A Study of Neural Networks Models applied to Natural Language Inference

Victor G. Noronha[1], **João C. P. da Silva**[1]

[1]Computer Science Departament – Federal University of Rio de Janeiro
Rio de Janeiro, RJ – Brazil

Abstract. *Natural Language Inference is a task that given two sentences, a premise $\mathcal{P}$ and hypothesis $\mathcal{H}$, try to establish entailment, contradiction or neutral relationships between them. The Stanford Natural Language Inference (SNLI) is a corpus that due to its size allows that neural-network based models perform competitively on SNLI benchmark with respect to other approaches to NLI. In this work, we revisit the original neural network-based models described on the original paper and propose some variants to those models, in order to analyze the impact of such modifications on their performance. The proposed modifications were driven by the plenty of results achieved in other works, not constrained to natural language problems, that use the so-called deep learning techniques, hoping this will bring a better understanding of the elements that make up such systems.*

1. Introduction

Natural Language Inference (NLI) is a task that given two sentences, a *premise* $\mathcal{P}$ and *hypothesis* $\mathcal{H}$, try to establish one of the following relationships between them: (i) *entailment*, when $\mathcal{H}$ is true whenever $\mathcal{P}$ is true; (ii) *contradiction*, when $\mathcal{H}$ is false if $\mathcal{P}$ is true and (iii) *neutral*, when neither entailment nor contradiction can be determined. For example, the sentences $\mathcal{P}$: *A man inspects the Uniform of a figure in some East Asian Country* and $\mathcal{H}$: *The man is sleeping* contradict each other, while $\mathcal{H}$: *A soccer game with multiple males playing* is entailed from $\mathcal{P}$: *Some men are playing a sport.* [Bowman et al. 2015]

A NLI system can be used in many applications in Natural Language Processing such as [Maccartney 2009]: (i) ensure that a summarized document does not contain any sentences that could be inferred from the rest of the summary, and thus eliminating redundancies; (ii) determine a semantic equivalence between two translations, the reference translation and the candidate one, even if the words that compose the translations are quite different. If the candidate translation and the reference one entail each other, then the candidate is probably a good translation.

The Stanford Natural Language Inference (SNLI) [Bowman et al. 2015] corpus provides 570,152 labeled sentence pairs and, according to [Bowman et al. 2015], is two orders of magnitude larger than all other resources of its type, allowing that a neural-network based model performs competitively on SNLI benchmark with respect to other approaches to NLI, such as rule-based systems, simple linear classifiers.

In this work, we revisit the original neural network-based models described on [Bowman et al. 2015], that were used to produce distributed representations of sentence meaning. We propose some variants to the baseline models presented in

[Bowman et al. 2015] in order to analyze the impact of such modifications on the performance of those models. The proposed modifications were driven by the plenty of results achieved in other works, not constrained to natural language problems but also on image processing tasks, that use the so-called deep learning techniques, hoping this will bring a better understanding of the elements that make up such systems.

2. Neural Network Algorithms and Techniques

2.1. LSTM with Variational Dropout

Recurrent Neural Networks (RNNs) [Elman 1990] are used for modelling time series, using their recurrent structure to retain (past) information which allows one to discovery temporal interrelationships between distant events in a temporal sequence of events. Unfortunately, this kind of model is hard to train because it overfits quickly, due to its *vanishing gradient* and *exploding gradient* problems, described in [Bengio et al. 1994].

In order to address this problem, more complex structures were proposed, like the LSTM (*Long Short-Term Memory*) [Hochreiter and Schmidhuber 1997], that is explicitly designed to avoid gradient problems. Another possible way to address these problems is to apply *dropout* [Srivastava et al. 2014], a technique that randomly removes neurons from a specific layer, along with their connections, in order to train potentially different networks on each training step, and thus reducing overfitting. But the technique has never been applied successfully to RNNs [Gal and Ghahramani 2016]. A variant of dropout to RNNs is proposed on [Gal and Ghahramani 2016] and applies dropout on the input and recurrent connections of the RNNs, in particular on LSTM model. The standard LSTM equations are ($sigm$ is the Logistic function and $\circ$ is the element-wise product):

