How to Construct Deep Recurrent Neural Networks

AUTHORS: R. PASCANU, C. GULCEHRE, K. CHO, Y. BENGIO

PRESENTATION: HAROUN HABEEB

PAPER: HTTPS://ARXIV.ORG/ABS/1312.6026

This presentation

Motivation

Formal RNN paradigm

Deep RNN designs

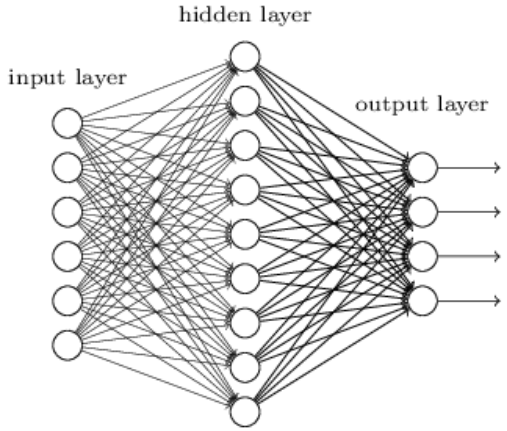
Experiments

Note on training

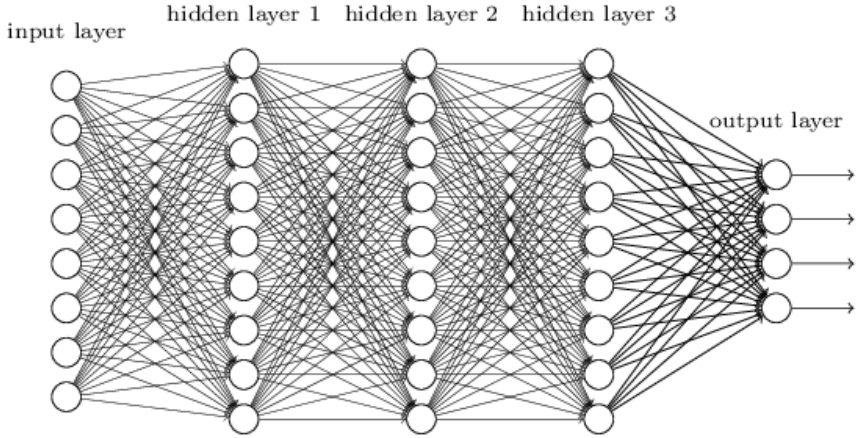
Takeaways

Motivation: Better RNNs?

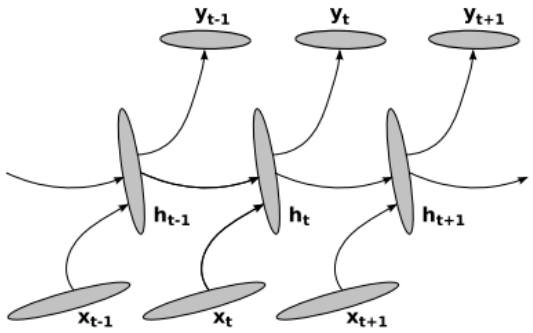
"Non-deep" feedforward neural network



Deep neural network

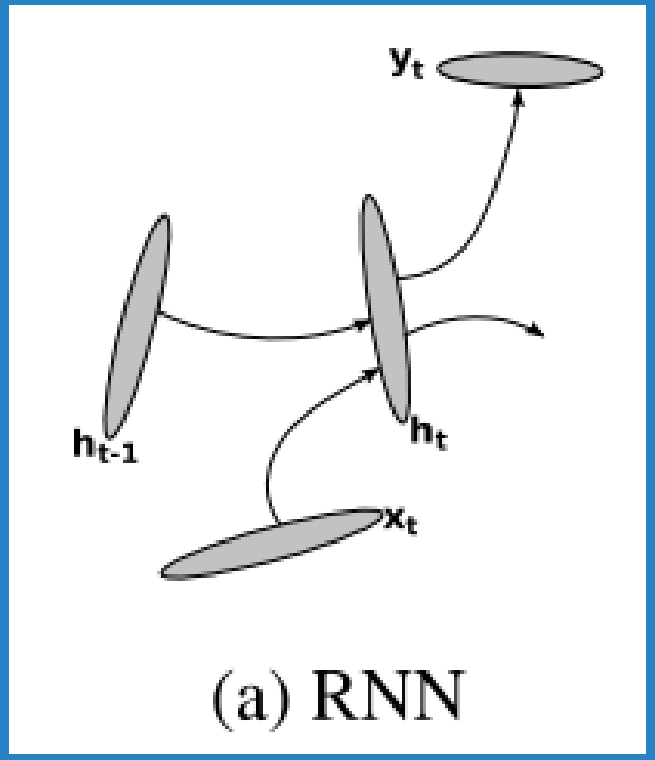


Depth makes feedforward neural networks more expressive



What about RNNS? How do you make them deep? Does depth help?

