Lipschitz Recurrent Neural Networks

N. Benjamin Erichson (UC Berkeley), Omri Azencot (Ben-Gurion University), Alejandro Queiruga (Google Research), Liam Hodgkinson (UC Berkeley), Michael W. Mahoney (UC Berkeley)

ICLR

Introduction

Motivation: Many interesting problems exhibit temporal structures that can be modeled with recurrent neural networks (RNNs), including problems in robotics, vision, natural language processing, and machine learning control.

Challenge: RNNs are known to have stability issues and are difficult to train, most notably due to the **vanishing** and **exploding** gradients problem.

Solution: In this work, we address these challenges by viewing RNNs as **dynamical systems** whose temporal evolution is governed by an **abstract system of differential equations** with an external input.

Model Formulation

Based on insights from dynamical systems theory, we propose a continuous-time Lipschitz RNN that describes the hidden state's evolution with two parts: a well-understood linear component plus a Lipschitz nonlinearity. Lipschitz recurrent neural network with the functional form:

$$\begin{cases} \dot{h} &= A_{\beta_A,\gamma_A} h + \tanh(W_{\beta_W,\gamma_W} h + Ux + b)\,, & \text{(1a)} \\ y &= Dh\,, & \text{(1b)} \end{cases}$$

This particular functional form facilitates stability analysis of the long-term behavior of the recurrent unit using tools from nonlinear systems theory.

Stability Analysis of Lipschitz Recurrent Units

One of the key contributions in this work is that we prove that model (1) is globally exponentially stable under some mild conditions on A and W.

Theorem 1

Let h^* be an equilibrium point of a DE of the form $\dot{h} = Ah + \sigma(Wh + Ux + b)$ for some $x \in \mathbb{R}^p$. The point h^* is globally exponentially stable if the eigenvalues of $A^{\text{sym}} := \frac{1}{2}(A + A^T)$ are strictly negative, W is non-singular, and either

(a) $\sigma_{\min}(A^{\text{sym}}) > M\sigma_{\max}(W)$; or

(b) σ is monotone non-decreasing, $W + W^T$ is negative definite, and $A^T W + W^T A$ is positive definite.

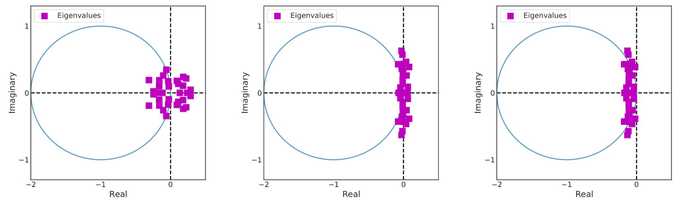
- For any initial hidden state we can guarantee that our Lipschitz unit converges to an equilibrium if it exists. Hence gradients can't explode.
- Intuitively, global exponential stability is guaranteed if the matrix A has eigenvalues with real parts sufficiently negative to counteract expanding trajectories in the nonlinearity.

Symmetric-Skew Hidden-to-Hidden Matrices

The hidden-to-hidden matrices are of the form:

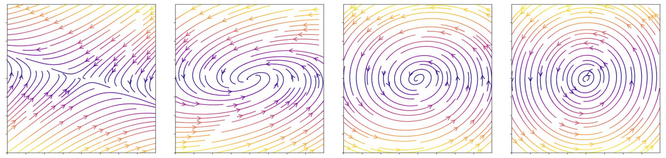
$$\begin{cases} A_{\beta_A,\gamma_A} &= (1-\beta_A)(M_A + M_A^T) + \beta_A(M_A - M_A^T) - \gamma_A I & \text{(2a)} \\ W_{\beta_W,\gamma_W} &= (1-\beta_W)(M_W + M_W^T) + \beta_W(M_W - M_W^T) - \gamma_W I, & \text{(2b)} \end{cases}$$

β controls the width of the spectrum, while increasing γ shifts the spectrum to the left, thus enforcing eigenvalues with non-positive real parts.



(a) $\beta = 0.5, \gamma = 0$ (b) $\beta = 0.75, \gamma = 0.00$ (c) $\beta = 0.75, \gamma = 0.1$

Our symmetric-skew scheme allows us to construct hidden-to-hidden matrices that exhibit dynamics with moderate decay and growth behavior.



(a) $\beta = 0.5, \gamma = 0$ (b) $\beta = 0.75, \gamma = 0.01$ (c) $\beta = 0.85, \gamma = 0.01$ (d) $\beta = 1.0, \gamma = 0$

Vector fields of hidden states that are governed by Eq. (1) trained for simple pendulum dynamics. In (a), an unstable model is shown. In (b) and (c), it can be seen that we yield models that are asymptotically stable,i.e., all trajectories are attracted by an equilibrium point. In contrast,in (d), a skew-symmetric parameterization leads to a stable model without an attracting equilibrium.

