New Recurrent Neural Network Variants for Sequence Labeling

Marco Dinarelli and Isabelle Tellier

LaTTiCe (UMR 8094), CNRS, ENS Paris, Université Sorbonne Nouvelle - Paris 3
PSL Research University, USPC (Université Sorbonne Paris Cité)
1 rue Maurice Arnoux, 92120 Montrouge, France
marco.dinarelli@ens.fr, isabelle.tellier@univ-paris3.fr

AUTHORS' DRAFT

March 24, 2016

Abstract

In this paper we study different architectures of Recurrent Neural Networks (RNN) for sequence labeling tasks. We propose two new variants of RNN and we compare them to the more traditional RNN architectures of Elman and Jordan. We explain in details the advantages of these new variants of RNNs with respect to Elman's and Jordan's RNN. We evaluate all models, either new or traditional, on three different tasks: POS-tagging of the French Treebank, and two tasks of Spoken Language Understanding (SLU), namely ATIS and MEDIA. The results we obtain clearly show that the new variants of RNN are more effective than the traditional ones.

1 Introduction

Recurrent Neural Networks [1, 2] are effective neural models able to take *context* into account for their decision function. For this reason, they are suitable for NLP tasks, in particular for sequence labeling [3, 4, 5, 6, 7, 8], where taking previous labels into account and modeling label dependencies are crucial for the design of effective models. It is indeed not surprising that Conditional Random Fields (CRF) [9] are among the best models for sequence labeling tasks, because their global decision function takes all possible labelings into account to predict the best labeling of a given sequence.

Most of the probabilistic models for sequence labeling (even earlier NLP models) are able to take some context into account for their decision function. In RNN, the contextual information is given by a loop connection in the network architecture. This

connection allows to keep, at the current time step, one or more pieces of information predicted at previous time steps. The effectiveness of recurrent architectures for NLP tasks comes from the combination of contextual information and distributional representations, or embeddings.

In the literature about RNNs for NLP tasks, mainly two architectures have been used, also called *simple* RNNs: Elman RNN [2] and Jordan RNN [1]. The difference between these two models lies in the recurrent connection: in Elman RNN the connection is a loop in the hidden layer. This connection allows Elman networks to use, at the current time step, hidden layer's states of previous time steps. In contrast, in Jordan RNN, the recurrent connection is between the output and the hidden layers. In this case the contextual information is made of one on more labels predicted at previous time steps. Since, in this work, we are dealing with sequence labeling tasks, time steps are *positions* in a given sequence. In the last few years, Elman and Jordan RNNs have been very successful in NLP, especially for language modeling [10, 11], but also for sequence labeling [3, 4, 5, 6, 7, 12].

The intuition which originated this paper is that embeddings allow a fine and effective modeling, not only of words, but also of labels and their dependencies, which are very important in sequence labeling tasks. Accordingly, we define two new variants of RNN to achieve this more effective modeling.

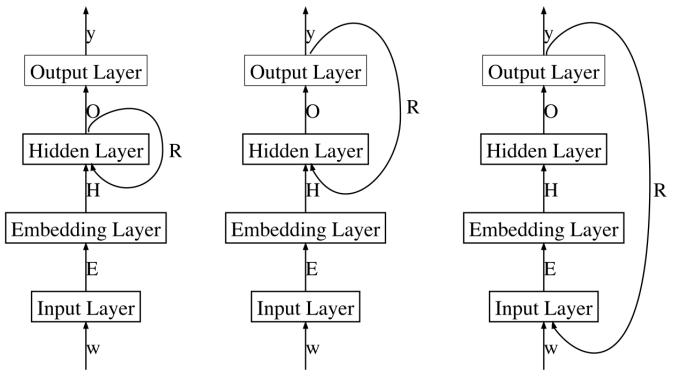
In the first variant, the recurrent connection is between the output and the input layers. In other words, this variant just gives labels predicted at previous positions in a sequence as inputs to the network. This contextual information is added to the usual input context made of words, and both are used to predict the label at the current position in the sequence. According to our intuition, and thanks to label embeddings, this variant of RNNs models label dependencies more effectively than in previous simple RNNs. The second variant we propose is a combination of an Elman RNN and of our first variant. This second variant can thus exploit both contextual information provided by the previous states of the hidden layer and the labels predicted at previous positions of a sequence.

A high-level schema of Elman's, Jordan's and our first variant of RNNs are shown in Figure 1. The schema of the second variant proposed in this paper can be obtained by adding the recursion of our first variant to the Elman architecture. In this figure, w is the input word, y is the predicted label, E, H, O et R are the parameter matrices between each pair of layers, they will be described in details in the next section. Before this, it is worth discussing the advantages our variants can bring with respect to traditional RNN architectures, and why they are expected to provide better modeling abilities.

First, since the output at previous positions is given as input to the network at the current position, the contextual information flows across the whole network, thus affecting each layer's input and output, and at both forward and backward phases. This allows in return more effective learning. In contrast, in Elman and Jordan RNNs, not all layers are affected at both forward and backward phases.

A second advantage of our variants is given by the use of label embeddings. Indeed, the first layer of our RNNs is just a *look-up* table mapping sparse "one-hot" representations into distributional representations.[1] Since in our variants the output of

[1]The "one-hot" representation of an element at position i in a dictionary V is a vector of size $|V|$ where



(a) Elman RNN (b) Jordan RNN (c) Our variant of RNN

Figure 1: High Level Schema of the Main RNNs Studied in this Work.

the network at previous steps is given as input at the current step, the mapping from sparse representations to embeddings involves both words and labels. Label embeddings can be pre-trained from data as it is usually done for words. Pre-trained word embeddings, e.g. with *word2vec*, have already proved their ability to capture very attractive syntactic and semantic properties [13, 14]. Using label embeddings, the same properties can also be learned for labels. More importantly, using several predicted labels as embeddings more effectively models label dependencies via the internal state of the network, which is the hidden layer's output.