Conventional RNNs



(a) RNN

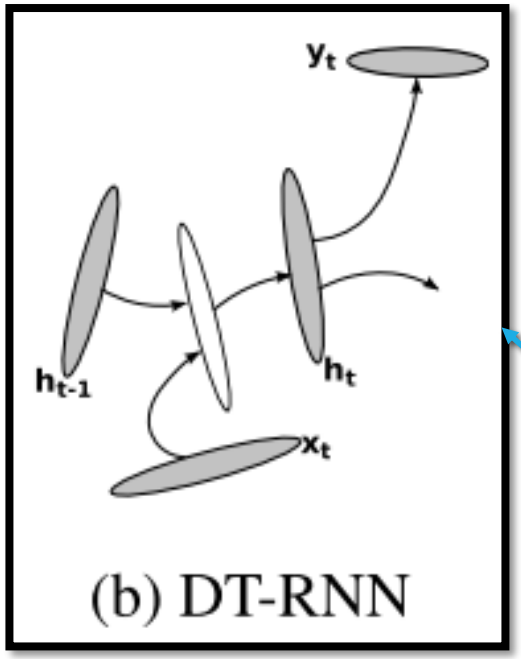
$$\boldsymbol{h}_t = f_h(\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$$
$$\boldsymbol{y}_t = f_o(\boldsymbol{h}_t)$$

Specifically:

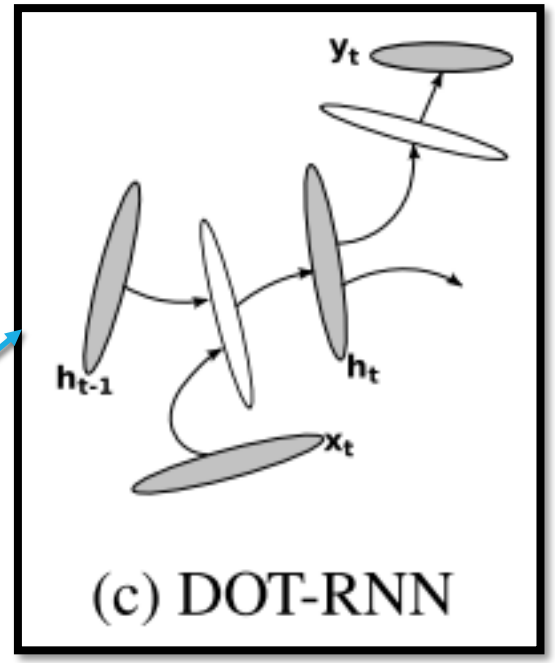
$$f_h(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}; \boldsymbol{W}, \boldsymbol{U}) = \phi_h\left(\boldsymbol{W}^T \boldsymbol{h}_{t-1} + \boldsymbol{U}^T \boldsymbol{x}_t\right)$$
$$f_o(\boldsymbol{h}_t; \boldsymbol{V}) = \phi_o(\boldsymbol{V}^T \boldsymbol{h}_t)$$

- How general is this?
- How easy is it to represent an LSTM/GRU in this form?
- What about bias terms?
- How would you make an LSTM deep?

