Learning to deal with long-term dependencies

Razvan Pascanu, Yoshua Bengio
Université de Montréal

Recurrent Neural Networks (RNN) encode a non-linear transformation of the input history into their hidden state, making them an ideal tool for modelling complex temporal structure. Unfortunately, most training algorithms for RNNs (Back propagation Through Time (BPTT) (Rumelhart et al., 1986), Real Time Recurrent Learning (RTRL)(Williams & Zipser, 1989), Extended Kalman Filter methods (EKF) (Puskorius & Feldkamp, 1994)) are based on gradient descent and therefore suffer from the *vanishing gradient problem*(Bengio et al., 1994). Simply stated, this means that such training algorithms are not able to discover long-term dependencies, because the gradients go exponentially to 0 as they travel back in time, when the network is stable enough to encode information for a long time.

There have been several attempts to overcome the vanishing gradient problem by either replacing the gradient based learning with approaches inspired by random search or simulated annealing (Bengio et al., 1994), or by changing the connectivity pattern of the recurrent weights such that the gradients can flow back in time (ElHihi & Bengio, 1996), (Hochreiter & Schmidhuber, 1997). We propose yet another solution that looks at how to modify the gradients such that the network learns to look far into the past.

If we define $J^{(k)}$ as in equation (1), then we can write the gradients of the cost C with respect to W (the recurrent weights of the RNN) as $\frac{\partial C}{\partial W} = \sum_{t=0}^{T} \frac{\partial C(t)}{\partial W}$, where $\frac{\partial C(t)}{\partial W}$ is defined in (2), $P_t(k)$ is given by (3) and $\mathbf{x}$ is the hidden state.

$$J^{(k)} = \frac{\partial \mathbf{x}(k)}{\partial \mathbf{x}(k-1)} \tag{1}$$

$$\frac{\partial C(t)}{\partial W} = \frac{\partial C(t)}{\partial \mathbf{x}(t)} \sum_{k=0}^{t} P_t(k) \frac{\partial \mathbf{x}(t-k)}{\partial W} \tag{2}$$

$$P_t(k) = \prod_{j=t-k}^{t} J^{(j)} \tag{3}$$

The sum in (2) is over past hidden states (up to time t). The terms for which k is close to 0 represent short-term dependencies, while when k is close to t the terms contain long-term contributions. The k-th term is weighted by $P_t(k)$. Gradients can vanish if the singular values of $P_t(k)$ are < 1: $|P_t(k)\frac{\partial \mathbf{x}(t-k)}{\partial W}|$ goes exponentially fast to 0 (or ∞) as k increases. What we would prefer, in order to model long-term dependencies, is to allow all terms $\frac{\partial \mathbf{x}(t-k)}{\partial W}$ to contribute about equally to $\frac{\partial C(t)}{\partial W}$. For this we need all $P_t(k)$ to be norm preserving matrices and this can be achieved if all $J^{(k)}$ have the same property.

An intuitive way of understanding the vanishing gradient problem is to see $J^{(k)}$ as a function of the input (over which we have no control) and W. As a minimum requirement for $J^{(k)}$ to be close to a rotation matrix, we need the largest singular value of $J^{(k)}$ to be close to 1 (there needs to be at least one direction in which $J^{(k)}$ is not damping). If there are singular values of $J^{(k)}$ greater than 1, it will sometimes be expanding, possibly making the singular values of $P_t(k)$ go to infinity, and the network response unstable (e.g. not robust to noise), and learning fails. But having singular values of $J^{(k)}$ smaller then 1 means that all $J^{(k)}$ will be damping. This ensures that the training is stable, but also that for a large k, $|P_t(k)|$ is virtually 0 and all long-term dependencies are ignored.

Regularizing the recurrent neural network to achieve norm-preserving Jacobians. This vicious cycle can be broken if we add to the training criterion a regularization term, $reg(\theta)$, where θ stands for the parameters of the model, encouraging the Jacobian $J^{(k)}$ to preserve the norm of the vectors it is applied to. Defining reg is not easy because computing the Jacobians $J^{(k)}$ directly, let alone different constraints on them, is computationally prohibitive. We propose instead a regularization (see equation (4)) expressed in terms of $\frac{\partial C(k)}{\partial h(k)}$ and $\mathbf{z}(k) = \frac{\partial C}{\partial h(k)}$ (values which must already be computed for BPTT). Note that by minimizing $reg(\theta)$ as defined in (4), we enforce the ratio between the norms to be 1, which is true if $J^{(k)}$ is a rotation matrix.

$$reg(\theta) = \sum_{k=1}^{T} \log \frac{\left| \frac{\partial C}{\partial \mathbf{x}(k)} - \frac{\partial C(k)}{\partial \mathbf{x}(k)} \right|}{\left| \frac{\partial C}{\partial \mathbf{x}(k-1)} \right|} = \sum_{k=1}^{T} \log \frac{\left| \frac{\partial C}{\partial \mathbf{x}(k-1)} J^{(k)} \right|}{\left| \frac{\partial C}{\partial \mathbf{x}(k-1)} \right|} \tag{4}$$

Computing gradients through reg can be done *easily and efficiently* if an automatic differentiation tool is used. In our experiments we use Theano (`http://deeplearning.net/software/theano/`), which not only does automatic differentiation but it also optimizes the computations for optimal performance as well as it provides a transparent way of making the code GPU-friendly.

