

# ADVANCES IN OPTIMIZING RECURRENT NETWORKS

Yoshua Bengio, Nicolas Boulanger-Lewandowski and Razvan Pascanu

U. Montreal

## ABSTRACT

After a more than decade-long period of relatively little research activity in the area of recurrent neural networks, several new developments will be reviewed here that have allowed substantial progress both in understanding and in technical solutions towards more efficient training of recurrent networks. These advances have been motivated by and related to the optimization issues surrounding deep learning. Although recurrent networks are extremely powerful in what they can in principle represent in terms of modeling sequences, their training is plagued by two aspects of the same issue regarding the learning of long-term dependencies. Experiments reported here evaluate the use of clipping gradients, spanning longer time ranges with leaky integration, advanced momentum techniques, using more powerful output probability models, and encouraging sparser gradients to help symmetry breaking and credit assignment. The experiments are performed on text and music data and show off the combined effects of these techniques in generally improving both training and test error.

**Index Terms**— Recurrent networks, deep learning, representation learning, long-term dependencies

## 1. INTRODUCTION

Machine learning algorithms for capturing statistical structure in sequential data face a fundamental problem [1, 2], called the *difficulty of learning long-term dependencies*. If the operations performed when forming a fixed-size summary of relevant past observations (for the purpose of predicting some future observations) are linear, this summary must exponentially forget past events that are further away, to maintain stability. On the other hand, if they are non-linear, then this non-linearity is composed many times, yielding a highly non-linear relationship between past events and future events. Learning such non-linear relationships turns out to be difficult, for reasons that are discussed here, along with recent proposals for reducing this difficulty.

Recurrent neural networks [3] can represent such non-linear maps ( $F$ , below) that iteratively build a relevant summary of past observations. In their simplest form, recurrent neural networks (RNNs) form a deterministic *state* variable  $h_t$  as a function of the present input observation  $x_t$  and the past value(s) of the state variable, e.g.,  $h_t = F_\theta(h_{t-1}, x_t)$ , where  $\theta$  are tunable parameters that control what will be remembered about the past sequence and what will be discarded. Depending on the type of problem at hand, a loss function  $L(h_t, y_t)$  is defined, with  $y_t$  an observed random variable at time  $t$  and  $C_t = L(h_t, y_t)$  the cost at time  $t$ . The generalization objective is to minimize the expected future cost, and the training objective involves the average of  $C_t$  over observed sequences. In principle, RNNs can be trained by gradient-based optimization procedures (using the back-propagation algorithm [3] to compute a gradient), but it was observed early on [1, 2] that capturing dependencies that span a long interval was difficult, making the task of

optimizing  $\theta$  to minimize the average of  $C_t$ 's almost impossible for some tasks when the span of the dependencies of interest increases sufficiently. More precisely, using a local numerical optimization such as stochastic gradient descent or second order methods (which gradually improve the solution), the proportion of trials (differing only from their random initialization) falling into the basin of attraction of a good enough solution quickly becomes very small as the temporal span of dependencies is increased (beyond tens or hundreds of steps, depending of the task).

These difficulties are probably responsible for the major reduction in research efforts in the area of RNNs in the 90's and 2000's. However, a revival of interest in these learning algorithms is taking place, in particular thanks to [4] and [5]. This paper studies the issues giving rise to these difficulties and discusses, reviews, and combines several techniques that have been proposed in order to improve training of RNNs, following up on a recent thesis devoted to the subject [6]. We find that these techniques generally help generalization performance as well as training performance, which suggest they help to improve the optimization of the training criterion. We also find that although these techniques can be applied in the online setting, i.e., as add-ons to stochastic gradient descent (SGD), they allow to compete with batch (or large minibatch) second-order methods such as Hessian-Free optimization, recently found to greatly help training of RNNs [4].