$$\begin{aligned}
\mathbf{i} &= sigm(\mathbf{h}_{t-1}\mathbf{U}_i + \mathbf{x}_t\mathbf{W}_i) \\
\mathbf{f} &= sigm(\mathbf{h}_{t-1}\mathbf{U}_f + \mathbf{x}_t\mathbf{W}_f) \\
\mathbf{o} &= sigm(\mathbf{h}_{t-1}\mathbf{U}_o + \mathbf{x}_t\mathbf{W}_o \\
\mathbf{g} &= tanh(\mathbf{h}_{t-1}\mathbf{U}_g + \mathbf{x}_t\mathbf{W}_g) \\
c_t &= \mathbf{f} \circ c_{t-1} + \mathbf{i} \circ \mathbf{g} \\
h_t &= \mathbf{o} \circ tanh(c_t)
\end{aligned} \tag{1}$$

where **i**, **f**, **o**, **g**, c_t, h_t, are the input gate, forget gate, output gate, input modulation gate, cell state and hidden state, respectively. $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ are the input and the previous hidden state vectors, respectively. With $\omega = \{\mathbf{W}_i, \mathbf{U}_i, \mathbf{W}_f, \mathbf{U}_f, \mathbf{W}_o, \mathbf{U}_o, \mathbf{W}_g, \mathbf{U}_g\}$ input and recurrent weight matrices for each gate, respectively, we can rewrite the gates equations on a vectorized form, explicitly including the dropout on the input and on the recurrent connections, as follows:

$$\begin{pmatrix} \mathbf{i} \\ \mathbf{f} \\ \mathbf{o} \\ \mathbf{g} \end{pmatrix} = \begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \end{pmatrix} \left(\begin{pmatrix} \mathbf{x}_t \circ \mathbf{z}_\mathbf{x} \\ \mathbf{h}_{t-1} \circ \mathbf{z}_\mathbf{h} \end{pmatrix} \cdot \omega \right)$$

(3)

where $\mathbf{z_x}$ and $\mathbf{z_h}$ are the random dropout masks applied to the input and recurrent connections.

2.2. Batch Normalization

Batch Normalization [Ioffe and Szegedy 2015] is a technique for controlling a phenomenon called *internal covariate shift*, which concerns the change of distribution of the layer's inputs, as the parameters of the previous layers change. The technique standardizes the layer inputs, reducing the effects of the changes on the previous layers. According to [Cooijmans et al. 2016], Batch Normalization takes the problem of optimize the neural network parameters to a better-conditioned situation.

The statistics (mean and variance) are calculated over a training mini-batch of size m, for each training step. The input is normalized and squashed through an affine transformation with γ and β parameters to be learned. The idea behind this linear transformation is that if there is a better distribution than the standardized normal, the normalized input will follow this distribution. The equations for a mini-batch $B = \{x_1, \ldots, x_m\}$ are:

$$\begin{aligned} \mu_B &= \frac{1}{m}\sum_{i=1}^{m} x_i \\ \sigma_B^2 &= \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \\ \hat{x}_i &= \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \\ y_i &= \gamma \hat{x}_i + \beta \end{aligned} \tag{4}$$

where y_i is taken as input for the next layer and ϵ is a constant added to the mini-batch variance for numerical stability.

2.3. Optimizers

Gradient Descent is one of the most popular algorithms to perform optimization on neural networks [Ruder 2016]. It simply changes the model parameters θ along the negative gradient direction, and this step is scaled by a learning rate η.

However, one of the weaknesses of this method is that the same learning rate η is applied to all parameters updates. For example, if the data is sparse, some feature dimensions will require larger updates on the gradient direction, whereas another features will only require a minimal update. With a fixed learning rate for all parameters, this kind of update is unfeasible. In order to address this problem, new optimization algorithms were proposed.

2.3.1. RMSProp

RMSProp [Tieleman and Hinton 2012] is a optimization algorithm quite similar to Adadelta [Zeiler 2012]. It divides the learning rate by an exponentially decaying average E of previous squared gradients g^2, at time t. The authors suggest a decay constant γ to be set 0.9, while a good default value for the learning rate η is 0.01. The parameter update rule is given by

$$\begin{aligned} E[g^2]_t &= \gamma E[g^2]_{t-1} + (1-\gamma)g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \end{aligned} \tag{5}$$

2.3.2. Adam

Adam [Kingma and Ba 2014] is another optimization method that compute per-dimension parameters updates. The name *Adam* is derived from adaptive moment estimation. According to the authors, it is designed to combine the advantages of AdaGrad [Duchi et al. 2011], which works well with sparse gradients, and RMSProp [Tieleman and Hinton 2012] which works well in on-line and non-stationary settings.