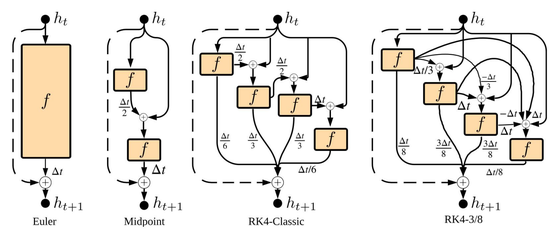
Training Continuous-time Recurrent Units

To learn the weights A, W, U and b, we discretize the continuous model using one step of a numerical integrator between sequence entries

$$h_{t+1} = h_t + \int_t^{t+\Delta t} f(h(s), s)\mathrm{d}s := h_t + \int_t^{t+\Delta t} Ah(s) + \tanh(Wh(s) + Ux(s) + b)\,\mathrm{d}s \quad (7)$$

$$\approx h_t + \Delta t \cdot \texttt{scheme}\,[f,\, h_t,\, \Delta t]\,, \quad (8)$$

where scheme represents one step of a numerical integration scheme.



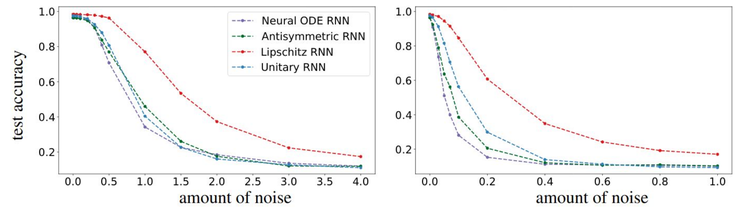
Results

Table 1: Evaluation accuracy on ordered and permuted pixel-by-pixel MNIST.

Name	ordered	permuted	N	# params
LSTM baseline by (Arjovsky et al., 2016)	97.3%	92.7%	128	≈68K
MomentumLSTM (Nguyen et al., 2020)	99.1%	94.7%	256	≈270K
Unitary RNN (Arjovsky et al., 2016)	95.1%	91.4%	512	≈9K
Full Capacity Unitary RNN (Wisdom et al., 2016)	96.9%	94.1%	512	≈270K
Soft orth. RNN (Vorontsov et al., 2017)	94.1%	91.4%	128	≈18K
Kronecker RNN (Jose et al., 2018)	96.4%	94.5%	512	≈11K
Antisymmteric RNN (Chang et al., 2019)	98.0%	95.8%	128	≈10K
Incremental RNN (Kag et al., 2020)	98.1%	95.6%	128	≈4K/8K
Exponential RNN (Lezcano-Casado & Martinez-Rubio, 2019)	98.4%	96.2%	360	≈69K
Sequential NAIS-Net (Ciccone et al., 2018)	94.3%	90.8%	128	≈18K
Lipschitz RNN using Euler (ours)	99.0%	94.2%	64	≈9K
Lipschitz RNN using RK2 (ours)	99.1%	94.2%	64	≈9K
Lipschitz RNN using Euler (ours)	**99.4%**	**96.3%**	128	≈34K
Lipschitz RNN using RK2 (ours)	99.3%	96.2%	128	≈34K

Table 2: Evaluation on TIMIT using 1 layer models. The mean squared error (MSE) is computes the distance between the predicted and actual log-magnitudes of each predicted frame in the sequence.

Name	val. MSE	test MSE	N	# params
LSTM (Helfrich et al., 2018)	13.66	12.62	158	≈200K
LSTM (Nguyen et al., 2020)	9.33	9.37	158	≈200K
MomentumLSTM (Nguyen et al., 2020)	5.86	5.87	158	≈200K
SRLSTM (Nguyen et al., 2020)	5.81	5.83	158	≈200K
Full-capacity Unitary RNN (Wisdom et al., 2016)	14.41	14.45	256	≈200K
Cayley RNN (Helfrich et al., 2018)	7.97	7.36	425	≈200K
Exponential RNN (Lezcano-Casado & Martinez-Rubio, 2019)	5.52	5.48	425	≈200K
Lipschitz RNN using Euler (ours)	2.95	2.82	256	≈198K
Lipschitz RNN using RK2 (ours)	**2.86**	**2.76**	256	≈198K



(a) White noise perturbations. (b) Salt and pepper perturbations.

Sensitivity with respect to different input perturbations.

Sketch Implementation in PyTorch

```
class LipschitzRNN(nn.Module):
    def __init__(self, input_dim, output_classes, n_units=128, eps=0.01,
                 beta=0.8, gamma=0.01, pi=0.0, gated=False, init_std=1):
        super(LipschitzRNN, self).__init__()

        self.E = nn.Linear(input_dim, n_units)
        self.D = nn.Linear(n_units, output_classes)
        self.I = torch.eye(n_units).to(get_device())
        self.C = nn.Parameter(gaussian_init_(n_units, std=init_std))
        self.B = nn.Parameter(gaussian_init_(n_units, std=init_std))

    def forward(self, x):
        T = x.shape[1]
        h = torch.zeros(x.shape[0], self.n_units).to(which_device(self))

        for i in range(T):
            z = self.E(x[:,i,:])

            I = self.gamma * self.I
            A = self.beta * (self.B - self.B.t()) + (1-self.beta) * (self.B + self.B.t()) - I
            W = self.beta * (self.C - self.C.t()) + (1-self.beta) * (self.C + self.C.t()) - I

            h = h + self.eps * (torch.matmul(h, A) + self.tanh(torch.matmul(h, W) + z))

        out = self.D(h)
        return out
```