Neural Networks 79 (2016) 1–11

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Regular expressions for decoding of neural network outputs

Tobias Strauß[*], Gundram Leifert, Tobias Grüning, Roger Labahn

Department of Mathematics, University of Rostock, Rostock, Germany

ARTICLE INFO

Article history:
Received 15 September 2015
Received in revised form 21 February 2016
Accepted 11 March 2016
Available online 25 March 2016

Keywords:
Regular expressions
Decoding
Connectionist Temporal Classification
Handwriting recognition

ABSTRACT

This article proposes a convenient tool for decoding the output of neural networks trained by Connectionist Temporal Classification (CTC) for handwritten text recognition. We use regular expressions to describe the complex structures expected in the writing. The corresponding finite automata are employed to build a decoder. We analyze theoretically which calculations are relevant and which can be avoided. A great speed-up results from an approximation. We conclude that the approximation most likely fails if the regular expression does not match the ground truth which is not harmful for many applications since the low probability will be even underestimated. The proposed decoder is very efficient compared to other decoding methods. The variety of applications reaches from information retrieval to full text recognition. We refer to applications where we integrated the proposed decoder successfully.

1. Introduction

Sequence labeling is the task of assigning a (class) label to each position of an incoming sequence such as speech or handwriting recognition. These tasks are typically very complex and even subproblems are challenging. This article focuses on the decoding problem i.e. finding the most likely label sequence for a given output of a classifier such as neural networks (NNs), Hidden Markov Models (HMMs) or Conditional Random Fields (CRFs).

Deep learning methods have pushed the research of complex tasks such as handwritten text recognition (see Graves & Schmidhuber, 2009). The special needs of such complex tasks require advanced decoding methods. For example, a typical subproblem in full text recognition is structuring the recognizers output into a sequence of regions of words, punctuations and numbers. In many cases, the most likely label sequence yields an acceptable segmentation. However, it happens that this label sequence is not feasible i.e. it does not match the expected structure and has to be corrected. Finding the optimal feasible structure is one of many applications of this article. For this aim, we describe feasible structures by regular expressions—a powerful pattern sequence which is used in nearly all computational text processing systems such as text editors and programming languages like Java or Python. We then derive an algorithm based on finite automata that yields the most likely label sequence fitting the previously described regular expression.

Beyond finding the optimal feasible label sequence fitting an expected structure (regular expression), we gain several other features since we also consider the functionality of capturing groups. A capturing group defines a part of the regular expression. The associated part of the matching label sequence can be used to structure the decoding result for further analysis. In case of our previous example, we obtain a complete segmentation into words, numbers and symbols without additional parsing facilitating the calculation of the matching subsequence and the likelihood. We just define word, number and symbol capturing groups. The complete decoding can be done in a few lines of code.

Keyword spotting is another obvious application which can be solved very conveniently. The keyword is either the beginning of the line or there is a space or another separating symbol (quotation marks, opening parenthesis, etc.) before the keyword. With the common notation of regular expressions, this pattern may be captured by inserting `(.*(?<pre>[ "(-]))?` before the keyword which means: If there is anything before the keyword, it ends with at least one of the aforementioned symbols. This last symbol (if there is one) is contained in the capturing group `pre`. Information about a group like its probability, containing text or its positions in the sequence are very important for the keyword spotting and will be provided directly by the derived algorithm. A low probability of the `pre`-group, for example, might indicate that a letter is more likely than our separating symbol such that the spotted character sequence is only part of a larger word. Analogously, there is an equivalent group after the keyword.

[*] Corresponding author.
E-mail addresses: tobias.strauss@uni-rostock.de (T. Strauß), gundram.leifert@uni-rostock.de (G. Leifert), tobias.gruening@uni-rostock.de (T. Grüning), roger.labahn@uni-rostock.de (R. Labahn).

http://dx.doi.org/10.1016/j.neunet.2016.03.003
0893-6080/© 2016 Elsevier Ltd. All rights reserved.