In addition to storing an exponentially decaying average of past squared gradients v_t like Adadelta and RMSProp, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum [Qian 1999]:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned} \tag{6}$$

where m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients, respectively. The authors indicated that these moment estimates are biased towards zero, which impairs the learning process. To avoid this, the authors proposed a bias correction, yielding the final Adam update rule:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^T} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^T} \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \end{aligned} \tag{7}$$

The default values for β_1, β_2 and ϵ are 0.9, 0.999, and 10^{-8}, respectively.

2.3.3. Nadam (Nesterov momentum + Adam)

Nadam [Dozat 2016] incorporates Nesterov momentum [Nesterov 1983] into Adam [Kingma and Ba 2014]. The equations are almost the same, except for now we define a momentum decay schedule given by μ_t, applied to the gradient and to first momentum (the second momentum remains the same). Finally, the first momentum is modified and we have the final Nadam update rule (the another equations remain the same):

$$\begin{aligned}
\mu_t &= \beta_1(1 - 0.5 * 0.96^{0.004t}) \\
\hat{g}_t &= \frac{g_t}{1 - \prod_{i=1}^{t} \mu_i} \\
\hat{m}_t &= \frac{m_t}{1 - \prod_{i=1}^{t+1} \mu_i} \\
\overline{m}_t &= (1 - \mu_t)\hat{g}_t + \mu_{t+1}\hat{m}_t \\
\theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\overline{m}_t
\end{aligned} \tag{8}$$

3. The original model

In our study, we focus on the original neural network model proposed by the authors [Bowman et al. 2015]. The first layer is the *sentence model layer* that produces a vector representation of each (premises and hypothesis) sentences, which is constructed without any exchange of information (word or phrase level) between these two sets of sentences. It was tested three sentence embedding models: *baseline sentence embedding model* which simply sums the embeddings of the words in each sentence; and two *sequence embedding models*: a *plain Recurrent Neural Network* (RNN) [Elman 1990] and a LSTM [Hochreiter and Schmidhuber 1997]. The word embeddings for all models are initialized with the 300-dimension reference GloVe vectors (840B token version) [Pennington et al. 2014].

The premise and hypothesis sentence vectors are mapped into a lower-dimensional space 100d (dimensions), using a additional $\tanh$ layer. Then they are concatenated and taken as input for a stack of three 200d layers, with $\tanh$ activation function. Finally a *softmax layer* provides the probability distribution of the sentence pair among the three possible classes (*entailment*, *contradiction* and *neutral*). All models are randomly initialized with standard techniques and trained using *Adadelta* [Zeiler 2012] until performance on the development step stops improving. A L2 *regularization* and *dropout* (which is a form of regularization that randomly remove layer units along with their connections) were applied to the inputs and outputs of the sentence embedding models though not to its internal connections [Bowman et al. 2015].

4. Model variants

Starting from the original model presented in section 3, we proposed 15 modifications, which are summarized on Table 1. The ? indicates that the correspondent parameters were not informed in [Bowman et al. 2015]. The main changes and techniques analyzed in our work were:

- On our experiments we used two pre-trained word vectors of different size, in order to check whether the word space dimension plays an important role on the final results:
  - A 100d version trained on the Wikipedia 2014 + Gigaword 5 corpus, with 6B tokens;
  - The 300d version trained on the Common Crawl corpus, with 840B tokens.
- On the sentence model layer, we made a few modifications:
  - We used the LSTM without dropout on its internal connections (like the original model) and without dropout on the sentence model layer′s inputs and outputs, to check the dropout importance on the sentence model layer.
  - We tested the LSTM with *Variational Dropout* [Gal and Ghahramani 2016], with probability $p = 0.5$ on both $\mathbf{z_x}$ and $\mathbf{z_h}$, and without dropout on the sentence model layer′s inputs and outputs. This kind of dropout showed excellent results on language modeling tasks, outperforming the previous state-of-the-art model [Gal and Ghahramani 2016], and on sentiment analysis tasks, so we decided to check its performance on another NLP task.
  - We tested the RNN with standard weight matrix initializations. More precisely, we initialize the input and recurrent weight matrices with Glorot Uniform [Glorot and Bengio 2010] and Orthogonal [Saxe et al. 2013] initializers, respectively. Our objective is to check the performance of a default RNN on the NLI task.
  - We tested the RNN with the *Identity matrix* as initial recurrent weight matrix [Le et al. 2015]. This initialization improved the RNN performance on various tasks [Le et al. 2015], so we decided to check its performance on NLI task.
- The activation function that maps the premise and the hypothesis into a lower-dimensional space was changed to ReLU. The technical advantages of the ReLU over $\tanh$ activation function are that it does not involve expensive-to-compute functions, and it does not saturate [Goldberg 2015]. When an activation function saturates, the gradients at this region of the function are near zero, driving the entire gradient to zero. In this situation, the network can not optimize its parameters. The ReLU activation does not have these problems.
- We applied *batch normalization* on the output of the sentence model layer before concatenating the two sentences, and then we take the concatenated sentence as input for a stack of three 200d ReLU fully-connected plus Batch Normalization (FC + BatchNorm) layers, and finally a *softmax layer*. The batch normalization was originally applied on image classification models. We decided to check its performance on a NLP task.
- We tested different optimizers, namely: RMSProp [Tieleman and Hinton 2012], Adam [Kingma and Ba 2014] and Nadam [Dozat 2016], to check whether a different optimizer could improve the results. We also tested the learning rate changes suggested on [Ioffe and Szegedy 2015], to check whether this recommendation can be successfully applied to per-dimension optimization methods on a NLP task.
- We reduce L2 penalty on the weights, following [Ioffe and Szegedy 2015], to check its impact on the final results.