As a third advantage, label embeddings give the network a stronger "signal" than what is ensured by the recurrent connection of traditional Elman and Jordan RNNs. In the case of Elman RNN, the recurrent connection is a loop in the hidden layer. It thus does not affect the embedding layer in the forward propagation phase. Moreover, Elman RNN does not use previous label predictions as contextual information at all. In contrast, in Jordan RNN, previous labels are used as contextual information since they are given as input to the hidden layer of the network. However, the representation used for labels in Jordan RNN is either a sparse "one-hot" representation, or a probability distribution over labels. It is clear that sparse "one-hot" representations do not provide much information to the network. In contrast, probability distributions over predicted labels allow a softer decision criterion. However, from our analysis, even probability distributions over labels do not provide much more information than sparse "one-hot" representations, as most of the probability mass is concentrated on one or few labels. For instance, this explains why [6] obtains a more effective Jordan RNN when training the network with gold labels than with both a sparse "one-hot" representation and a probability distribution over predicted labels.

Another advantage of having the recurrent connection between output and input layers, and so using a context of label embeddings, is an increased robustness of the

the i-th component has the value 1 while all the others are 0.

model to prediction mistakes. This increased robustness comes from the syntactic and semantic properties embeddings can encode [14]. In contrast, Jordan RNN, using sparse representations or "picked" probability distribution representations of predicted labels, is more affected by prediction error propagation.

All these advantages are also provided by the second variant of RNN proposed in this paper, which uses both a label embeddings context, like the first variant, and the loop connection at the hidden layer, like an Elman RNN.

In order to have a fair and straightforward comparison, together with the results of our new variants of RNNs, we provide the results obtained with our implementation of Elman and Jordan RNNs. Our Elman and Jordan RNNs are very close to the state-of-the-art and not just simple baselines, even if we did not implement every optimization features.

All models are evaluated on the POS-tagging task of the French Treebank [15, 16] and on two Spoken Language Understanding (SLU) tasks [17]: ATIS [18] and MEDIA [19], which can be both modeled as sequence labeling tasks. ATIS is a relatively simple task, where effective label dependencies modeling is not really an issue. POS-tagging, given the absolute magnitude of results [16], can be also considered as a relatively simple task in this respect. MEDIA, in contrast, is a more difficult task where the ability of a model to take label dependencies into account is crucial. Evaluating our models in these different conditions allows us to show the general effectiveness of our models, but also their ability to take label dependencies into account.

The results we obtain on these tasks with our implementations, despite they are not always better than the state-of-the-art, provide a stable ranking of the various RNN architectures: at least one of our variants, surprisingly the simpler one, is always better than Jordan and Elman RNNs.

The remainder of the paper is organized as follows: in the next section we describe in details RNNs, starting from traditional Elman and Jordan architectures, to arrive to our variants. In section 3, we describe the corpora used for evaluation, the parameter setting and all the results obtained, in comparison with state-of-the-art models. In section 4 we draw some conclusions.

2 Recurrent Neural Networks

2.1 Elman and Jordan RNNs

The RNNs we consider in this work have the same architecture used also for Feedforward Neural Network Language Models (NNLM) described in [20]. In this architecture we have 4 layers: input, embedding, hidden and output layers. Words are given as input to the network as indexes, corresponding to their position in a dictionary V.

The index of a word is used to select its embedding (or distributional representation) in a real-valued matrix $E \in R^{|V|XN}$, $|V|$ being the size of the dictionary and N being the dimensionality of the embeddings (which is a parameter to be chosen). As shown in figure 1, the matrix E is the set of parameters between the input and the embedding layers, and we name $E(v(w_t))$ the embedding of the word w given as input at the position t of a sequence. $v(w_t) = i$ is the index of the word w_t in the dictionary,

and it can be seen alternatively as a "one-hot" vector representation (the vector is zero everywhere except at position $v(w)$, where it has value 1).

We name H and O the real-valued matrices of parameters between the embedding and the hidden layers, and between the hidden and output layers, respectively. The output of the network, given the input w_t, is y_t.

The activation function at the hidden layer is a sigmoid.[2] The output layer is a softmax[3], the output of the network y is thus a probability distribution over output labels L.

In contrast to NNLM, RNNs have one more connection, the recursive connection, between two layers depending on the type of RNNs: as mentioned in the previous section, Elman RNNs [2] have a recursion loop at the hidden layer. Since the hidden layer encodes the internal representation of the input to the network, the recurrent connection of an Elman network allows to keep "in memory" words used as input at previous positions in the sequence. Jordan RNNs [1] instead have a recursion between the output and the hidden layer. This means that a Jordan RNN can take previous predicted labels into account to predict the label at current position in a sequence. Whatever the type of RNNs, we name R the matrix of parameters at the recursion connection.

Our implementation of Jordan and Elman RNNs is standard, please see [1, 2, 21] for more details. Also we find [6] a good reading for understanding RNNs.

