weighted mean of the reservoir states (rows) of $\mathbf{X}_{\tau_i}$, i.e., $\mathbf{x}_{\tau_i} = \frac{1}{\sum_k w_{\tau_i,k}}\mathbf{w}_{\tau_i}^T\mathbf{X}_{\tau_i}$ where $\mathbf{w}_{\tau_i} = (w_{\tau_i,1}, \ldots, w_{\tau_i,|\tau_i|})$. Note that the computation $\mathbf{w}_{\tau_i}$ only requires $|\tau_i|$ lookup operations in the lexicon, and no learning process.

For the sake of comparison, we also tested the absence of merging strategy – called the *none* merging strategy – as used in a previous study [22]. This case corresponds to a *many-to-many* training paradigm. For each $i = 1, \ldots, T$, the successive reservoir states (rows) $\big(\mathbf{X}_{\tau_i,k}\big)_{k=1}^{|\tau_i|}$ of $\mathbf{X}_{\tau_i}$ are not merged but associated with the same duplicated target $\mathbf{y}_i^{\text{target}}$. These considerations lead to the extended training set

$$\mathcal{S}'' = \Big\{ \big(\mathbf{X}_{\tau_i,k}, \mathbf{y}_i^{\text{target}}\big) : i = 1, \ldots, T \text{ and } k = 1, \ldots, |\tau_i| \Big\}.$$

The output weights $\mathbf{W}_{\text{out}}$ are then computed as the solution of a Ridge regression for this new supervised learning problem. In this context, the processing of each embedded input text $\mathbf{E}_{\tau_i}$ induces a sequence of predictions $\langle \mathbf{y}_{i,1}, \ldots, \mathbf{y}_{i,|\tau_i|} \rangle$, instead of a single one (cf. Equations (1)–(3)). The final prediction $\mathbf{y}_i$ associated with embedded input text $\mathbf{E}_{\tau_i}$ is then given as the average of the output sequence, i.e., $\mathbf{y}_i := \text{avg}\langle \mathbf{y}_{i,1}, \ldots, \mathbf{y}_{i,|\tau_i|} \rangle$.

Both many-to-one "mean" and "lexicon mean" merging strategies have the advantage of keeping the cardinality of the training set to the number of text samples ($|\mathcal{S}'| = T$). But they also have the disadvantage of aggregating the reservoir state dynamics in a relatively coarse manner. Still, the strategies turn out to work surprisingly well in practice. The many-to-many "none" merging strategy results in drastically reduced performance, both in terms of accuracy and training time.

C. Pre-trained embeddings

In this work, four classical pre-trained word embeddings of the same dimension 300 are considered: fastText en 300d [28], GloVe 6B-300d, GloVe 42B-300d and GloVe 840B-300d [29]. For the main results, we focused on GloVe 840B-300d. Once downloaded, the computational cost of the embedding process reduces to simple lookup matrix operations.

[3]The weighted mean strategy imposes the consideration of positive weights.