Results. The task we are addressing is that of mem-

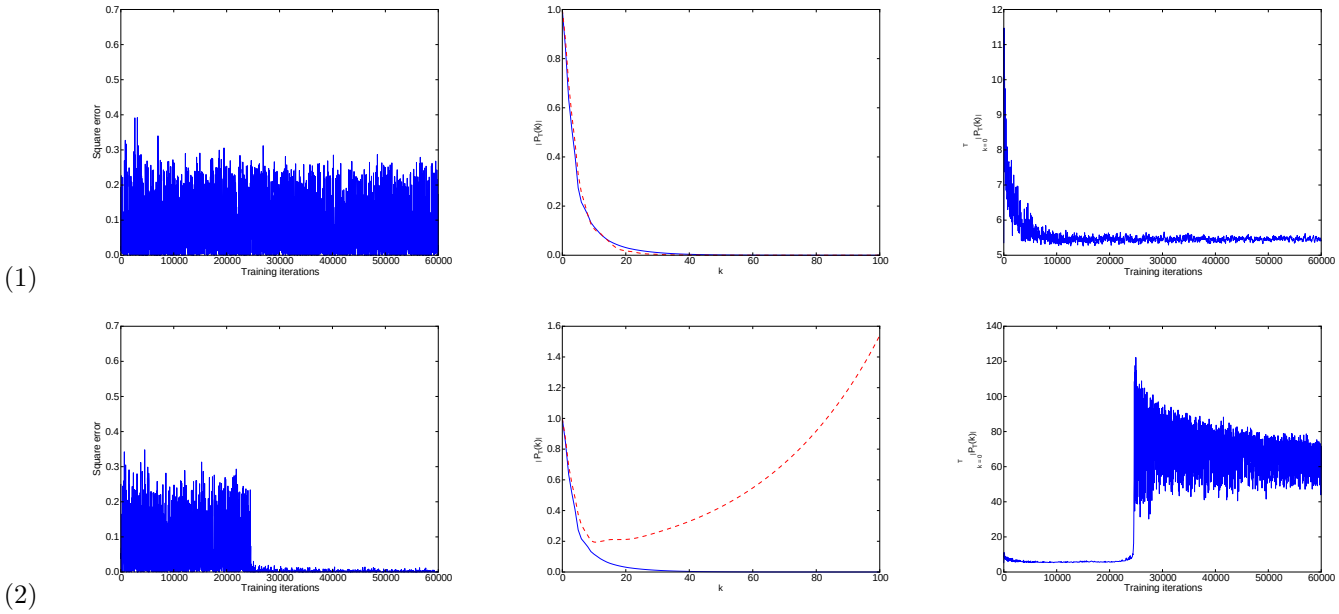

Figure 1: Results on sequences of length 100. Row (1) shows results for a RNN trained on squared error, while row (2) shows results for RNN trained on squared error plus the regularization term. Column (a) shows the evolution of squared error during training, column (b) shows the value of $|P_T(k)|$ for all values of k before training (blue continuous line) and after training (red dashed line) and column (c) the evolution of $\sum_{k=0}^{T} |P_T(k)|$ during training (showing the cumulative gradient over different durations).

orizing a number for a long period of time. The input to the RNN is a 1-dimensional time-series of 100 steps which is always 0 except for a spike of random amplitude in $(0, 1]$ at position 4. The cost is defined only on the last time step of the output in the form of the squared error between the output and the target (the amplitude of the spike). In order to be able to solve the task, the only difficulty the network has to overcome is that of the gradients being able to travel sufficiently far into the past to see the spike.

Figure 1 shows results of a RNN trained only on the squared error cost (row (1)) and on the regularized cost (row (2), the regularization term is weighted by 0.01). In both cases W starts as a damping matrix, making $|P_T(k)|$ to go exponentially to 0. Because of this the training algorithm is unable to see the spike initially. Without the regularization, the network has no incentive to increase the magnitude of $P_T(k)$, since this will not directly reduce the error, and as such the network is not able to solve the task. With the regularization, the magnitude of $P_T(k)$ slowly grows allowing the network to solve the task.

References

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

ElHihi, S., & Bengio, Y. (1996). Hierarchical recurrent neural networks for long-term dependencies. *Advances in Neural Information Processing Systems 8 (NIPS'95)* (pp. 493–499). MIT Press, Cambridge, MA.

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

Puskorius, G., & Feldkamp, L. (1994). Neurocontrol of nonlinear dynamical systems with kalman filter trained recurrent networks. *IEEE Transactions on Neural Networks*, *5*, 279–297.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). *Learning internal representations by error propagation*. Cambridge, MA, USA: MIT Press.

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.*, *1*, 270–280.