2.2 RNN Learning

Learning the RNNs described above consists in learning the parameters $\Theta = (E, H, O, R)$, and the biases, which we omit to keep notation lighter. We use a cross-entropy cost function between the expected label c_t and the predicted label y_t at the position t in the sequence, plus a $L2$ regularization term [22]:

$$C = -c_t \cdot log(y_t) + \frac{\lambda}{2} |\Theta|^2 \tag{1}$$

where λ is an hyper-parameter of the model. Since y_t is a probability distribution over output labels, we can also view the output of a RNN as the probability of the predicted label y_t: $P(y_t|I, \Theta)$. I is the input given to the network plus the contextual information "kept in memory" via the recurrent connection. For Elman RNN $I_{Elman} = w_{t-w}...w_t...w_{t+w}, h_{t-1}$, that is the word input context and the output of the hidden layer at the previous position in the sequence. For Jordan RNN $I_{Jordan} = w_{t-w}...w_t...w_{t+w}, y_{t-1}$, that is the same word input context as Elman RNN, and the label predicted at the previous position. We can thus associate the following decision function in RNNs to predict the label at position t in a sequence:

$$\tilde{l_t} = argmax_{j \in 1,...,|L|} P(y_t^j|I, \Theta) \tag{2}$$

Where $\tilde{l_t}$ is a particular discrete label.

[2]Given input x, using a sigmoid the output value is computed as $f(x) = \frac{1}{1+e^{-x}}$

[3]Using a set of values expressing a numerical preference $l \in L$, the higher the better, the softmax is computed as $g(l) = \frac{e^l}{\sum_{l \in L} e^l}$

We use the back-propagation and the stochastic gradient descent algorithms for learning the weights Θ. The gradient of the cross-entropy function with respect to the network output, that is the error of the network with respect to its prediction, is:

$$\frac{\delta C}{\delta y_t} = y_t - c_t \tag{3}$$

that is the difference between the output of the network and the "one-hot" representation of the gold output label c_t at position t in the sequence. Using back-propagation algorithm, the error is *back-propagated* in the network to compute the gradient at each layer, with respect to each parameter matrix (more details in [22]). Traditional stochastic grandient descent method can be improved by using momentum [22]: the gradient is accumulated over learning iterations giving some inertia to previous parameters update directions. This can avoid learning to get stuck in local optima. We use this improvement in our networks. Learning is thus performed as:

$$\Theta_{k+1} = \Theta_k + \epsilon \cdot \frac{\Lambda C}{\delta \Theta} \tag{4}$$

where ϵ is the learning rate, an hyper-parameter to be chosen. ΛC is the momentum, which is initialized to zero and then accumulates gradients along training iterations:

$$\frac{\Lambda C}{\delta \Theta} = \mu \cdot \frac{\Lambda C}{\delta \Theta} + \frac{1}{N} \frac{\delta C}{\delta \Theta} \tag{5}$$

μ is another hyper-parameter, giving more or less weight to the momentum with respect to the current gradient, N is the size of the data used at each parameters update. Since we are using Stochastic Gradient Descent, training data are split into mini-batches, parameters update is performed after processing each mini-batch, and N is thus the size of the mini-batch. Please see [22] for more details about hyper-parameters and momentum in neural networks.

2.3 Learning Tricks

In order to improve model learning, we applied some of the recommandations proposed in [22]. In particular we use:

Random training data access : That is, we shuffle training data before each training epoch.

Learning rate decay : The learning rate ϵ is initialized with a given value, and during training epochs it is decreased linearly in the number of epochs.

Another choice for learning models concern the back-propagation algorithm. Indeed, because of the recurrent nature of their architecture, in order to properly learn RNNs the back-propagation through time algorithm should be used [23]. This is supposed to allow RNNs to learn arbitrarily long past context. However, [11] has shown that RNNs for language modelling learn best with just 5 time steps in the past. This may be due to the fact that, at least in NLP tasks, the past information kept "in memory" by the network via the recurrent architecture, actually fades away after some time steps.

Since the back-propagation through time algorithm is quite more expensive than the traditional back-propagation algorithm, [6] has preferred to use explicit output context in Jordan RNNs and to learn the model with the traditional back-propagation algorithm, apparently without loosing performance.

In this work we use the same strategy. When using explicitly output labels from previous time steps as context, giving them as input to the hidden layer of a Jordan RNN, the hidden layer activity is computed as:

$$h_t = \Sigma(I_t \cdot H + [y_{t-c+1} y_{t-c+2} ... y_{t-1}] \cdot R) \tag{6}$$

where c is the size of the history of previous labels that we want to explicitly use as context to predict next label. $[\cdot]$ indicates the concatenation of vectors.

All the modifications applied to the Jordan RNN so far can be applied in a similar way to the Elman RNN.

2.4 New Recurrent Neural Network Architectures

As mentioned in the introduction, the main difference of the variants of RNN we propose in this work, with respect to traditional RNNs, is to have a recurrent connection between the output and the input layer, as opposed to the Elman and Jordan RNNs which have the recurrent connection at the hidden layer only, and from the output to the hidden layer, respectively.

This simple modification to the architecture of the network has important consequences for the model and, as we will see next, for results. Such consequences have been already mentioned in the introduction, here we describe in more details only the more important ones.

The most interesting consequence is the fact that output labels are mapped into distributional representations, like it is done more usually for input items. Indeed, the first layer of our network, is just a mapping from sparse "one-hot" representations to distributional representations. Such mapping results in very fine features encoding and in attractive syntactic and semantic properties as shown by *word2vec* and related works [13]. Such features and properties can be learnt from data in the same way as we can do for words. In the simplest case, this can be done by using sequences of output labels just like we would use sequences of input items. However it can be done in much more complex ways, using structured information when this is available, like syntactic parse trees or structured semantic labels such as named entities or entity relations. In this work we learn label embeddings using sequences of output labels associated to word sequences in annotated data. It is worth noting that the idea of using label embeddings has been introduced by [24] in the context of dependency parsing. Authors have learned embeddings, but without pre-training, of dependency labels resulting in a very effective parser. In this paper we focus on the use of several label embeddings as context, encoding thus label dependencies, which are very important in sequence labeling tasks. Also, in this work we provide fair comparison of our variants with traditional RNN architectures of Elman and Jordan. As we will see in the evaluation section, despite a local decision function (see equation 2), when label dependency modeling is crucial in the task, our variants outperform remarkably Elman and Jordan RNNs.