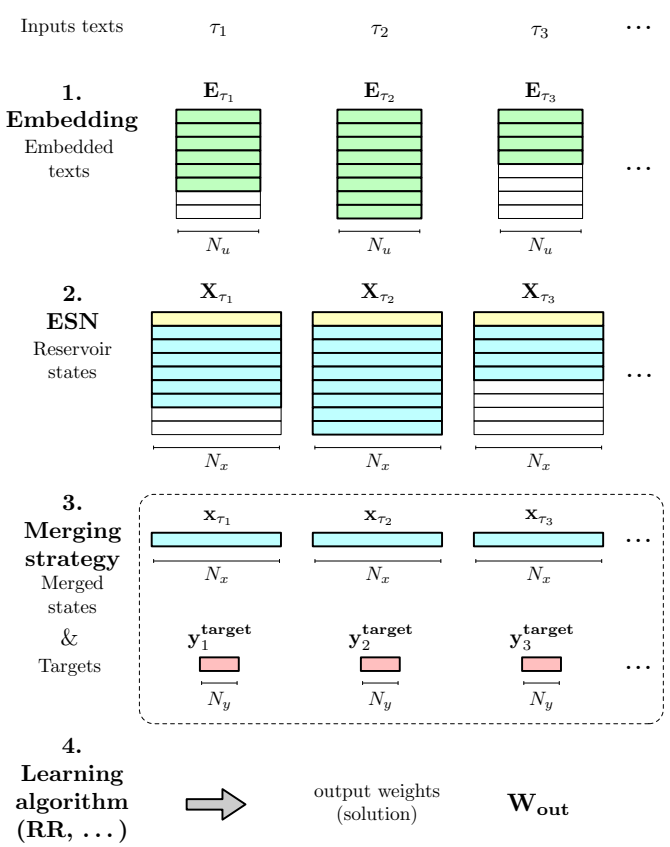

Fig. 2: Customized training paradigm of an echo state network. Horizontal rectangles represent vectors. Empty rectangles represent padding null vectors enabling batch parallelization of the training process. **1. Embedding:** each raw input text is tokenized and embedded into a sequence of input vectors (green). **2. ESN:** input vectors (green) are passed through the ESN with "warm" initial state (yellow), producing corresponding reservoir states (blue). **3. Merging strategy:** reservoir states are merged into a single merged state (blue). The process is repeated for all input texts. **4. Learning algorithm:** the output weights $\mathbf{W}_{\text{out}}$ are computed as the solution of a supervised learning problem whose inputs and outputs are the merged reservoir states (blue) and the categorical targets (red), respectively (dashed rectangle).

D. Datasets

In this study, the standard IMDb dataset for binary sentiment classification as well as the benchmark TREC-6 and TREC-50 datasets for fine-grained question classification are considered. In this way, two different domains of text classification and different class granularities are represented. The datasets are described in more detail below.

IMDb. The Large Movie Review Dataset (IMDb) is a balanced dataset for binary sentiment classification composed of $50'000$ movie reviews from IMDB labeled as positive or negative [30].

TREC. The Text REtrieval Conference dataset (TREC) for question classification consists of $5'952$ open-domain, fact-based questions divided into broad semantic categories [31]. It has both a six-class (TREC-6) and a fifty-class (TREC-50) version.

E. Implementation and hyperparameter tuning

The model was implemented in PyTorch 1.5.0 and the code run on 1 GPU (32 GB of memory) of a NVIDIA V100.

The hyperparameters of interest are the input scaling a, the leaking rate α, the spectral radius ρ and the regularization parameter λ. For each dataset, the tuning of a, α and ρ was performed by a grid search on 20% of the training set, using 3-fold cross validation and averaging over random 5 seeds, in order to attenuate the initialization effect of the ESN. Afterwards, for each reservoir size, the best hyperparameters a, α and ρ were kept fixed, and an additional tuning of λ was performed via linear search.

III. Results

This section analyzes the learning performance of ESNs on the IMDb and TREC datasets and compares it with that of corresponding RR-baselines and benchmark models.

A. Effect of embedding

We show that the quality of the pre-trained embedding plays a significant role in the performance of the network. In fact, contrary to what might be assumed at first sight, the representational properties of the embedding are actually preserved through the random input projection and reservoir trandsformation $\mathbf{W}_{\text{in}}$ and $\mathbf{W}_{\text{res}}$, respectively.

This feature might be explained as follows. Since the input and reservoir weights $\mathbf{W}_{\text{in}}$ and $\mathbf{W}_{\text{res}}$ are not learned but kept fixed, they are fixed finite-dimensional linear operators between normed spaces. Hence, $\mathbf{W}_{\text{in}}$ and $\mathbf{W}_{\text{res}}$ are bounded operators, and thus, are Lipschitz continuous. By composition of Lipschitz functions, the right hand side of Equation 2, denoted by $\Phi(\mathbf{u}(t+1))$, is also a Lipschitz function. This means that there exists $L \in \mathbb{R}^+$ such that $\|\Phi(\mathbf{u}) - \Phi(\mathbf{u}')\| \leq L\,\|\mathbf{u} - \mathbf{u}'\|$ for all $\mathbf{u}, \mathbf{u}' \in \mathbb{R}^{N_u}$. In other words, if L is not too large, the distance between embedded features is preserved through the input and reservoir projections.