## 2. LEARNING LONG-TERM DEPENDENCIES AND THE OPTIMIZATION DIFFICULTY WITH DEEP LEARNING

There has been several breakthroughs in recent years in the algorithms and results obtained with so-called *deep learning* algorithms (see [7] and [8] for reviews). Deep learning algorithms discover multiple levels of representation, typically as deep neural networks or graphical models organized with many levels of representation-carrying latent variables. Very little work on deep architectures occurred before the major advances of 2006 [9, 10, 11], probably because of *optimization difficulties* due to the high level of non-linearity in deeper networks (whose output is the composition of the non-linearity at each layer). Some experiments [12] showed the presence of an extremely large number of apparent local minima of the training criterion, with no two different initializations going to the same *function* (i.e. eliminating the effect of permutations and other symmetries of parametrization giving rise to the same function). Furthermore, qualitatively different initialization (e.g., using unsupervised learning) could yield models in completely different regions of function space. An unresolved question is whether these difficulties are actually due to local minima or to ill-conditioning (which makes gradient descent converge so slowly as to appear stuck in a local minimum). Some ill-conditioning has clearly been shown to be involved, especially for the difficult problem of training deep auto-encoders, through comparisons [13] of stochastic gradient descent and Hessian-free optimization (a second order optimiza-

tion method). These optimization questions become particularly important when trying to train very large networks on very large datasets [14], where one realizes that a major challenge for deep learning is the *underfitting* issue. Of course one can trivially overfit by increasing capacity in the wrong places (e.g. in the output layer), but what we are trying to achieve is learning of more powerful representations in order to also get good generalization.

The same questions can be asked for RNNs. When the computations performed by a RNN are unfolded through time, one clearly sees a deep neural network *with shared weights* (across the 'layers', each corresponding to a different time step), and with a cost function that may depend on the output of intermediate layers. Hessian-free optimization has been successfully used to considerably extend the span of temporal dependencies that a RNN can learn [4], suggesting that ill-conditioning effects are also at play in the difficulties of training RNN.

An important aspect of these difficulties is that the gradient can be decomposed [2, 15] into terms that involve *products of Jacobians*  $\frac{\partial h_t}{\partial h_{t-1}}$  over subsequences linking an event at time  $t_1$  and one at time  $t_2$ :  $\frac{\partial h_{t_2}}{\partial h_{t_1}} = \prod_{\tau=t_1+1}^{t_2} \frac{\partial h_\tau}{\partial h_{\tau-1}}$ . As  $t_2 - t_1$  increases, the products of  $t_2 - t_1$  of these Jacobian matrices tend to either vanish (when the leading eigenvalues of  $\frac{\partial h_t}{\partial h_{t-1}}$  are less than 1) or explode (when the leading eigenvalues of  $\frac{\partial h_t}{\partial h_{t-1}}$  are greater than 1<sup>1</sup>). This is problematic because the total gradient due to a loss  $C_{t_2}$  at time  $t_2$  is a sum whose terms correspond to the effects at different time spans, which are weighted by  $\frac{\partial h_{t_2}}{\partial h_{t_1}}$  for different  $t_1$ 's:

$$\frac{\partial C_{t_2}}{\partial \theta} = \sum_{t_1 \leq t_2} \frac{\partial C_{t_2}}{\partial h_{t_2}} \frac{\partial h_{t_2}}{\partial h_{t_1}} \frac{\partial h_{t_1}}{\partial \theta(t_1)}$$

where  $\frac{\partial h_{t_1}}{\partial \theta(t_1)}$  is the derivative of  $h_{t_1}$  with respect to the instantiation of the parameters  $\theta$  at step  $t_1$ , i.e., that directly come into the computation of  $h_{t_1}$  in  $F$ . When the  $\frac{\partial h_{t_2}}{\partial h_{t_1}}$  tend to vanish for increasing  $t_2 - t_1$ , the long-term term effects become exponentially smaller in magnitude than the shorter-term ones, making it very difficult to capture them. On the other hand, when  $\frac{\partial h_{t_2}}{\partial h_{t_1}}$  "explode" (becomes large), gradient descent updates can be destructive (move to poor configuration of parameters). It is not that the gradient is wrong, it is that gradient descent makes small but finite steps  $\Delta\theta$  yielding a  $\Delta C$ , whereas the gradient measures the effect of  $\Delta C$  when  $\Delta\theta \rightarrow 0$ . A much deeper discussion of this issue can be found in [15], along with a point of view inspired by dynamical systems theory and by the geometrical aspect of the problem, having to do with the shape of the training criterion as a function of  $\theta$  near those regions of exploding gradient. In particular, it is argued that the strong non-linearity occurring where gradients explode is shaped like a cliff where not just the first but also the second derivative becomes large *in the direction orthogonal to the cliff*. Similarly, flatness of the cost function occurs simultaneously on the first and second derivatives. Hence dividing the gradient by the second derivative in each direction (i.e., pre-multiplying by the inverse of some proxy for the Hessian matrix) could in principle reduce the exploding and vanishing gradient effects, as argued in [4].