Using the same notation used for Elman and Jordan RNNs, we name E_w the embedding matrix for words, and E_l the embedding matrix for labels. In the same way as before,

$$I_t = [E_w(v(w_{t-w}))...E_w(v(w_t))...E_w(v(w_{t+w}))]$$

is the concatenation of vectors representing the input words at position t in a sequence, while

$$L_t = [E_l(v(y_{t-c+1}))E_l(v(y_{t-c+2}))...E_l(v(y_{t-1}))]$$

is the concatenation of vectors representing output labels predicted at previous c steps. Then the hidden layer activities are computed as:

$$h_t = \Sigma([I_t L_t] \cdot H) \tag{7}$$

Again, $[\cdot]$ means the concatenation of the two matrices and we omit biases to keep notation lighter. The remainder of the layer activities, as well as the error computation and back-propagation are computed in the same way as before.

Another important consequence of having the recursion between output and input layers is robustness. This is a direct consequence of using embeddings for output labels. Since we use several predicted labels as context at each position t (see L_t above), at least in the later stages of learning when the model is close to the final optimum, it is unlikely to have several mistakes in the same context. Even with mistakes, thanks to the properties of distributed representations [14], mistaken labels have very similar representations to the correct labels. For instance, using an example shown in [14], if we use *Paris* instead of *Rome*, this has no effect in many NLP tasks, e.g. they are both proper nouns in POS-tagging, location in named entity recognition etc. Using distributed representations for labels provides the same robustness on the output side.[4]

Jordan RNNs cannot provide in general the same robustness. This is because the recursion between output and hidden layers of Jordan RNN constrains to use either a "one-hot" representation of labels or the probability distribution given as output by the softmax. In the first case it is clear that a mistake may have more effect than when using a distributed representation, as the only value that is not zero is just at the wrong position in the representation. But also when using the probability distribution, while it may seem a much softer decision than a "one-hot" representation, we have found that most of the probability mass is "picked" on one or few labels, it doesn't provide thus much more softness than a "one-hot" representation. In any case, as we will see next, distributed representations provide a much more desirable smoothness.

The second variant of RNN that we propose combines together the caracteristics of an Elman RNN and our first variant. This allows at the same time to keep in memory the internal state of the network at previous steps and to exploit all advantages coming from the use of label embeddings as context, explained above. In the second variant, the only difference with respect to the first one is the computation of the hidden layer activities, where we use the concatenation of the c previous hidden layer states in addition to the information already used in the first variant:

[4]Sometimes in POS-tagging, models mistake verbs and nouns, which may seem to have different embeddings. However in these cases, the models make such errors because these particular verbs occur in the same context of nouns (e.g. "the sleep is important"), and so they have similar representations as nouns.

$$h_t = \Sigma([I_t L_t] \cdot H + [h_{t-c+1} h_{t-c+2} ... h_{t-1}] \cdot R) \quad (8)$$

Beyond this difference, the second variant works in the same way as the first variant.

It is important to note that different versions of RNNs exist: *forward*, *backward* and bidirectional [21]. They allow different modelling of sequence labelling tasks. In this work however we are interested in a comparison of different variants of RNNs than a comparison of different versions of RNNs. While that would provide a wider comparison, it would not change the relative ranking among the different type of RNNs. Taking also space constraints into account, we live a comparison of different RNN versions for future work.

2.5 Recurrent Neural Network Complexity

Before discussing the empirical evaluation of RNN models utilised in this work, we provide an analysis of the complexity in terms of the number of parameters involved in each model.

In Jordan RNN we have:

$$|V| \times D + ((2w+1)D + c|O|) \times |H| + |H| \times |O| \quad (9)$$

where $|V|$ is the size of the input dictionary, D is the dimensionality of the embeddings, $|H|$ and $|O|$ are the dimensionalities of the hidden and output layers, respectively. w is the size of the window of words used as context on the input side[5]. c is the size of the context of labels, which is multiplied by the dimensionality of the output label dictionary, that is $|O|$.

Using the same symbols, in an Elman RNN and in our first variant we have, respectively:

$$|V| \times D + ((2w+1)D + c|H|) \times |H| + |H| \times |O| \quad (10)$$

$$|V| \times D + |O| \times D + ((2w+1)D + cD) \times |H| + |H| \times |O| \quad (11)$$

The only difference between the Jordan and Elman RNNs is in the factors $c|O|$ and $c|H|$. Their difference in complexity depends thus on the size of the output layer (Jordan) with respect to the size of the hidden layer (Elman). Since in sequence labeling tasks the hidden layer is usually bigger, Elman RNN is more complex than Jordan RNN.

The difference between the Jordan RNN and the first variant proposed in this work is in the factors $|O| \times D$ and cD. The first is due to the use of label embeddings[6], the second is due to the use of such embeddings as input to the hidden layer. Since often D and O have sizes in the same order of magnitude, and thanks to the use of vectorized operations on matrices, we didn't found a remarkable difference in terms of training and testing time between the Jordan RNN and our first variant. This simple analysis

[5]The word input context is thus made of w words on the left and w on the right of the word at a given position t, plus the word at t itself, which gives a total of $2w + 1$ input words

[6]We use embeddings of the same size D for words and labels.

shows also that our first variant of RNN needs roughly the same number of connection at the hidden layer as a Jordan RNN. Our variant is thus only conceptually simpler, but it is architecturally equivalent to a Jordan RNN.