These considerations are empirically confirmed by our analyses. We trained an ESN of size 1000 as well as its Ridge regression (RR) baseline (cf. Section III-B for a formal definition), both with the merging strategy "mean", on the IMDb dataset. The following four pre-trained embeddings of dimension 300 are used as input features: fastText en 300d [28], GloVe 6B-300d, GloVe 42B-300d and GloVe 840B-300d [29]. The test accuracies as a function of the embeddings are reported in Figure 3. For the RR-baseline, the accuracy for fasttext.en.300d is surprisingly larger than that for glove.6B.300d. More complex GloVe embeddings then lead to better performance. For the ESN, more complex embeddings consistently lead to significantly better accuracies. The highest improvement is achieved when passing from glove.6B.300d to glove.42B.300d. The same pattern is observed for other datasets.

B. Effect of the reservoir

By definition, the ESNs considered here consist of a pre-trained embedding layer (EMB), followed by a reservoir transformation (RES), followed by a Ridge regression layer (RR).

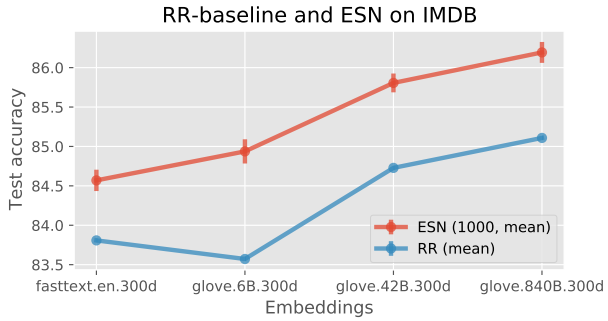

Fig. 3: Test accuracy of an ESN of reservoir size 1000 and its RR-baseline, both with the merging strategy "mean", on the IMDb dataset. Four pre-trained embeddings of dimension 300 are used as input features: fasttext.en.300d, glove.6B.300d, glove.42B.300d and glove.840B.300d. For the ESN, results are averaged over 10 random seeds. The points and errors bar represent the means and standard deviations of the results, respectively.

For each ESN, we define its corresponding *Ridge regression baseline* (RR-baseline) as the model composed of the pre-trained embedding layer (EMB) directly followed by the Ridge regression layer (RR). In this way, the comparison between an ESN (EMB + RES + RR) and its corresponding RR-baseline (EMB + RR) allows us to properly assess the contribution of the reservoir to the classification results.

The accuracy of ESNs with different reservoir sizes together with that of their corresponding RR-baselines are reported in Figure 4 and Tables I–III. All ESNs and associated baselines use the pre-trained embedding glove.840B.300d. The hyperparameters (input scaling a, leaking rate α and spectral radius ρ and regularization parameter λ) were obtained via grid search as described in Section II-E. The results are averaged over 5 random seeds. For RR-baselines, λ was also optimized via linear search. Since these methods are deterministic, no multiple seeds were used.

For all datasets and all merging strategies, the ESNs outperform their corresponding RR-baselines significantly. Regarding IMDb, the ESNs with reservoir size 500 already surpass their RR-baselines. Moreover, the accuracy increases monotonously with the reservoir size, but flattens for larger reservoirs. For reservoirs of size 5000, the ESNs exceed their baselines by about 2.5 points of accuracy. For TREC-6 and TREC-50, The ESNs also significantly surpass their RR-baselines, from a reservoir size of 500 onwards. For TREC-6, the accuracy increases up to a reservoir of 3000, and then stagnates for larger reservoirs. This effect may reflect an overfitting behavior of large reservoirs, or might be due to a non-optimal grid search for the optimization of hyperparameters. We would lean towards the second alternative. In this case, the ESNs with reservoir size 5000 outperform their RR-baselines by up to 12 points of accuracy. For TREC-50, the trend is monotonously increasing, with even a bouncing effect for large reservoirs. Here, the ESNs with reservoir size 5000 considerably surpass their RR-baselines by about 19 points of accuracy.