<sup>1</sup>Note that this is not a sufficient condition, but a necessary one. Further more one usually wants to operate in the regime where the leading eigenvalue is larger than 1 but the gradients do not explode.

### 3. ADVANCES IN TRAINING RECURRENT NETWORKS

#### 3.1. Clipped Gradient

To address the exploding gradient effect, [16, 15] recently proposed to *clip gradients above a given threshold*. Under the hypothesis that the explosion occurs in very small regions (the cliffs in cost function mentioned above), most of the time this will have no effect, but it will avoid aberrant parameter changes in those cliff regions, while guaranteeing that the resulting updates are still in a descent direction. The specific form of clipping used here was proposed in [15] and is discussed there at much greater length: when the norm of the gradient vector  $g$  for a given sequence is above a *threshold*, the update is done in the direction  $\text{threshold} \frac{g}{\|g\|}$ . As argued in [15], this very simple method implements a very simple form of second order optimization in the sense that the second derivative is also proportionally large in those exploding gradient regions.

#### 3.2. Spanning Longer Time Ranges with Leaky Integration

An old idea to reduce the effect of vanishing gradients is to introduce shorter paths between  $t_1$  and  $t_2$ , either via connections with longer time delays [17] or inertia (slow-changing units) in some of the hidden units [18, 19], or both [20]. Long-Short-Term Memory (LSTM) networks [21], which were shown to be able to handle much longer range dependencies, also benefit from a linearly self-connected memory unit with a near 1 self-weight which allows signals (and gradients) to propagate over long time spans.

A different interpretation to this slow-changing units is that they behave like low-pass filter and hence they can be used to focus certain units on different frequency regions of the data. The analogy can be brought one step further by introducing band-pass filter units [22] or by using domain specific knowledge to decide on what frequency bands different units should focus. [23] shows that adding low frequency information as an additional input to a recurrent network helps improving the performance of the model.

In the experiments performed here, a subset of the units were forced to change slowly by using the following "leaky integration" state-to-state map:  $h_{t,i} = \alpha_i h_{t-1,i} + (1 - \alpha_i) F_i(h_{t-1}, x_t)$ . The standard RNN corresponds to  $\alpha_i = 0$ , while here different values of  $\alpha_i$  were randomly sampled from (0.02, 0.2), allowing some units to react quickly while others are forced to change slowly, but also propagate signals and gradients further in time. Note that because  $\alpha < 1$ , the vanishing effect is still present (and gradients can still explode via  $F$ ), but the *time-scale* of the vanishing effect can be expanded.

#### 3.3. Combining Recurrent Nets with a Powerful Output Probability Model

One way to reduce the underfitting of RNNs is to introduce multiplicative interactions in the parametrization of  $F$ , as was done successfully in [4]. When the output predictions are multivariate, another approach is to capture the high-order dependencies between the output variables using a powerful output probability model such as a Restricted Boltzmann Machine (RBM) [24, 25] or a deterministic variant of it called NADE [26, 25]. In the experiments performed here, we have experimented with a NADE output model for the music data.

### 3.4. Sparser Gradients via Sparse Output Regularization and Rectified Outputs

[7] hypothesized that one reason for the difficulty in optimizing deep networks is that in ordinary neural networks gradients diffuse through the layers, diffusing credit and blame through many units, maybe making it difficult for hidden units to specialize. When the gradient on hidden units is more sparse, one could imagine that symmetries would be broken more easily and credit or blame assigned less uniformly. This is what was advocated in [27], exploiting the idea of rectifier non-linearities introduced earlier in [28], i.e., the neuron non-linearity is  $out = \max(0, in)$  instead of  $out = \tanh(in)$  or  $out = \text{sigmoid}(in)$ . This approach was very successful in recent work on deep learning for object recognition [29], beating by far the state-of-the-art on ImageNet (1000 classes). Here, we apply this deep learning idea to RNNs, using an L1 penalty on outputs of hidden units to promote sparsity of activations. The underlying hypothesis is that if the gradient is concentrated in a few paths (in the unfolded computation graph of the RNN), it will reduce the vanishing gradients effect.