In contrast, the second variant we propose has the following complexity:

$$|V| \times D + |O| \times D + ((2w+1)D + cD + c|H|) \times |H| + |H| \times |O| \qquad (12)$$

The additional term $c|H|$ with respect to the first variant, is due to the use of the same recurrent connection as an Elman RNN. Even using vectorized operations for matrix calculations, the second variant is more complex and slow in both training and testing time by a factor 1.5 with respect to the other RNNs.

3 Experimental Evaluation

3.1 Corpora

We have evaluated our models on three different corpora:

The Air Travel Information System (ATIS) task [18] has been designed to automatically provide flight information in SLU systems. The semantic representation is frame based and the goal is to find the correct frame and the corresponding semantic slots. As an example, in the sentence *"I want the flights from Boston to Philadelphia today"*, the correct frame is `FLIGHT` and the words *Boston*, *Philadelphia* and *today* must be annotated with the concepts `DEPARTURE.CITY`, `ARRIVAL.CITY` and `DEPARTURE.DATE`, respectively.

ATIS is a relatively simple task dating from 1993. The training set is made of 4978 sentences taken from the "context independent" data in the ATIS-2 and ATIS-3 corpora. The test set is made of 893 sentences, taken from the ATIS-3 NOV93 and DEC94 datasets. There are not official development data provided with this corpus, we have thus taken a part of the training data at random and used as development data to optimize our model parameters. Please see [18] for more details on the ATIS corpus.

The French corpus MEDIA [19] has been created to evaluate SLU systems in providing tourist information, in particular hotel information in France. It is composed of 1250 dialogues acquired with a Wizard-of-OZ protocol where 250 speakers have applied 5 hotel reservation scenarios. The dialogues have been annotated following a rich semantic ontology. Semantic components can be combined to create complex semantic labels.[7] In addition to the rich annotation used, another difficulty is created by the annotation of coreferences. Some words cannot be correctly annotated without using previous dialog turns information. For example in the sentence *"Yes, the one at less than fifty euros per night"*, *the one* refers to an hotel previously introduced in the dialog. Statistics on training, development and test data from this corpus are shown in table 1.

Both ATIS and MEDIA can be modelled as sequence labelling tasks using the *BIO* chunking notation [25]. We use these two particular corpora, not just because they

[7] For example the label `localisation` can be combined with `city`, `relative-distance`, `general-relative-place`, `street` etc.

	training		development		test	
# sentences	12,908		1,259		3,005	
	words	concepts	words	concepts	words	concepts
# words	94,466	43,078	10,849	4,705	25,606	11,383
# vocab.	2,210	99	838	66	1,276	78
# OOV%	–	–	1.33	0.02	1.39	0.04

Table 1: Statistics on the corpus MEDIA

Section	# sentences	# tokens	# unk. tokens
FTB-train	9881	278083	
FTB-dev	1235	36508	1790 (4,9%)
FTB-test	1235	36340	1701 (4,7%)

Table 2: Statistics on the FTB corpus used for POS-tagging

provide two different settings for evaluation, but also because several different works compared on ATIS [5, 6, 7, 26]. [7] is the only work providing results on MEDIA with RNN, it also provides results obtained with CRF, allowing an interesting comparison.

The French Treebank (FTB) corpus is presented in [15]. The version of the corpus we use for POS-tagging is exactly the same used in [16]. This allows a direct comparison of our results with those obtained in that work. In contrast with [16], which obtains the best result using an external lexicon of names with associated POS, we don't use any external resource in this work ([16] provides results also without using the external lexicon). Statistics on the FTB corpus are shown in table 2.

3.2 Settings

We use the same dimensionality settings used by [5], [6] and [7] on the ATIS and MEDIA tasks, that is embeddings have 200 dimensions, hidden layer has 100 dimensions. We also use the same context size for words, that is $w = 3$, and we use $c = 6$ as labels context size in our variants and in Jordan RNN. We use the same tokenization, basically consisting of words lower-casing.

In contrast, in our models we use a "more traditional" implementation: the sigmoid activation function at the hidden layer and the $L2$ regularization. While [5], [6], [7] and [26] use the rectified linear activation function and the dropout regularization [22] [27].

For the POS-tagging task we have used 200-dimensional embeddings, 300-dimensional hidden layer, again $w = 3$ for the context on the input side, and 6 context labels on the output side. The bigger hidden layer gave better results during validation, due to the larger word dictionary in this task with respect to the others, roughly 25000 for the FTB against 2000 for MEDIA and 1300 for ATIS. In contrast to [16], which has used several features of words (prefixes, suffixes, capitalisation information etc.), we only performed a simple tokenization for reducing the size of the input dictionary: all numbers have been mapped to a conventional symbol (NUM), and nouns not corresponding to proper names and starting with a capital letter have been converted to lowercase. We preferred this simple tokenisation, instead of using rich features, because our goal in

Model	F1 measure
Words	
[6] E-RNN	93.65%
[6] J-RNN	93.77%
[26] E-RNN	**94.98%**
E-RNN	93.41%
J-RNN	93.61%
I-RNN	**93.84%**
I+E-RNN	93.74%
Classes	
[7] E-RNN	96.16%
[7] CRF	95.23%
[5] E-RNN	96.04%
[26] E-RNN	**96.24%**
[26] CRF	95.16%
E-RNN	94.73%
J-RNN	94.94%
I-RNN	**95.21%**
I+E-RNN	94.84%

Table 3: Results on the ATIS corpus. Only using words as input (upper part of the table), and using words and classes as input (lower part of the table).

this work is not obtaining the best results ever, it is to compare Jordan and Elman RNNs with our variants of RNN and show that our variants works better (at least one). Adding many features and/or building sophisticated models would make the message less clear, as results would be probably better but the improvements could be attributed to rich and sophisticated models, instead of to the model itself.