These results show that the temporal dynamics captured by the reservoir drastically improves the classification results.

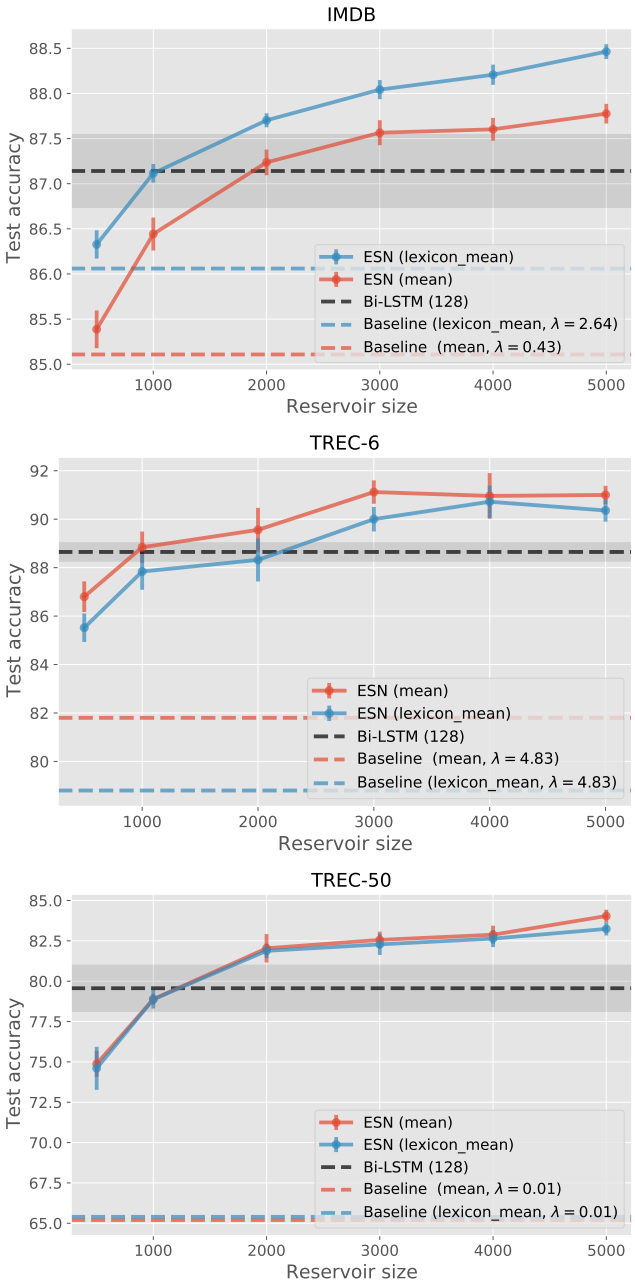

Fig. 4: Performance of ESNs, RR-baselines and Bi-LSTM over the datasets IMDb, TREC-6 and TREC-50. **Blue and red solid traces:** Test accuracy of ESNs with increasing reservoir size using the 'lexicon mean" (blue trace) and the "mean" (red trace) merging strategies, respectively. Results are averaged over 5 random seeds: the points and associated error bars represent the means and standard deviations of the accuracy, respectively. **Blue and red dashed traces:** Test accuracy of RR-baselines with 'lexicon mean' (blue trace) and "mean' (red trace) merging strategies, respectively. **Black traces:** Test accuracy of Bi-LSTM with 2×128 units trained during 50 epochs. Results are averaged over 5 random seeds. Dashed lines and shaded areas around them represent the mean and standard deviation of the accuracy, respectively.