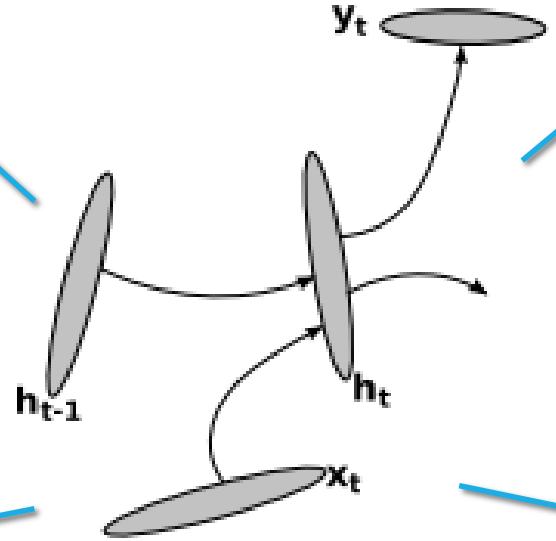
THE DEEPENING



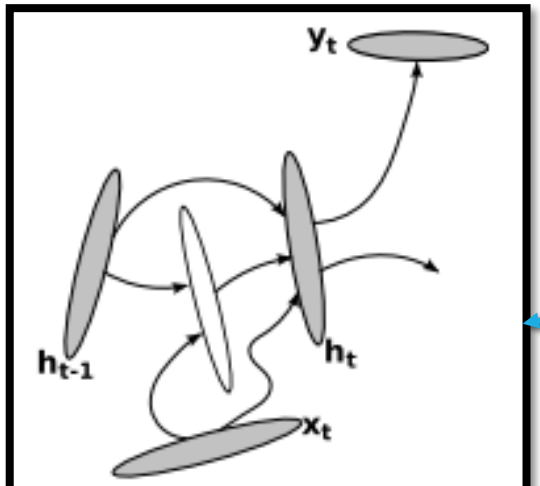
(b) DT-RNN



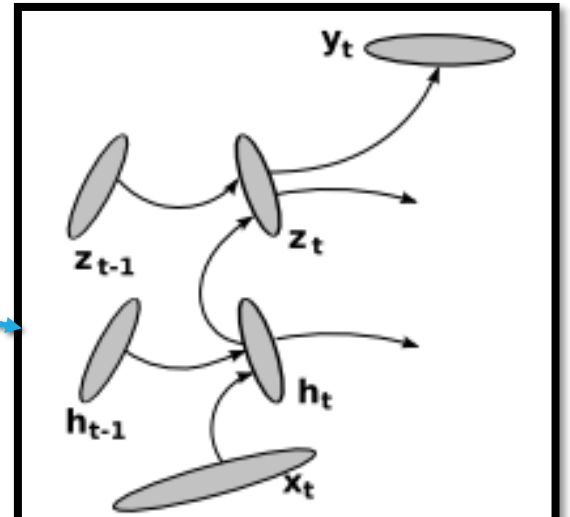
(c) DOT-RNN



(a) RNN

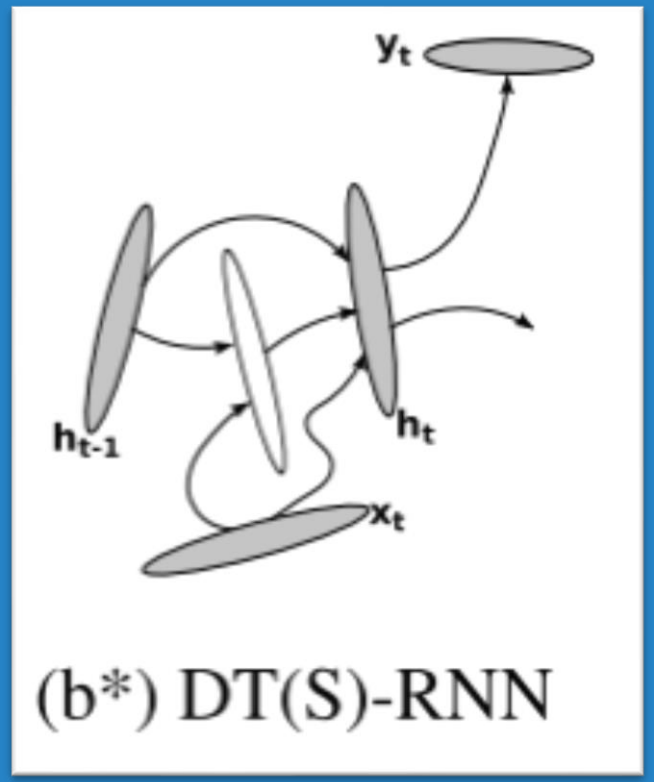


(b*) DT(S)-RNN



(d) Stacked RNN

DT(S)-RNN

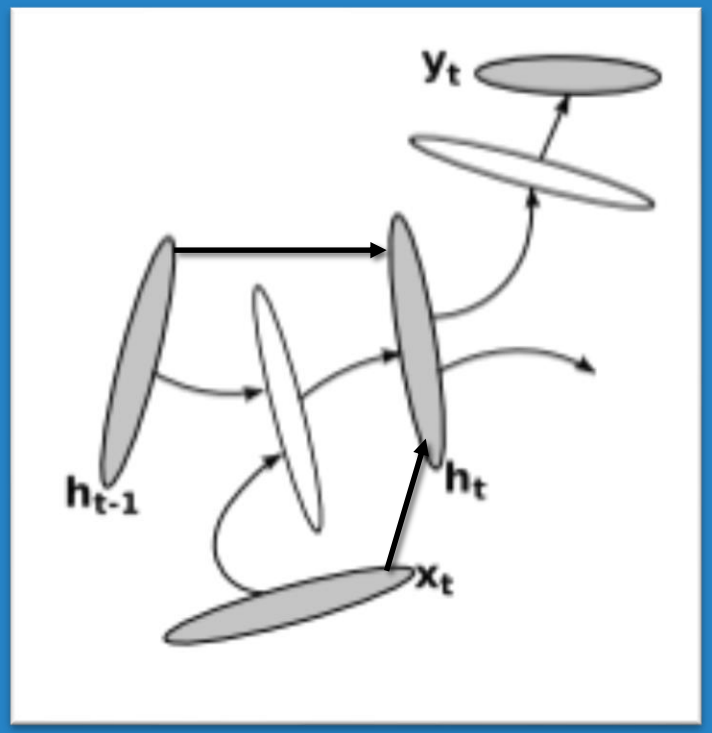


$$\boldsymbol{y}_t = f_o(\boldsymbol{h}_t)$$
$$\boldsymbol{h}_t = f_h(g(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}), \boldsymbol{x}_t, \boldsymbol{h}_{t-1})$$

Specifically:

$$y_t = \psi(W h_t)$$
$$h_t = \phi_L($$
$$V_L^T \phi_{L-1}\left(\ldots V_2^T \phi_1(V_1^T h_{t-1} + U x_t)\right)$$
$$+\bar{W}^T h_{t-1}$$
$$+\bar{U}^T x_t)$$

DOT(S)-RNN