Regular expressions can be very complex and the calculation of the probability of all feasible sequences can be very time consuming. We give an approximation of the most likely label sequence which we motivate theoretically and experimentally. The approximation is also fundamental to the proposed decoder since a conventional A^*-search suffers from a combinatorial explosion of all feasible sequences and leads to inefficient decoding times. It is developed for *neural networks trained by Connectionist Temporal Classification (CTC)*. Thus, CTC-trained systems are assumed all over the paper. Some of the currently most successful handwriting recognition systems were trained with CTC as shown in several competitions. To give just one example, the probably most challenging real world task is the Maurdor project which was won by A2IA in 2014 using CTC (see Moysset et al., 2014). CTC is not limited to text recognition. Recently the performance of several speech recognition systems trained with CTC equaled those of other state of the art methods (e.g. Graves & Jaitly, 2014, Sak et al., 2015).

The proposed algorithm is an essential part of the award winning systems (Leifert, Grüning, Strauß, & Labahn, 2014; Strauß, Grüning, Leifert, & Labahn, 2014) which were also trained with CTC. Recently, the system reaffirmed the capability by winning the HTRtS15 competition (Sánchez, Toselli, Romero, & Vidal, 2015).

The performant connection between regular expressions and machine learning algorithms has been investigated in previous articles. In the context of speech recognition, Mohri, Pereira, and Riley (2008) showed in detail how to incorporate static prior knowledge like n-grams or phoneme models into finite state transducers. Although the authors exploit similar models to do the decoding, the purpose differs from ours since they model more static connections between ton, speech and language while we aim at a flexible, adaptive decoding algorithm. Earlier, Dupont, Denis, and Esposito (2005) provided a comprehensive analysis of links between probabilistic automata (i.e. automata with a probabilistic transition) and HMMs from a theoretical point of view finally concluding – among other – that there is a correspondence between both models. This basically means, HMMs can be seen as the probabilistic version of finite automata.

Some links between regular expressions, their corresponding automata and HMMs are given in Krogh et al. (1998). The authors showed how to create HMMs from regular expressions to detect biological sequences. A similar but generalized approach is given in Kessentini, Chatelain, and Paquet (2013). There the authors construct a simplified HMM model for a general text line in the context of word spotting. These text line models basically consist of the keyword surrounded by space and filler models. They also proposed an enhanced model where only the prefix or suffix of the keyword is given. This model allows a set of feasible words containing the defined prefix or suffix.

Recently, Bideault et al. published a similar approach to ours in Bideault, Mioulet, Chatelain, and Paquet (2015). They proposed an HMM–BDLSTM hybrid model for word spotting exploiting regular expressions. Their model uses the posterior probability of the network as emission probability of the HMM (which means using $\mathrm{P}(y|x)$ as estimator for $\mathrm{p}(x|y)$, where x is the hidden variable and y is the observation). Analogously to Kessentini et al. (2013), they build small HMM models in advance (e.g. for a keyword, for digits or letters) and combine them to a model capturing the regular expression. The authors then applied their model to keyword and "regex" spotting.

In contrast to the above articles, we do not make use of an HMM model. Yet in Kessentini et al. (2013), the HMMs work only as convenient graphical model for decoding rather than as classifier. Instead of using a generative model to find the most likely sequence, our algorithm is based on the original graphical structure of the regular expressions: The finite state automata. If the automaton accepts a label sequence, it is feasible. Hence, we are able to search in the output of a neural network for any regular expression without any previously created or trained generative model. That means as input simply serve a regular expression and the network's output matrix and the output is the most likely sequence, their probability or the capturing groups defined by the regular expression.

The remainder of this article is organized as follows: We first give a formal definition of decoding (Section 2). In Section 3, we give a brief introduction to regular expressions and automata. Furthermore, we modify the automaton slightly to adapt it to the NN-decoding requirements. We introduce the RegEx-Decoder in Section 4. We finish with some experiments (Section 5) and a conclusion. Appendix provides the proofs of our theorems for theoretically interested readers.