C. *Effect of merging strategies*

Both "mean" and "lexicon mean" merging strategies have the great advantage of keeping the cardinality of the training set to the number of texts only ($|\mathcal{S}'| = T$). This characteristic

enables a fast training process. On the other hand, the "mean" strategy has the disadvantage of aggregating the reservoir states in a coarse way. The "lexicon mean" is more refined, but targeted at sentiment tasks only, due to the very nature of the SO-CAL lexicon. It can be regarded as a rudimentary *frozen attention mechanism* with negligible additional computational cost. Accordingly, it doesn't increase the training time. The "none" merging strategy, by contrast, increases the cardinality of the training set from $|\mathcal{S}'| = T$ to $|\mathcal{S}''| = \sum_{i=1}^{T} |\tau_i|$, resulting in an increased training time.

The results for the ESNs and their RR-baselines are reported in Tables I–III. For RR-baselines, we see that the "none" merging strategy leads to drastically inferior performance than its competitors, both in terms of accuracy and training time (e.g., for IMDb, the "none", "mean" and "lex. mean" merging strategies lead to accuracies of 76.25%, 85.11% and 86.06%, respectively). The largest performance gap is observed for the TREC-50 dataset, where the "none" and "mean" merging strategies lead to accuracies of 27.40% and 65.20%, respectively. The significant performance drop of the "none" merging strategy was also consistently observed in the context of ESNs, and for all datasets. Consequently, the study of this strategy has eventually been discarded.

We now focus on the "mean" and "lexicon mean" merging strategies for ESNs. As expected, in the context of a sentiment task (IMDb), the "lexicon mean" strategy performs significantly better than the "mean" one, providing a gain of about 0.7 accuracy point while maintaining an equivalent training time (cf. Figure 4 and Table I). For non-sentiment tasks (TREC-6, TREC-50), the "lexicon mean" strategy either slightly degrades or tend to coincide with the performance of the "mean" (cf. Figure 4 and Table II–III). For TREC-50, the performance of the "mean" and "lexicon mean" strategies are surprisingly similar.

Overall, despite their simplicity, both merging strategies work surprisingly well. For the sentiment task, the added value of the "lexicon mean" strategy is significant.

D. Comparison with Bi-LSTM and state-of-the-art models

LSTM and Bi-LSTM networks represent benchmark approaches for prediction and classification of sequential data. In the context of text classification, several state-of-the-art models are based on improved Bi-LSTM architectures [32].

In order to compare the performance of ESNs to that of benchmark models, we trained a standard Bi-LSTM network on the IMDb and TREC datasets. The network is composed of 1 hidden layer with 2×128 units. The cross-entropy loss and Adam optimizer with default parameters were used, and training was performed during 50 epochs. Results are reported in Figure 4 and Tables I–III. We see that ESN with sufficiently large reservoir size significantly outperform the Bi-LSTM networks. More importantly, we see that the training time of the ESNs is from 4 to more than 20 times faster than that of the Bi-LSTM networks. The difference in training time is more important for large datasets and small number of classes. Indeed, on TREC-50 and TREC-6, the training of ESNs is about 4 and 10 times faster than that of the Bi-LSTM, respectively. On IMDB, if is about 23 times faster. Clearly, increasing the number of layers and units of the Bi-LSTM networks would eventually lead to an over-performance of these models over the ESNs. But the training time and number of parameters would also be drastically increased.

As far as state-of-the-art models are concerned, on IMDB, the best reported model achieves an accuracy of 96.21% [33] (cf. NLP-progress). On TREC-6 and TREC-50, best models reach the remarkable accuracy of 98.07% [34] and 97.2% [35], respectively (cf. NLP-progress). For IMDB and TREC-6, those best models are children of the "transformer revolution" [36, 37]. They are composed of transformer-like deep feedforward (and not recurrent) architectures, and are pre-trained on huge corpora before being fine-tuned on downstream tasks.