Tabela 1. Model Parameters

	dim.	optimizer	dropout	L2 penalty	learnig rate
sum of words	300	Adadelta	standard	?	?
1	100	Adadelta	no DP	4e-6	1.0
2	300	RMSProp	no DP	4e-6	0.001
3	300	RMSProp	no DP	4e-3	0.001
4	300	RMSProp	no DP	4e-6	0.045
5	300	Adam	no DP	4e-6	0.001
6	300	Nadam	no DP	4e-6	0.002
7	300	Adadelta	no DP	4e-6	1.0
RNN model	300	Adadelta	standard	?	?
8	300	Nadam	no DP	4e-6	0.002
9	300	RMSProp	no DP	4e-6	0.001
10	300	Nadam	no DP	4e-6	0.002
11	300	RMSProp	no DP	4e-6	0.001
LSTM model	300	Adadelta	standard	?	?
12	300	Nadam	no DP	4e-6	0.002
13	300	RMSProp	no DP	4e-6	0.001
14	300	Nadam	variational	4e-6	0.002
15	300	RMSProp	variational	4e-6	0.001

5. Modified Models

The *baseline sentence embedding model* [Bowman et al. 2015] was trained using a pre-trained 300-dimension GloVe version, containing more than 2 million words. The word space was fine-tuned as part of training. The model parameters were initialized using standard techniques and training using Adadelta. A L2 regularization was applied (the strength coefficient was not informed by the authors), and additionally applied dropout to inputs and outputs of the sentence embedding model. The learning rate used on Adadelta was not informed by the authors.

We proposed 7 modifications. The first one was upon the original sum of words model, and the other ones were upon the first modified model:

1. We used a pre-trained 100-dimension GloVe version [Pennington et al. 2014] (we did not modify the word vectors during training), expecting that decreasing the dimension will impair model performance. We tested the Adadelta optimizer, we applied a small L2 penalty of 4e-6 upon our FC+BatchNorm layers, originally applied to image classification tasks, and we removed dropout to inputs and outputs of the sentence embedding layer [Ioffe and Szegedy 2015].
2. We changed the first modified model (1), now using a pre-trained 300-dimension GloVe version and the RMSProp optimizer, to check whether a optimizer similar to Adadelta exhibits a similar performance.
3. On [Ioffe and Szegedy 2015], the authors reduced the weight regularization on their modified Inception model, and they achieved an accuracy improvement on the ImageNet Classification task [Russakovsky et al. 2015]. On all our previous models, we are using a reduced learning rate of 4e-6, assuming that this weak

regularization, along with the Batch Normalization layers, will exhibits a good performance, outperforming the original models. To check whether this recommendation (use a weak L2 penalty) is really necessary to achieve the best result, we tested a model that disregards the recommendation, setting the L2 penalty to 4e-3 on this modified model.

4. We changed the model (2), increasing the learning rate to 0.045 (the original is 0.001), following another recommendation present in [Ioffe and Szegedy 2015].
5. According to [Ruder 2016], Adam [Kingma and Ba 2014] might be the best overall choice (his work did not consider Nadam [Dozat 2016]). In order to verify this statement, we changed model (2) optimizer to Adam, with default learning rate of 0.001.
6. According to [Dozat 2016], in most cases, like word analogy and language modelling tasks, the improvement of Nadam over Adam is fairly dramatic. We decided to check this statement on NLI task, modifying the model (2).
7. We changed the model (2), using Adadelta as optimizer. This model is the most similar to the original one, and the objective of this test is to check the dropout influence in final results.