### 3.5. Simplified Nesterov Momentum

Nesterov accelerated gradient (NAG) [30] is a first-order optimization method to improve stability and convergence of regular gradient descent. Recently, [6] showed that NAG could be computed by the following update rules:

$$v_t = \mu_{t-1}v_{t-1} - \epsilon_{t-1}\nabla f(\theta_{t-1} + \mu_{t-1}v_{t-1}) \quad (1)$$

$$\theta_t = \theta_{t-1} + v_t \quad (2)$$

where  $\theta_t$  are the model parameters,  $v_t$  the velocity,  $\mu_t \in [0, 1]$  the momentum (decay) coefficient and  $\epsilon_t > 0$  the learning rate at iteration  $t$ ,  $f(\theta)$  is the objective function and  $\nabla f(\theta')$  is a shorthand notation for the gradient  $\frac{\partial f(\theta)}{\partial \theta}|_{\theta=\theta'}$ . These equations have a form similar to standard momentum updates:

$$v_t = \mu_{t-1}v_{t-1} - \epsilon_{t-1}\nabla f(\theta_{t-1}) \quad (3)$$

$$\theta_t = \theta_{t-1} + v_t \quad (4)$$

$$= \theta_{t-1} + \mu_{t-1}v_{t-1} - \epsilon_{t-1}\nabla f(\theta_{t-1}) \quad (5)$$

and differ only in the evaluation point of the gradient at each iteration. This important difference, thought to counterbalance too high velocities by “peeking ahead” actual objective values in the candidate search direction, results in significantly improved RNN performance on a number of tasks.

In this section, we derive a new formulation of Nesterov momentum differing from (3) and (5) only in the linear combination coefficients of the velocity and gradient contributions at each iteration, and we offer an alternative interpretation of the method. The key departure from (1) and (2) resides in committing to the “peeked-ahead” parameters  $\Theta_{t-1} \equiv \theta_{t-1} + \mu_{t-1}v_{t-1}$  and backtracking by the same amount before each update. Our new parameters  $\Theta_t$  updates become:

$$v_t = \mu_{t-1}v_{t-1} - \epsilon_{t-1}\nabla f(\Theta_{t-1}) \quad (6)$$

$$\begin{aligned} \Theta_t &= \Theta_{t-1} - \mu_{t-1}v_{t-1} + \mu_t v_t + v_t \\ &= \Theta_{t-1} + \mu_t \mu_{t-1} v_{t-1} - (1 + \mu_t) \epsilon_{t-1} \nabla f(\Theta_{t-1}) \end{aligned} \quad (7)$$

Assuming a zero initial velocity  $v_1 = 0$  and velocity at convergence of optimization  $v_T \simeq 0$ , the parameters  $\Theta$  are a completely equivalent replacement of  $\theta$ .

Note that equation (7) is identical to *regular* momentum (5) with different linear combination coefficients. More precisely, for an equivalent velocity update (6), the velocity contribution to the new parameters  $\mu_t \mu_{t-1} < \mu_t$  is reduced relatively to the gradient contribution  $(1 + \mu_t) \epsilon_{t-1} > \epsilon_{t-1}$ . This allows storing past velocities for a longer time with a higher  $\mu$ , while actually using those velocities more conservatively during the updates. We suspect this mechanism is a crucial ingredient for good empirical performance. While the “peeking ahead” point of view suggests that a similar strategy could be adapted for regular gradient descent (misleadingly, because it would amount to a reduced learning rate  $\epsilon_t$ ), our derivation shows why it is important to choose search directions aligned with the current velocity to yield substantial improvement. The general case is also simpler to implement.

## 4. EXPERIMENTS

In the experimental section we compare vanilla SGD versus SGD plus some of the enhancements discussed above. Specifically we use the letter ‘C’ to indicate that gradient clipping is used, ‘L’ for leaky-integration units, ‘R’ if we use rectifier units with L1 penalty and ‘M’ for Nesterov momentum.