	IMDb	
	Accuracy (%)	Training time (sec.)
RR (none)	76.25	16.97 (batch=64)[4]
RR (mean)	85.11	3.74 (batch=2048)
RR (lex. mean)	86.06	2.49 (batch=2048)
Bi-SLTM (128)	87.14 ± 0.40	899.32 ± 4.01
ESN (5000, mean)	87.78 ± 0.11	38.47 ± 0.14
ESN (5000, lex. mean)	$\mathbf{88.46 \pm 0.08}$	$\mathbf{39.10 \pm 0.23}$

Table I: Test accuracy and training time of the RR-baselines, Bi-LSTM and ESNs over the IMDb dataset. ESNs and RR-baselines are trained with the merging strategies "mean" and "lexicon mean". ESNs size is 5000 (see Figure 4 for other reservoir sizes). Bi-LSTM contains 1 hidden layer of 2×128 units and is trained during 50 epochs. All results are averaged over 5 seeds.

	TREC-6	
	Accuracy (%)	Training time (sec.)
RR (none)	56.20	1.92 (batch=2048)
RR (mean)	81.80	0.42 (batch=2048)
RR (lex. mean)	78.80	0.13 (batch=2048)
Bi-SLTM (128)	88.65 ± 0.38	27.04 ± 5.11
ESN (3000, mean)	$\mathbf{91.12 \pm 0.48}$	$\mathbf{2.64 \pm 0.12}$
ESN (4000, lex. mean)	90.72 ± 0.66	4.35 ± 0.09

Table II: Test accuracy and training time of the RR-baselines, Bi-LSTM and ESNs over the TREC-6 dataset.

	TREC-50	
	Accuracy (%)	Training time (sec.)
RR (none)	27.40	1.81 (batch=2048)
RR (mean)	65.20	0.15 (batch=2048)
RR (lex. mean)	65.40	0.13 (batch=2048)
Bi-SLTM (128)	79.56 ± 1.43	27.12 ± 0.93
ESN (5000, mean)	$\mathbf{83.96 \pm 0.23}$	$\mathbf{7.02 \pm 0.14}$
ESN (5000, lex. mean)	83.32 ± 0.27	7.13 ± 0.23

Table III: Test accuracy and training time of the RR-baselines, Bi-LSTM and ESNs over the TREC-50 dataset.

IV. Discussion

Regarding previous works, an important reservoir computing approach to grammatical inference with further applications in human-robot interaction has been proposed [17–21]. These studies are of a biological inspiration, exploiting the homology between the cortico-striatal system and reservoir

computing. In this context, the system is able to learn and predict in real-time the grammatical functions of successive semantic words (object, predicate, agent, recipient) contained in a sentence – with the aim to reconstructing its coded meaning. The nature of this task imposes a many-to-many training paradigm.

More recently, ESNs have been used on an authorship attribution task [22]. In this context, the ESNs are enhanced with three embedding as well as one feature extractor layers, all of them trained via gradient descent. Despite the many-to-one nature of the task, the networks are trained in a many-to-may manner corresponding to the "none" merging strategy described in the paper (use duplicated targets to achieve a 1-1 mapping between input words and targets). Results show that ESNs compete with SVM methods.

ESNs have also been used in the context of named entity recognition (NER) [23]. Here, various pre-trained embeddings are also used as features. Bi-directional ESNs with logistic regression or deep readout layers are trained via gradient descent. The NER task imposes a many-to-many training scheme. The method achieves competitive results in terms of accuracy and training times.

Finally, a relevant attention-based ESN model has been proposed in the context of question classification [24]. In this study, a multi-ring reservoir topology is considered. The many-to-one training scheme is achieved through the consideration of a self-attention mechanism which concatenates the bidirectional reservoir dynamics into a single contextual vector. The enhanced model is trained via gradient descent. The model achieves a high accuracy on the TREC-6 dataset ($93.5\% \pm 0.9$) with reasonable training time (65 ± 8 seconds). By comparison, our model achieves an accuracy of $91.12\% \pm 0.48$ and a training time of 2.64 ± 0.12 seconds on the same dataset. No other dataset is considered in their study.