We trained all RNNs with exactly the same protocol: i) we first train neural language models to obtain word and label embeddings. This language model is like the one in [20], except it uses both words/labels in the past and in the future to predict next word/label. ii) we train all RNNs using the same embeddings trained at previous steps. We train the RNN for word embeddings for 20 epochs, the RNN for label embeddings for 10 epochs, and we train the RNNs for sequence labeling for 20 epochs. The number of epochs has been roughly optimized on development data. Also we roughly optimized on development data the learning rate and the parameter λ for regularization.

3.3 Evaluation

The evaluation of all models described in this work is shown in table 3 4 and 5, in terms of F1 measure for ATIS and MEDIA, in terms of *Accuracy* on the FTB. In all tables, our implementation of Elman and Jordan RNN are indicated as `E-RNN` and `J-RNN`. The new variants of RNN that we propose in this work are indicated as `I-RNN` and `I+E-RNN`, the latter being the combination of Elman RNN with the first variant.

The table 3 shows results on the ATIS corpus. In the higher part of the table we

Model	F1 measure
[7] E-RNN	81.94%
[7] J-RNN	83.25%
[7] CRF	**86.00%**
E-RNN	82.64%
J-RNN	83.06%
I-RNN	**84.91%**
I+E-RNN	84.58%

Table 4: Results on the MEDIA corpus

Model	F1 measure
[16] $MElt^0_{fr}$	**97.00%**
E-RNN	96.31%
J-RNN	96.31%
I-RNN	**96.42%**
I+E-RNN	96.28%

Table 5: Results of POS-tagging on the FTB corpus

show results obtained using only words as input to the RNNs. In the lower part of the table results are obtained using both words and word-classes available for this task. Such classes concern city names, airports and time expressions like date and time of departures. They allow the models to generalize from specific words triggering concepts.[8]

We would like to note that our results on the ATIS corpus are not always comparable with those published in the litterature because of the following reasons: *i)* Models published in the litterature uses a *rectified linear* activation function at the hidden layer[9] and the *dropout* regularization. Our models use the sigmoid activation function and the $L2$ regularization. These are known to be less effective for learning RNN (see [22] for explenations). *ii)* For experiments on ATIS we have used roughly 18% of the training data as development data, we thus used a smaller training set for learning our models. *iii)* The works we compare with, not always give details on how classes available for this task have been integrated into their models. *iv)* Layers dimensionality and hyper-parameters setting do not always match those of published works. In fact, to avoid running too much experiments, we have fixed our settings based on some works in the litterature, but this doesn't allow a straightforward comparison with other published works.

Despite this, the message we want to pass with this paper is still true for two reasons: *i)* Some of our results are close, or even better, than state-of-the-art. *ii)* We pro-

[8] For example the cities of *Boston* and *Philadelphia* in the example above are mapped to the class CITY-NAME. If a model has never seen *Boston* during the training phase, but it has seen at least one city name, it will possibly annotate *Boston* as a departure city thanks to some discriminative context, such as the preposition *from*.

[9] $f(x) = max(0, x)$.

vide a fair comparison with our own version of traditional Elman and Jordan RNNs. This shows that our variants of RNNs are improving models that are comparable with state-of-the-art, not simple baselines.

The results in the higher part of table 3 show that the best model on the ATIS task, with these settings, is the Elman RNN of [26]. We would like to note that it is not clear how the improvements of [26] with respect to [6] (in part due to same authors) have been obtained. Indeed, in [6] authors obtain the best result with a Jordan RNN, while in [26] an Elman RNN obtains the best performance. During our experiments, using exactly the same settings, we have found that improvements of [26] cannot be due to parameters optimization only. We thus conclude that the difference between our resutls and those in [26] are due to reasons mentioned above.

Beyond this, we note that our Elman and Jordan RNN implementations are roughly equivalent to those of [6]. Also, our first variant of RNN, `I-RNN`, obtains the second best result (93.84 in bold). Our second variant is still more effective than our Elman and Jordan RNN, but slightly worse than `I-RNN`.

The results in the lower part of the table 3 (*Classes*), obtained using both word and classes as input to RNNs, are quite better than those obtained using only words as input. Beyond this, they follow roughly the same behavior, with the difference that in this case our Jordan RNN is slightly better than our second variant. But the first variant `I-RNN` obtains the best result among our own implementation of RNNs (95.21 in bold). In this case also, we attribute differences with respect to published results to the different settings mentioned above. For comparison, in this part of the table we show also results obtained using CRF.

As a general remark, on the ATIS task, using either words or both words and classes as input for the models, we can see that results are always quite high. This is due to the fact that this task is relatively simple, in particular label dependencies can be easily modeled, as in this task there is basically no segmentation of concepts over different consecutive words (one concept corresponds to one word). In this settings, the potential of the new variant of RNNs to model label dependencies cannot be fully exploited. This limitation is proved also by results obtained by [7] and [26] using CRF. Indeed, despite the decision function of RNNs uses several pieces of information to predict the next label, it is still a local decision function. That is, RNNs don't take the whole sequence of labels into account in their decision function. In contrast, CRFs use a global decision function taking all the possible labeling of a given input sequence into account to predict the best sequence of labels. The fact that CRFs are less effective than RNN on ATIS, is a clear sign that label dependency modeling is relatively simple in this task. Despite this, the results in table 3 shows that the variant `I-RNN` obtains always the best result among our own implementation of RNNs. This prove the effectivenes of using a context of label embeddings even on a relatively simple task.