The original model using RNN was trained under the same conditions as the original sum of words one. We proposed 4 modifications:

8. We used the 300-dimension GloVe version, without updating word vectors. We initialized the RNN input and recurrent weight matrices with Glorot Uniform [Glorot and Bengio 2010] and Orthogonal [Saxe et al. 2013] initializers, respectively (according to [Chollet et al. 2015]). We removed dropout from the model and we applied a small L2 penalty of 4e-6 upon our FC+BatchNorm Layers [Ioffe and Szegedy 2015]. We tested this model using the Nadam optimizer with *gradient clipping* [Pascanu et al. 2013] (threshold=1.0), expecting that a more complex optimizer improve the model performance.
9. As we can see later on the Results section, the RMSProp optimizer exhibited the best results, so we decided to check the performance of this optimizer on the RNN and LSTM models. On the RNN, we modified the previous model (8), changing the optimizer to RMSProp.
10. We modified the weights initialization of the model (8), following [Le et al. 2015]. We initialize the recurrent weight matrix using the identity matrix scaled by 0.01, and the input weights were initialized with a random matrix, whose entries are sampled from a Gaussian distribution with mean of zero and standard deviation of 0.001, with ReLU activation function. This modifications were originally applied on the adding problem, designed to examine the power of recurrent models in learning long-term dependencies, MNIST [LeCun et al. 1998] digits classification when the 784 pixels are presented sequentially to the recurrent net, speech recognition and language modelling tasks. We expect that the property of keeping the errors constant offered by this initialization, along with an activation function that does not saturate, improve the model performance, and outperform the previous RNN models.
11. We modified the optimizer of the previous model (10), changing to RMSProp.

The original model using LSTM was trained under the same conditions as the original sum of words one. We proposed 4 modifications:

12. We used the 300-dimension GloVe version, without updating word vectors. We removed dropout from the whole model and we applied a small L2 penalty of 4e-6 upon our FC+BatchNorm Layers [Ioffe and Szegedy 2015]. We tested this model using the Nadam optimizer, to check its performance, expecting that a more complex optimizer, along with the recommendations in [Ioffe and Szegedy 2015], improves the model performance.
13. As mentioned earlier on the RNN variants subsection, we tested the RMSProp on our LSTM models, modifying the previous model (12).
14. We modified the model (12), applying the variational dropout to it [Gal and Ghahramani 2016]. The variational dropout improved the standard LSTM performance on language modeling and sentiment analysis tasks, so we decided to apply this kind of regularization to the NLI task, to validate the hypothesis that a strong regularization on the input and recurrent connections can efficiently control overfitting, and so improve the LSTM performance [Gal and Ghahramani 2016].
15. We modified the optimizer of the previous model (14), changing to RMSProp.

All variants were implemented using Keras, [Chollet et al. 2015], a Python framework for neural networks and deep learning, and trained on a Tesla K80 GPU.

6. Results

On the Table 2, we present the results of our experiments. We observed that:

1. The results were a bit worse, what indicates that the additional information contained on the extra dimensions is important to the final accuracy of the model.
2. The performance was improved. The RMSProp update rule is very similar to the Adadelta one. Due to this, what can explain the performance improvement are the dropout absence along with the batch normalization on the fully-connected layers.
3. The results on the dev and test set were better with this stronger regularization, indicating that, on this specific task, setting a weak L2 weight regularization is not essential, in order to achieve a good performance.
4. This modification do not worked well, probably due to the learning rate increase, what indicates that the learning rate increase should be more carefully applied on optimizers that use a per-dimension update rule.
5. The Adam results were worse on this specific NLP task.
6. Nadam outperforms Adam, as expected, providing another example task where the Nesterov Momentum improves the optimizer performance.
7. The dropout absence improved the results. Another modification that can explain the results improvement is the ReLU activation function. As pointed out previously, the ReLU does not saturate, that is, the gradients are far from zero. Gradients near zero cause the vanishing gradient problem [Bengio et al. 1994], a common problem on networks with recurrent structures, which impairs the learning process. Due to this property, the ReLU helps improving the final results.
8. As expected, the model outperforms the original one.
9. RMSProp outperforms Nadam on the train and dev set, but not on the test set.
10. As pointed out by [Glorot and Bengio 2010], a proper weight initialization plays an important role on the final results, as we can see on Table 2, and on the Deep Learning field, in general.