V. Conclusion

We studied the the learning capabilities of ESNs with pre-trained word embedding as input features on two text classification tasks: sentiment analysis and question classification. More specifically, we first proposed a customized training paradigm targeted to the many-to-one nature of the tasks. In this context, the many-to-one reduction of the networks' dynamics is achieved by means of straightforward merging strategies. This simplified representation of the sequential inputs is actually capable of retaining relevant information from the ESN dynamics. As a result, competitive performance with extremely fast training times are obtained.

Nowadays, state-of-the-art NLP models achieve remarkable performance on more and more complex tasks. Oftentimes, the best models are children of the "transformer revolution" [36, 37]. In this context, the consideration of deep feedforward architectures coupled with attention mechanisms and a specific 2-step training paradigm – pre-training and fine-tuning – are capable of surpassing the power of recurrent architectures. But those models are often monstrous in terms resources, even if very recently, a lot of progress has been made towards the design of smaller, faster, cheaper and lighter versions of them [38].

In contrast, this study falls within the context of light and fast-to-train NLP models. Our choice of rudimentary merging strategies, as opposed to trainable attention-based ones, has been made with the intention of prioritizing the size and speed of the models. Our results show that ESNs can be considered as robust, efficient and fast ML methods for text classification.

Acknowledgments

This work was partially supported by the Czech Science Foundation under grant number GA19-05704S. The authors are grateful to Playtika Ltd. for contributing to an inspiring R&D environment.

References

[1] K. Greff, R. K. Srivastava, J. Koutnìk, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[2] Y. Bengio, P. Y. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[3] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks," GMD - German National Research Institute for Computer Science, GMD Report 148, 2001.

[4] ——, "Short term memory in echo state networks," GMD - German National Research Institute for Computer Science, GMD-Report 152, 2002.

[5] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," *Science*, vol. 304, no. 5667, pp. 78–80, 2004.

[6] H. Jaeger, "Echo state network," *Scholarpedia*, vol. 2, no. 9, p. 2330, 2007.

[7] W. Maass and H. Markram, "On the computational power of circuits of spiking neurons," *J. Comput. Syst. Sci.*, vol. 69, no. 4, pp. 593–616, 2004.

[8] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[9] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[10] M. Lukoševičius, *A Practical Guide to Applying Echo State Networks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 659–686.

[11] C. Gallicchio, A. Micheli, and L. Pedrelli, "Design of deep echo state networks," *Neural Networks*, vol. 108, pp. 33–47, 2018.

[12] B. Pang and L. Lee, "Opinion mining and sentiment analysis," *Foundations and Trends in Information Retrieval*, vol. 2, no. 1-2, pp. 1–135, 2008.

[13] B. Altinel and M. Ganiz, "Semantic text classification: A survey of past and recent advances," *Information Processing & Management*, vol. 54, no. 6, pp. 1129 – 1153, 2018.

[14] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

[15] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao, "Deep Learning Based Text Classification: A Comprehensive Review," *arXiv e-prints*, p. arXiv:2004.03705, Apr. 2020.

[16] J. L. Elman, "Learning and development in neural networks: the importance of starting small," *Cognition*, vol. 48, no. 1, pp. 71–99, 1993.

[17] P. F. Dominey, M. Hoen, and T. Inui, "A neurolinguistic model of grammatical construction processing," *Journal of Cognitive Neuroscience*, vol. 18, no. 12, pp. 2088–2107, 2006.

[18] M. H. Tong, A. D. Bickett, E. M. Christiansen, and G. W. Cottrell, "Learning grammatical structure with echo state networks," *Neural Networks*, vol. 20, no. 3, pp. 424–432, 2007.

[19] X. Hinaut and P. F. Dominey, "Real-time parallel processing of grammatical structure in the fronto-striatal system: A recurrent network simulation study using reservoir computing," *PLOS ONE*, vol. 8, no. 2, pp. 1–18, 02 2013.