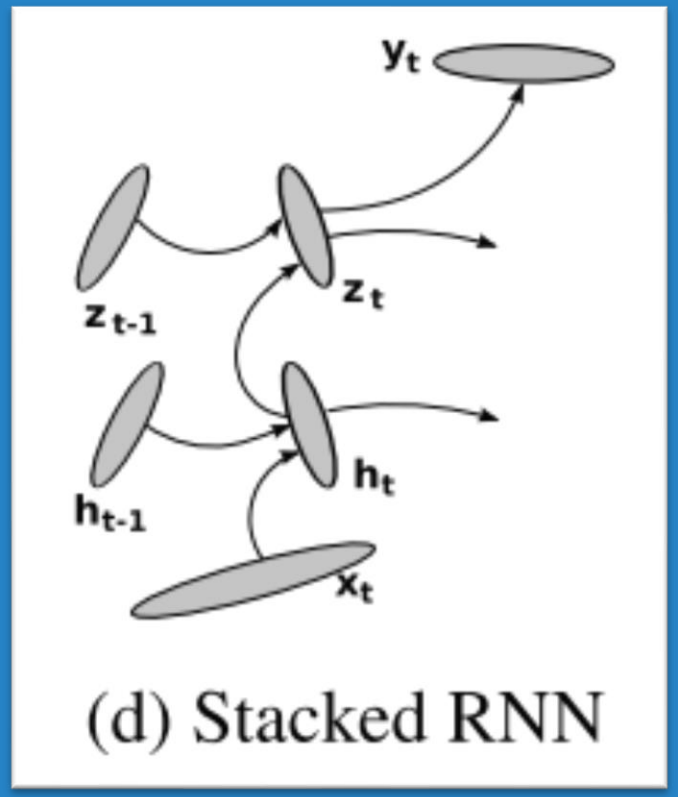
$$\boldsymbol{y}_t = f_o(\boldsymbol{h}_t)$$
$$\boldsymbol{h}_t = f_h(g(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}), \boldsymbol{x}_t, \boldsymbol{h}_{t-1})$$

Specifically:

$$y_t = \psi_0(W_L^T \psi_L(\dots W_1^T \psi_1(W^T h_t))$$

$$h_t = \phi_L(V_L^T \phi_{L-1}\left(\dots V_2^T \phi_1(V_1^T h_{t-1} + U x_t)\right) + \overline{W}^T h_{t-1} + \overline{U}^T x_t)$$

sRNN



(d) Stacked RNN

$$\boldsymbol{h}_t^0 = \boldsymbol{f}_h^0(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}^0)$$
$$\forall l: \boldsymbol{h}_t^{(l)} = f_h^{(l)}(\boldsymbol{h}_t^{l-1}, \boldsymbol{h}_{t-1}^l)$$
$$\boldsymbol{y}_t = f_o\left(\boldsymbol{h}_t^{(L)}\right)$$