11. One more time, RMSProp did not outperform Nadam on the test set.
12. As expected, the model outperforms the original one, as we can see on the Table 2. Due to its internal architecture, the LSTM was expected to provide the best performance, what really happened.
13. RMSProp outperforms Nadam on the train and validation set, but not on the test set, one more time.
14. As we can see on the Table 2, this model achieved the best overall performance, empirically validating the hypothesis.
15. The proposed modification improved the train and validation results, although it did not improve the accuracy of the test set.

Tabela 2. Experiment Results (on %)

Model	Train	Dev	Test
baseline sum of words model	79.3	-	75.3
1	74.10	72.48	72.23
2	82.33	76.88	76.52
3	79.69	77.58	77.50
4	74.38	32.97	32.90
5	78.86	72.47	72.48
6	79.01	76.70	76.31
7	79.52	76.81	76.48
baseline RNN model	73.1	-	72.2
8	78.95	74.32	74.58
9	82.61	74.69	74.47
10	83.26	75.86	75.83
11	83.73	74.41	74.08
baseline LSTM model	84.8	-	77.6
12	93.24	79.86	78.98
13	**93.98**	80.17	78.73
14	88.88	80.88	**80.78**
15	89.64	**81.56**	80.62

7. Conclusions

In this work, we analyzed some modifications applied to neural network-based models described on [Bowman et al. 2015] used in the NLI task. Among the proposed modifications, those which indicated a improved performance were: (i) a higher dimensional space; (ii) the absence of dropout on the sentence embedding layer and variational dropout applied to LSTM (improving the LSTM performance - exhibiting the best overall results); (iii) RNN initialization with the identity matrix outperformed the previous RNN models; (iv) activation function change on the layer that maps sentence into a lower-dimensional space; (v) batch normalization technique proved its effectiveness on a NLP task, and showing that dropout can be removed to a network with batch normalization layers, without loss performance; (vi) the use of more complex optimizers, like Adam and Nadam; (vii) more complex sentence embedding model structures really improves the model performance on the NLI task; (viii) the more complex optimizer (Nadam) exhibits the best overall performance; (ix) improve the initialization of the model: this approach is validated on our task, as we can see comparing the performance of the original RNN model, the RNN following the standard weight initializations presented on [Chollet et al. 2015] and finally the RNN initialized with the Identity matrix and Gaussian distribution.

The objective of this work was not to propose a new state of the art for the inference task, ouperforming [Wang et al. 2017], but to serve as a centralized source of information on neural networks and Deep Learning and the performance of its various algorithms and optimization techniques, to be used in future works.

8. Acknowledgments

The authors would like to thank the Laboratório de Combinatória e Computação Científica (LC3 - UFRJ), for providing Tesla K80 GPU, and so the feasibility of this study.

Referências

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. (2015). A large annotated corpus for learning natural language inference. In *Proc., EMNLP 2015, Lisbon, Portugal, Sept. 17-21, 2015*, pages 632–642.

Chollet, F. et al. (2015). Keras. `https://github.com/fchollet/keras`.

Cooijmans, T., Ballas, N., Laurent, C., and Courville, A. C. (2016). Recurrent batch normalization. *CoRR*, abs/1603.09025.

Dozat, T. (2016). Incorporating nesterov momentum into adam. In *International Conference on Learning Representations (2016).*

Duchi, J. C., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179 – 211.

Gal, Y. and Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks. In *NIPS*, pages 1019–1027.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proc. of 13th Int. Conf. on Art. Int. and Stat., AISTATS 2010, Sardinia, Italy, May 13-15, 2010*, pages 249–256.

Goldberg, Y. (2015). A primer on neural network models for natural language processing. *CoRR*, abs/1510.00726.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Le, Q. V., Jaitly, N., and Hinton, G. E. (2015). A simple way to initialize recurrent networks of rectified linear units. *CoRR*, abs/1504.00941.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Maccartney, B. (2009). *Natural Language Inference*. PhD thesis, Stanford, CA, USA. AAI3364139.

Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *ICML (3)*, volume 28 of *JMLR Proceedings*, pages 1310–1318. JMLR.org.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proc., EMNLP 2014, Oct. 25-29, 2014, Doha, Qatar*, pages 1532–1543.

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

Saxe, A. M., McClelland, J. L., and Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *CoRR*, abs/1312.6120.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *J. of Machine Learning Research*, 15(1):1929–1958.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

Wang, Z., Hamza, W., and Florian, R. (2017). Bilateral multi-perspective matching for natural language sentences. In *IJCAI 2017*.

Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701.