### 4.1. Music Data

We evaluate our models on the four polyphonic music datasets of varying complexity used in [25]: classical piano music (Piano-midi.de), folk tunes with chords instantiated from ABC notation (Nottingham), orchestral music (MuseData) and the four-part chorales by J.S. Bach (JSB chorales). The symbolic sequences contain high-level pitch and timing information in the form of a binary matrix, or *piano-roll*, specifying precisely which notes occur at each time-step. They form interesting benchmarks for RNNs because of their high dimensionality and the complex temporal dependencies involved at different time scales. Each dataset contains at least 7 hours of polyphonic music with an average polyphony (number of simultaneous notes) of 3.9.

Piano-rolls were prepared by aligning each time-step (88 pitch labels that cover the whole range of piano) on an integer fraction of the beat (quarter note) and transposing each sequence in a common tonality (C major/minor) to facilitate learning. Source files and preprocessed piano-rolls split in train, validation and test sets are available on the authors’ website<sup>2</sup>.

#### 4.1.1. Setup and Results

We select hyperparameters, such as the number of hidden units  $n_h$ , regularization coefficients  $\lambda_{L1}$ , the choice of non-linearity function, or the momentum schedule  $\mu_t$ , learning rate  $\epsilon_t$ , number of leaky units  $n_{leaky}$  or leaky factors  $\alpha$  according to log-likelihood on a validation set and we report the final performance on the test set for the best choice in each category. We do so by using random search [31] on the following intervals:

$$\begin{aligned} n_h &\in [100, 400] & \epsilon_t &\in [10^{-4}, 10^{-1}] \\ \mu_t &\in [10^{-3}, 0.95] & \lambda_{L1} &\in [10^{-6}, 10^{-3}] \\ n_{leaky} &\in \{0\%, 25\%, 50\%\} & \alpha &\in [0.02, 2] \end{aligned}$$

The cutoff threshold for gradient clipping is set based on the average norm of the gradient over one pass on the data, and we used 15 in this case for all music datasets. The data is split into sequences

<sup>2</sup>[www-etud.iro.umontreal.ca/~boulanni/icml2012](http://www-etud.iro.umontreal.ca/~boulanni/icml2012)

**Table 1.** Log-likelihood and expected accuracy for various RNN models in the symbolic music prediction task. The double line separates sigmoid recognition layers (above) to structured output probability models (below).

| Model              | Piano-midi.de |              |                 | Nottingham    |              |                 | MuseData      |              |                 | JSB chorales  |              |                 |
|--------------------|---------------|--------------|-----------------|---------------|--------------|-----------------|---------------|--------------|-----------------|---------------|--------------|-----------------|
|                    | LL<br>(train) | LL<br>(test) | ACC %<br>(test) | LL<br>(train) | LL<br>(test) | ACC %<br>(test) | LL<br>(train) | LL<br>(test) | ACC %<br>(test) | LL<br>(train) | LL<br>(test) | ACC %<br>(test) |
| RNN (SGD)          | -7.10         | -7.86        | 22.84           | -3.49         | -3.75        | 66.90           | -6.93         | -7.20        | 27.97           | -7.88         | -8.65        | <b>29.97</b>    |
| RNN (SGD+C)        | -7.15         | <b>-7.59</b> | 22.98           | -3.40         | -3.67        | 67.47           | -6.79         | -7.04        | 30.53           | -7.81         | -8.65        | <b>29.98</b>    |
| RNN (SGD+CL)       | -7.04         | <b>-7.57</b> | 22.97           | -3.31         | -3.57        | 67.97           | <b>-6.47</b>  | <b>-6.99</b> | <b>31.53</b>    | -7.78         | -8.63        | <b>29.98</b>    |
| RNN (SGD+CLR)      | <b>-6.40</b>  | -7.80        | <b>24.22</b>    | <b>-2.99</b>  | -3.55        | <b>70.20</b>    | -6.70         | -7.34        | 29.06           | <b>-7.67</b>  | -9.47        | <b>29.98</b>    |
| RNN (SGD+CRM)      | -6.92         | -7.73        | 23.71           | -3.20         | <b>-3.43</b> | 68.47           | -7.01         | -7.24        | 29.13           | -8.08         | -8.81        | 29.52           |
| RNN (HF)           | -7.00         | <b>-7.58</b> | 22.93           | -3.47         | -3.76        | 66.71           | -6.76         | -7.12        | 29.77           | -8.11         | <b>-8.58</b> | 29.41           |
| RNN-RBM            | N/A           | -7.09        | <b>28.92</b>    | N/A           | -2.39        | <b>75.40</b>    | N/A           | -6.01        | <b>34.02</b>    | N/A           | -6.27        | 33.12           |
| RNN-NADE (SGD)     | -7.23         | -7.48        | 20.69           | -2.85         | -2.91        | 64.95           | -6.86         | -6.74        | 24.91           | -5.46         | -5.83        | 32.11           |
| RNN-NADE (SGD+CR)  | -6.70         | -7.34        | 21.22           | -2.14         | -2.51        | 69.80           | -6.27         | -6.37        | 26.60           | -4.44         | -5.33        | 34.52           |
| RNN-NADE (SGD+CRM) | -6.61         | -7.34        | 22.12           | -2.11         | -2.49        | 69.54           | -5.99         | -6.19        | 29.62           | <b>-4.26</b>  | <b>-5.19</b> | <b>35.08</b>    |
| RNN-NADE (HF)      | <b>-6.32</b>  | <b>-7.05</b> | 23.42           | <b>-1.81</b>  | <b>-2.31</b> | 71.50           | <b>-5.20</b>  | <b>-5.60</b> | 32.60           | -4.91         | -5.56        | 32.50           |