Specifically:

$$\boldsymbol{y}_t = \psi\left(W^T \boldsymbol{h}_t^{(L)}\right)$$
$$h_t^{(0)} = \phi^{(0)}\left(U_0^T \boldsymbol{x}_t + W_0^T \boldsymbol{h}_{t-1}^{(0)}\right)$$
$$\forall l: \boldsymbol{h}_t^{(l)} = \phi^{(l)}\left(U_l^T \boldsymbol{h}_t^{l-1} + W_l^T \boldsymbol{h}_{t-1}^{(l)}\right)$$

Experiment 0: Parameter count

			RNN	DT(S)-RNN	DOT(S)-RNN	sRNN 2 layers
Music	Notthingam	# units	600	400,400	400,400,400	400
		# parameters	465K	585K	745K	550K
	JSB Chorales	# units	200	400,400	400,400,400	400
		# parameters	75K	585K	745K	550K
	MuseData	# units	600	400,400	400,400,400	600
		# parameters	465K	585K	745K	1185K
Language	Char-level	# units	600	400,400	400,400,600	400
		# parameters	420K	540K	790K	520K
	Word-level	# units	200	200,200	200,200,200	400
		# parameters	4.04M	6.12M	6.16M	8.48M

Table 1: The sizes of the trained models. We provide the number of hidden units as well as the total number of parameters. For DT(S)-RNN, the two numbers provided for the number of units mean the size of the hidden state and that of the intermediate layer, respectively. For DOT(S)-RNN, the three numbers are the size of the hidden state, that of the intermediate layer between the consecutive hidden states and that of the intermediate layer between the hidden state and the output layer. For sRNN, the number corresponds to the size of the hidden state at each level

Food for thought: Not clear which one has most number of parameters – sRNN or DOT(S)-RNN.

Experiment 1: Polyphonic Music Prediction

Task:

```
f|"A"ecc c2f|"A"ecc c2f|"A"ecc c2f|"Bm"BcB "E7"B2f|
"A"ecc c2f|"A"ecc c2c/2d/2|"D"efe "E7"dcB| [1"A"Ace a2:|
 [2"A"Ace ag=g||\
```

Sequence of musical notes

Next note(s)

	RNN	DT(S)-RNN	DOT(S)-RNN	sRNN	DOT(S)-RNN*
Notthingam	3.225	3.206	3.215	3.258	2.95
JSB Chorales	8.338	8.278	8.437	8.367	7.92
MuseData	6.990	6.988	6.973	6.954	6.59

Table 2: The performances of the four types of RNNs on the polyphonic music prediction. The numbers represent negative log-probabilities on test sequences. (*) We obtained these results using DOT(S)-RNN with L_p units in the deep transition, maxout units in the deep output function and dropout (Gulcehre *et al.*, 2013).

Food for thought:
Sure, depth helps, but * helps a lot more in this case. What about RNN* and other models with *?

Experiment 2: Language Modelling

Task (LM on PTB) : **Sequence of characters/words** → **Next character/word**

	RNN	DT(S)-RNN	DOT(S)-RNN	sRNN	$*$	$\star$
Character-Level	1.414	1.409	**1.386**	1.412	1.41[1]	1.24[3]
Word-Level	117.7	112.0	**107.5**	110.0	123[2]	117[3]

Table 3: The performances of the four types of RNNs on the tasks of language modeling. The numbers represent bit-per-character and perplexity computed on test sequence, respectively, for the character-level and word-level modeling tasks. $*$ The previous/current state-of-the-art results obtained with shallow RNNs. $\star$ The previous/current state-of-the-art results obtained with RNNs having long-short term memory units.

Food for thought:
Deepening LSTMs? Stack them or DOT(S) them?

Note on training

- Training RNNs can be hard because of vanishing/exploding gradients.
- Authors did a bunch of things:
  - Clipped gradients, threshold = 1
  - **Sparse** weight matrices ($\left|\left|W\right|\right|_0 = 20$)
  - Normalized weight matrices $\Rightarrow \max_{i,j} W_{i,j} = 1$
  - Add gaussian noise to gradients
  - Used dropout, maxout, L_p units

Takeaways

- Plain, shallow RNNs are not great.
- DOT-RNNs do well. Following should be deep networks
  - $y = f(h, x)$
  - $h_t = f(g(x_t, h_{t-1}), x_t, h_{t-1})$ - both f and g
- Training can be really hard.
- Thresholding gradients, Dropout, maxout units are helpful/needed
- LSTMs are good

Questions?