2. Training and decoding

This section introduces the CTC training scheme for neural networks and some basic aspects of their decoding. We mainly follow the notation of Graves, Fernández, Gomez, and Schmidhuber (2006).

Let Σ be the alphabet and $\Sigma' = \Sigma \cup \{\star\}$ where $\star$ is an artificial garbage label (also called blank) indicating that none of the labels from Σ are present. We call the garbage label *not a character (NaC)* in the following. An element of Σ is called *character* and appears in the ground truth. Sequences from $\Sigma^* := \bigcup_{t\in\mathbb{N}} \Sigma^t$ are called *words*. Elements of Σ' are *labels* and represent different classes of the NN. Sequences of $(\Sigma')^*$ are called *paths*. The most likely path is called *best path*. Assume a neural network which maps an input $\boldsymbol{X}$[1] to a matrix $\boldsymbol{Y} \in \bigcup_{T=1}^{\infty} [0,1]^{T\times|\Sigma'|}$ of probabilities per position and label. i.e. $y_{t,l}$ denotes the probability for the lth label at position t. Note that we assume that $\forall t : \sum_l y_{t,l} = 1$ and $\forall t, l : y_{t,l} > 0$ throughout the paper.

To map a path $\boldsymbol{\pi}$ to a word $\boldsymbol{z}$, one merges consecutive identical π_t and deletes the *NaC*s. Let $\mathcal{F} : (\Sigma')^* \to \Sigma^*$ define the related function which maps a path to a word. More precisely: $\mathcal{F}(\boldsymbol{\pi}) = \mathcal{D}(\mathcal{S}(\boldsymbol{\pi}))$ is the composition of two functions $\mathcal{D}$ and $\mathcal{S}$ where $\mathcal{S}$ deletes all consecutive identical labels and $\mathcal{D}$ deletes all remaining *NaC*s.

We assume that the likelihoods $y_{t,c}$ are conditionally independent for distinct t given $\boldsymbol{X}$. Thus, the likelihood of any path $\boldsymbol{\pi}$ is given as

$$\mathrm{P}(\boldsymbol{\pi}|\boldsymbol{X}) = \prod_{t=1}^{T} y_{t,\pi_t}. \tag{1}$$

The probability of any word $\boldsymbol{z}$ is then the sum of the probabilities of all paths mapping to $\boldsymbol{z}$:

$$\mathrm{P}(\boldsymbol{z}|\boldsymbol{X}) = \sum_{\boldsymbol{\pi}\in\mathcal{F}^{-1}(\boldsymbol{z})} \mathrm{P}(\boldsymbol{\pi}|\boldsymbol{X}).$$

Let $\bar{\boldsymbol{z}} \in (\Sigma')^*$ be the extension of the word $\boldsymbol{z} \in \Sigma^*$, that means we add a *NaC* before $\boldsymbol{z}$, after $\boldsymbol{z}$ and between each pair of characters. Thus, $|\bar{\boldsymbol{z}}| = 2|\boldsymbol{z}| + 1$. Then one could calculate $\mathrm{P}(\boldsymbol{z}|\boldsymbol{X})$ in an iterative manner: The forward variable $\alpha_i(t)$ denotes the probability of the prefix $z_1, \ldots, z_{\lceil \frac{i-1}{2} \rceil}$ of $\boldsymbol{z}$ at time t given $\boldsymbol{X}$ and, hence, $\alpha_1(t)$ denotes the probability of the empty word prefix. Thus,

$$\alpha_1(t) = \prod_{t'=1}^{t} y_{t',\bar{z}_1} = \prod_{t'=1}^{t} y_{t',\star}.$$

[1] In contrast to $\boldsymbol{Y}$, both dimensions of $\boldsymbol{X}$ may vary.