**Table 2.** Entropy (bits per character) and perplexity for various RNN models on next character and next word prediction task.

| Model         | Penn Treebank Corpus<br>word level |                      | Penn Treebank Corpus<br>character level |                   |
|---------------|------------------------------------|----------------------|---|-------------------|
|               | perplexity<br>(train)              | perplexity<br>(test) | entropy<br>(train)                      | entropy<br>(test) |
| RNN (SGD)     | 112.11                             | 145.16               | 1.78                                    | 1.76              |
| RNN (SGD+C)   | 78.71                              | 136.63               | <b>1.40</b>                             | <b>1.44</b>       |
| RNN (SGD+CL)  | 76.70                              | 129.83               | 1.56                                    | 1.56              |
| RNN (SGD+CLR) | <b>75.45</b>                       | <b>128.35</b>        | 1.45                                    | 1.49              |

of 100 steps over which we compute the gradient. The hidden state is carried over from one sequence to another if they belong to the same song, otherwise is set to 0.

Table 1 presents log-likelihood (LL) and expected frame-level accuracy for various RNNs in the symbolic music prediction task.

Results clearly show that these enhancements allow to improve on regular SGD in almost all cases; they also make SGD competitive with HF for the sigmoid recognition layers RNNs.

## 4.2. Text Data

We use the Penn Treebank Corpus to explore both word and character prediction tasks. The data is split by using sections 0-20 as training data (5017k characters), sections 21-22 as validation (393k characters) and sections 23-24 as test data (442k characters).

For the word level prediction, we fix the dictionary to 10000 words, which we divide into 30 classes according to their frequency in text (each class holding approximately 3.3% of the total number of tokens in the training set). Such a factorization allows for faster implementation, as we are not required to evaluate the whole output layer (10000 units) which is the computational bottleneck, but only the output of the corresponding class [32].

### 4.2.1. Setup and Results

In the case of next word prediction, we compute gradients over sequences of 40 steps, where we carry the hidden state from one sequence to another. We use a small grid-search around the parameters

used to get state of the art results for this number of classes [32], i.e., with a network of 200 hidden units yielding a perplexity of 134. We explore learning rate of 0.1, 0.01, 0.001, rectifier units versus sigmoid units, cutoff threshold for the gradients of 30, 50 or none, and no leaky units versus 50 of the units being sampled from 0.2 and 0.02.

For the character level model we compute gradients over sequences of 150 steps, as we assume that longer dependencies are more crucial in this case. We use 500 hidden units and explore learning rates of 0.5, 0.1 and 0.01.

In table 2 we have entropy (bits per character) or perplexity for various RNNs on the word and character prediction tasks. Again, we observe substantial improvements in both training and test perplexity, suggesting that these techniques make optimization easier.

## 5. CONCLUSIONS

Through our experiments we provide evidence that part of the issue of training RNN is due to the rough error surface which can not be easily handled by SGD. We follow an incremental set of improvements to SGD, and show that in most cases they improve both the training and test error, and allow this enhanced SGD to compete or even improve on a second-order method which was found to work particularly well for RNNs, i.e., Hessian-Free optimization.