In the table 4 we show results on the corpus MEDIA. As we already mentioned, this task is more difficult because of an annotation semantically richer, but also because of the annotation of coreferences and segmentation of concepts over several words. This last aspect creates relatively long label dependencies, which cannot be taken into account by simple models. The difficulty of this task is confirmed by the absolute magnitude of results in table 4 (roughly 10 F1 measure points lower than results on the ATIS task).

As we can see in table 4, CRFs are in general much more effective than RNNs on MEDIA. This outcome can be expected as RNNs use a local decision function not able to keep long label dependencies into account. We can see also that our own implementation of Elman and Jordan RNNs are comparable with those of [7], which are state-of-the-art RNNs using a rectified linear function and dropout regularization. For this reason we can say that our Elman and Jordan RNNs are not simple baselines.

More importantly, results on the MEDIA task shows that in this particular experimental settings where taking label dependencies into account is crucial, the new variant of RNNs proposed in this paper are remarkably more effective than Elman and Jordan RNNs, both our own implementation and the state-of-the-art implementation used in [7]. We attribute this effectiveness to a better modeling of label dependencies, due in turn to the use of a label embeddings context.

In table 5 we show results obtained on the POS-tagging task of the FTB corpus. On this task we compare our own implementations of all RNNs to the state-of-the-art results obtained in [16] with the model $MElt^0_{fr}$. We would like to underline that the model $MElt^0_{fr}$, while it does not use external resources like the model obtaining the best absolute result in [16], uses several features of words that provide a clear advantage over features used in our RNNs. It can be thus expected that the model $MElt^0_{fr}$ outperforms the RNNs. The message to take from results in table 5 is that RNNs are all quite close to each other. However the `I-RNN` variant is slightly more performant, providing once more the best result among RNNs.

We would like to note that we have performed some experiments on ATIS and MEDIA without pre-training label embeddings. Results are not substantially different from those obtained with label embeddings pre-training. This may seem surprising. However on relatively small tasks (see task descriptions above), it is not rare to have similar or even better results without using pre-training. This is due to the fact that learning effective embeddings require a relatively large amount of data. Indeed, [7] shows both results obtained using embeddings pre-trained with *word2vec* and without any embeddings pre-training. Results are indeed very similar. Beyond this, the fact to have roughly the same results on a difficult task like MEDIA without label embeddings pre-training, is a clear sign that our variants of RNNs are superior to traditional RNNs because they use a context made of label embeddings, as the gain with respect to Elman and Jordan RNNs cannot be attributed to the use of pre-trained embeddings. These can encode effectively label dependencies.

It is someway surprising that `I-RNN` systematically outperforms `I+E-RNN`, the latter model integrates more information at the hidden layer and thus should be able to take advantage of both Elman and `I-RNN` characteristics. While an analysis to explain this outcome is not trivial, our interpretation is that using two recursions in a RNN gives actually redundant information to the model. Indeed, the output of the hidden layer keeps the internal state of the network, which is the internal (distributed) representation of the input n-gram of words around position t and the previous c labels. The recursion at the hidden layer allows to keep this information "in memory" and to use it at the next step $t + 1$. However using the recursion of `I-RNN` the previous c labels are also given explicitely as input to the hidden layer. This may be redundant, and constrain the model to learn an increased amount of noise. A similar idea of hybrid

RNN model has been tested in [26] without showing a clear advantage on Elman and Jordan RNNs. Despite this observations, we believe the model I+E−RNN is promising. Using a larger hidden layer or splitting the hidden layer in two parts, one for previous hidden state and one for previous label embeddings, and using an additional hidden layer to merge these two, coupled with the use of *dropout* regularization, can achieve improvements over the I−RNN model. We live this as future work.

What we can say in general on results obtained on all the presented tasks, is that RNN architectures using label embeddings context can model label dependencies in a more effective way, even when such dependencies are relatively simple (like in ATIS and FTB). The two variant of RNNs proposed in this work, but more in particular the I−RNN variant, are for this reason more effective than Elman and Jordan RNNs on sequence labeling tasks.

3.4 Evaluation as Training and Tagging Time

Since our implementations of RNNs are at this moment prototypes[10], it does not make sense to compare them to state-of-the-art in terms of training and tagging time. However it is worth providing training times at least to have an idea and to have a comparison among different RNNs.

As explained in section 2.5, our first variant I−RNN and the Jordan RNN have the same complexity. Also, since the size of the hidden layer is in the same order of magnitude as the size of the output layer (i.e. the number of labels), also Elman RNN has roughly the same complexity as the Jordan RNN. This is reflected in the training time.

The training time for label embeddings is always relatively short, as the size of the output layer, that is the number of labels, is always relatively small. This training time thus can vary from few minutes for ATIS and MEDIA, to less that 1 hour for the FTB corpus.

Training word embeddings is also very fast on ATIS and MEDIA, taking less than 30 minutes for ATIS and roughly 40 minutes for MEDIA. In contrast training word embeddings on the FTB corpus takes roughly 5 days.

Training the RNN taggers takes roughly the same time as training the word embeddings, as the size of the word dictionary is the dimension that most affects the computational complexity at softmax used to predict next label.

Concerning the second variant of RNN proposed in this work, I+E−RNN, this is slower as it is more complex in terms of number of parameters. Training I+E−RNN on ATIS and MEDIA takes roughly 45 minutes and 1 hour. In contrast training I+E−RNN on the FTB corpus takes roughly 6 days.

We didn't keep track of tagging time, however it is always negligible with respect to training time, and it is always measured in minutes.

All times provided here are with a 1.7 GHz CPU, single process.