[20] X. Hinaut, M. Petit, G. Pointeau, and P. F. Dominey, "Exploring the acquisition and production of grammatical constructions through human-robot interaction with echo state networks," *Frontiers in Neurorobotics*, vol. 8, p. 16, 2014.

[21] X. Hinaut, F. Lance, C. Droin, M. Petit, G. Pointeau, and P. F. Dominey, "Corticostriatal response selection in sentence production: Insights from neural network simulation with reservoir computing." *Brain Lang*, vol. 150, pp. 54–68, 2015.

[22] N. Schaetti, "Behaviors of reservoir computing models for textual documents classification," in *International Joint Conference on Neural Networks, IJCNN 2019 Budapest, Hungary, July 14-19, 2019*. IEEE, 2019, pp. 1–7.

[23] R. Ramamurthy, R. Stenzel, R. Sifa, A. Ladi, and C. Bauckhage, "Echo state networks for named entity recognition," in *ICANN 2019, Proceedings*, ser. LNCS, I. V. Tetko, V. Kůrkovà, P. Karpov, and F. Theis, Eds., no. 11731. Springer, 2019, pp. 110–120.

[24] D. Di Sarli, C. Gallicchio, and A. Micheli, "Question classification with untrained recurrent embeddings," in *AI*IA 2019, Proceedings*, ser. Lecture Notes in Computer Science, M. A. et al., Ed., vol. 11946. Springer, 2019, pp. 362–375.

[25] I. B. Yildiz, H. Jaeger, and S. J. Kiebel, "Re-visiting the echo state property," *Neural Networks*, vol. 35, pp. 1–9, 2012.

[26] M. Gandhi and H. Jaeger, "Echo state property linked to an input: Exploring a fundamental characteristic of recurrent neural networks," *Neural Computation*, vol. 25, no. 3, pp. 671–696, 2013.

[27] M. Taboada, J. Brooke, M. Tofiloski, K. D. Voll, and M. Stede, "Lexicon-based methods for sentiment analysis," *Comput. Linguistics*, vol. 37, no. 2, pp. 267–307, 2011.

[28] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *CoRR*, vol. abs/1607.04606, 2016.

[29] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *EMNLP 2014, Proceedings*, A. Moschitti, B. Pang, and W. Daelemans, Eds. ACL, 2014, pp. 1532–1543.

[30] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *ACL HLT 2011, Proceedings*, D. Lin, Y. Matsumoto, and R. Mihalcea, Eds. ACL, 2011, pp. 142–150.

[31] X. Li and D. Roth, "Learning question classifiers," in *COLING 2002, Proceedings*. ACL, 2002.

[32] P. Zhou, Z. Qi, S. Zheng, J. Xu, H. Bao, and B. Xu, "Text classification improved by integrating bidirectional LSTM with two-dimensional max pooling," in *COLING 2016, Proceedings*, N. Calzolari, Y. Matsumoto, and R. Prasad, Eds. ACL, 2016, pp. 3485–3495.

[33] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *NIPS 2019, Proceedings*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 5753–5763.

[34] D. Cer, Y. Yang, S. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, Y. Sung, B. Strope, and R. Kurzweil, "Universal sentence encoder," *CoRR*, vol. abs/1803.11175, 2018.

[35] H. T. Madabushi and M. Lee, "High accuracy rule-based question classification using question syntax and semantics," in *COLING 2016, Proceedings*, N. Calzolari, Y. Matsumoto, and R. Prasad, Eds. ACL, 2016, pp. 1220–1230.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS 2017, Proceedings*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 5998–6008.

[37] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT 2019, Proceedings*, J. Burstein, C. Doran, and T. Solorio, Eds., vol. 1. ACL, 2019, pp. 4171–4186.

[38] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter," *CoRR*, vol. abs/1910.01108, 2019. [Online]. Available: http://arxiv.org/abs/1910.01108