## 6. REFERENCES

- [1] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, T.U. München," 1991.
- [2] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE T. Neural Nets*, 1994.
- [3] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.
- [4] J. Martens and I. Sutskever, "Learning recurrent neural networks with Hessian-free optimization," in *ICML'2011*, 2011.
- [5] T. Mikolov, S. Kombrink, L. Burget, J. Cernocky, and S. Khudanpur, "Extensions of recurrent neural network language model," in *ICASSP 2011*, 2011.
- [6] I. Sutskever, *Training Recurrent Neural Networks*, Ph.D. thesis, CS Dept., U. Toronto, 2012.
- [7] Yoshua Bengio, *Learning deep architectures for AI*, Now Publishers, 2009.
- [8] Y. Bengio, A. Courville, and P. Vincent, "Unsupervised feature learning and deep learning: A review and new perspectives," Tech. Rep., arXiv:1206.5538, 2012.
- [9] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, pp. 1527–1554, 2006.
- [10] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *NIPS'2006*, 2007.
- [11] M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun, "Efficient learning of sparse representations with an energy-based model," in *NIPS'2006*, 2007.
- [12] D. Erhan, Y. Bengio, A. Courville, P. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?," *J. Machine Learning Res.*, (11) 2010.
- [13] J. Martens, "Deep learning via Hessian-free optimization," in *ICML'2010*, 2010, pp. 735–742.
- [14] Q. Le, M. Ranzato, R. Monga, M. Devin, G. Corrado, K. Chen, J. Dean, and A. Ng, "Building high-level features using large scale unsupervised learning," in *ICML'2012*, 2012.
- [15] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio, "Understanding the exploding gradient problem," Tech. Rep., Université De Montréal, 2012, arXiv:arXiv:1211.5063.
- [16] Tomas Mikolov, *Statistical Language Models based on Neural Networks*, Ph.D. thesis, Brno University of Technology, 2012.
- [17] T. Lin, B. G. Horne, P. Tino, and C. L. Giles, "Learning long-term dependencies is not as difficult with NARX recurrent neural networks," Tech. Rep. UMICAS-TR-95-78, U. Mariland, 1995.
- [18] S. ElHihi and Y. Bengio, "Hierarchical recurrent neural networks for long-term dependencies," in *NIPS'1995*, 1996.
- [19] Herbert Jaeger, Mantas Lukosevicius, Dan Popovici, and Udo Siewert, "Optimization and applications of echo state networks with leaky- integrator neurons," *Neural Networks*, vol. 20, no. 3, pp. 335–352, 2007.
- [20] I. Sutskever and G. Hinton, "Temporal kernel recurrent neural networks," *Neural Networks*, vol. 23, no. 2, (23) 2, 2010.
- [21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [22] Udo Siewert and Welf Wustlich, "Echo-state networks with band-pass neurons: Towards generic time-scale-independent reservoir structures," *Preliminary Report*, October 2007.
- [23] Tomas Mikolov and Geoffrey Zweig, "Context dependent recurrent neural network language model," Workshop on Spoken Language Technology, 2012.
- [24] I. Sutskever, G. Hinton, and G. Taylor, "The recurrent temporal restricted Boltzmann machine," in *NIPS'2008*, 2009.
- [25] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," in *ICML'2012*, 2012.
- [26] H. Larochelle and I. Murray, "The Neural Autoregressive Distribution Estimator," in *AISTATS'2011*, 2011.
- [27] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *AISTATS'2011*, 2011.
- [28] V. Nair and G.E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *ICML'2010*, 2010.
- [29] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," in *NIPS'2012*, 2012.
- [30] Yu Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence  $o(1/k^2)$ ," *Doklady AN SSSR (translated as Soviet. Math. Docl.)*, vol. 269, pp. 543–547, 1983.
- [31] James Bergstra and Yoshua Bengio, "Random search for hyper-parameter optimization," *J. Machine Learning Res.*, vol. 13, pp. 281–305, 2012.
- [32] Tomas Mikolov, Stefan Kombrink, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur, "Extensions of recurrent neural network language model," in *Proc. 2011 IEEE international conference on acoustics, speech and signal processing (ICASSP 2011)*, 2011.