[10] Our implementations are basically written in Octave https://www.gnu.org/software/octave/

4 Conclusions

In this paper we have studied different architectures of Recurrent Neural Networks on sequence labeling tasks. We have proposed two new variants of RNN to better model label dependencies, and we have compared these variants to the traditional architecture of Elman and Jordan RNNs. We explained which are the advantages provided by the proposed variants with respect to previous RNNs. We have evaluated all RNNs, new and traditional architectures, on three different tasks, two of Spoken Language Understanding and one of POS-tagging on the French Treebank. The results show that, despite RNNs don't improve state-of-the-art because of the use of less features, the new variant of RNN `I-RNN` always outperforms Elman and Jordan RNNs.

References

[1] Jordan, M.I.: Serial order: A parallel, distributed processing approach. In Elman, J.L., Rumelhart, D.E., eds.: Advances in Connectionist Theory: Speech. Erlbaum, Hillsdale, NJ (1989)

[2] Elman, J.L.: Finding structure in time. COGNITIVE SCIENCE **14** (1990) 179–211

[3] Collobert, R., Weston, J.: A unified architecture for natural language processing: Deep neural networks with multitask learning. In: Proceedings of the 25th International Conference on Machine Learning. ICML '08, New York, NY, USA, ACM (2008) 160–167

[4] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. J. Mach. Learn. Res. **12** (2011) 2493–2537

[5] Yao, K., Zweig, G., Hwang, M.Y., Shi, Y., Yu, D.: Recurrent neural networks for language understanding, Interspeech (2013)

[6] Mesnil, G., He, X., Deng, L., Bengio, Y.: Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In: Interspeech 2013. (2013)

[7] Vukotic, V., Raymond, C., Gravier, G.: Is it time to switch to word embedding and recurrent neural networks for spoken language understanding? In: InterSpeech, Dresde, Germany (2015)

[8] Xu, W., Auli, M., Clark, S.: CCG supertagging with a recurrent neural network. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 2: Short Papers. (2015) 250–255

[9] Lafferty, J., McCallum, A., Pereira, F.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proceedings of the Eighteenth International Conference on Machine Learning (ICML), Williamstown, MA, USA (2001) 282–289

[10] Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., Khudanpur, S.: Recurrent neural network based language model. In: INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010. (2010) 1045–1048

[11] Mikolov, T., Kombrink, S., Burget, L., Cernock, J., Khudanpur, S.: Extensions of recurrent neural network language model. In: ICASSP, IEEE (2011) 5528–5531

[12] Zennaki, O., Semmar, N., Besacier, L.: Unsupervised and lightly supervised part-of-speech tagging using recurrent neural networks. In: Proceedings of the 29th Pacific Asia Conference on Language, Information and Computation, PACLIC 29, Shanghai, China, October 30 - November 1, 2015. (2015)

[13] Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. CoRR **abs/1301.3781** (2013)

[14] Mikolov, T., Yih, W., Zweig, G.: Linguistic regularities in continuous space word representations. In: Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics. (2013) 746–751

[15] Abeillé, A., Clément, L., Toussenel, F.: Building a Treebank for French. In: Treebanks : Building and Using Parsed Corpora. Springer (2003) 165–188

[16] Denis, P., Sagot, B.: Coupling an annotated corpus and a lexicon for state-of-the-art pos tagging. Lang. Resour. Eval. **46** (2012) 721–736

[17] De Mori, R., Bechet, F., Hakkani-Tur, D., McTear, M., Riccardi, G., Tur, G.: Spoken language understanding: A survey. IEEE Signal Processing Magazine **25** (2008) 50–58

[18] Dahl, D.A., Bates, M., Brown, M., Fisher, W., Hunicke-Smith, K., Pallett, D., Pao, C., Rudnicky, A., Shriberg, E.: Expanding the scope of the atis task: The atis-3 corpus. In: Proceedings of the Workshop on Human Language Technology. HLT '94, Stroudsburg, PA, USA, Association for Computational Linguistics (1994) 43–48

[19] Bonneau-Maynard, H., Ayache, C., Bechet, F., Denis, A., Kuhn, A., Lefèvre, F., Mostefa, D., Qugnard, M., Rosset, S., Servan, S. Vilaneau, J.: Results of the french evalda-media evaluation campaign for literal understanding. In: LREC, Genoa, Italy (2006) 2054–2059

[20] Bengio, Y., Ducharme, R., Vincent, P., Jauvin, C.: A neural probabilistic language model. JOURNAL OF MACHINE LEARNING RESEARCH **3** (2003) 1137–1155

[21] Schuster, M., Paliwal, K.: Bidirectional recurrent neural networks. Trans. Sig. Proc. **45** (1997) 2673–2681

[22] Bengio, Y.: Practical recommendations for gradient-based training of deep architectures. CoRR **abs/1206.5533** (2012)

[23] Werbos, P.: Backpropagation through time: what does it do and how to do it. In: Proceedings of IEEE. Volume 78. (1990) 1550–1560

[24] Chen, D., Manning, C.: A fast and accurate dependency parser using neural networks. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, Association for Computational Linguistics (2014) 740–750

[25] Ramshaw, L., Marcus, M.: Text chunking using transformation-based learning. In: Proceedings of the 3rd Workshop on Very Large Corpora, Cambridge, MA, USA (1995) 84–94

[26] Mesnil, G., Dauphin, Y., Yao, K., Bengio, Y., Deng, L., Hakkani-Tur, D., He, X., Heck, L., Tur, G., Yu, D., Zweig, G.: Using recurrent neural networks for slot filling in spoken language understanding. IEEE/ACM Transactions on Audio, Speech, and Language Processing (2015)

[27] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research **15** (2014) 1929–1958