Long Short-Term Memory

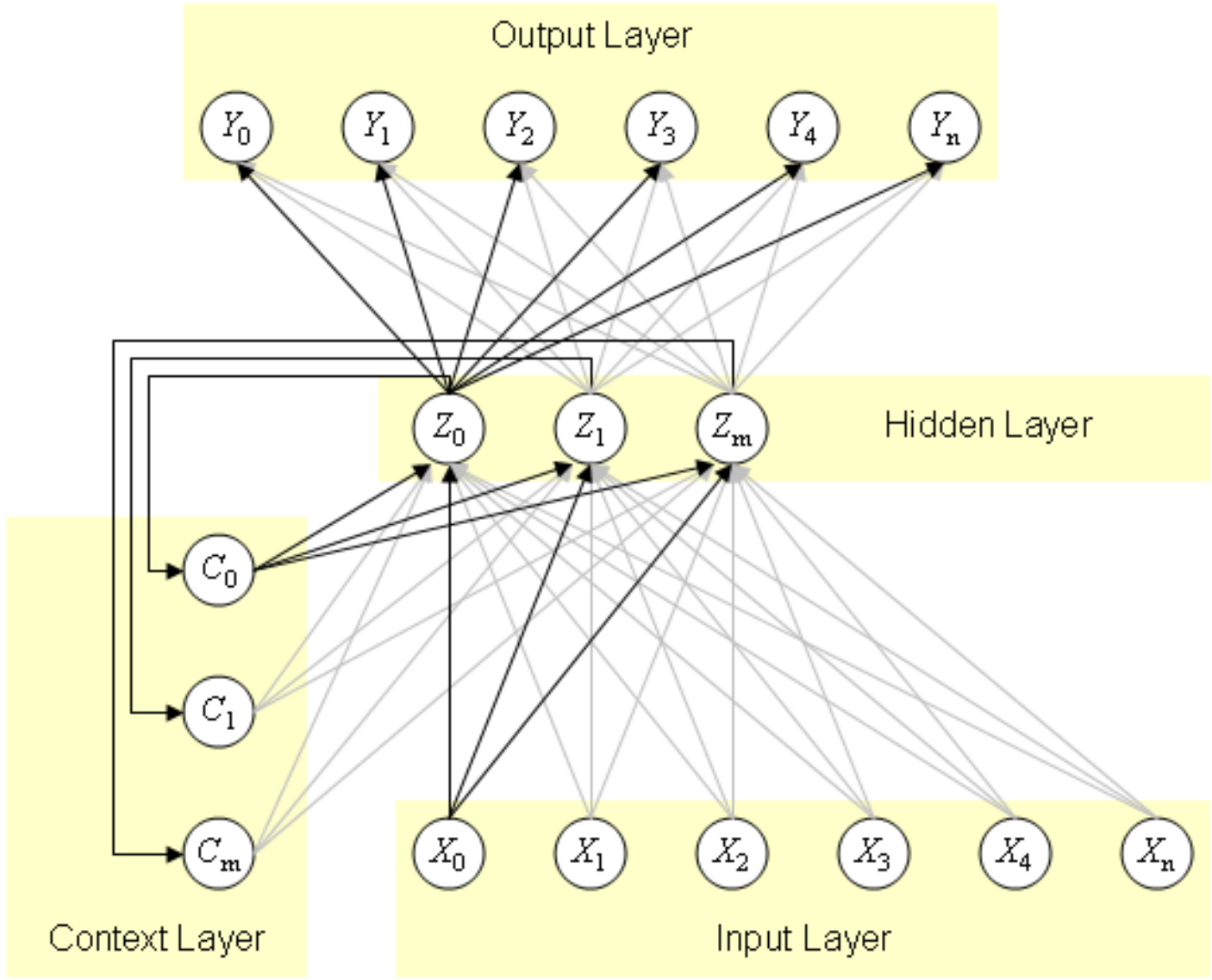
Sepp Hochreiter & Jürgen Schmidhuber (1997)

(Presented by Kelly Zhang)

Learning Sequences

- Difficulties:
  - Variable length inputs
  - Long time-lags (memory)
- Approach:
  - Recurrence
- New difficulties:
  - Vanishing error gradients through depth / time

Elman Nets (Vanilla RNN)

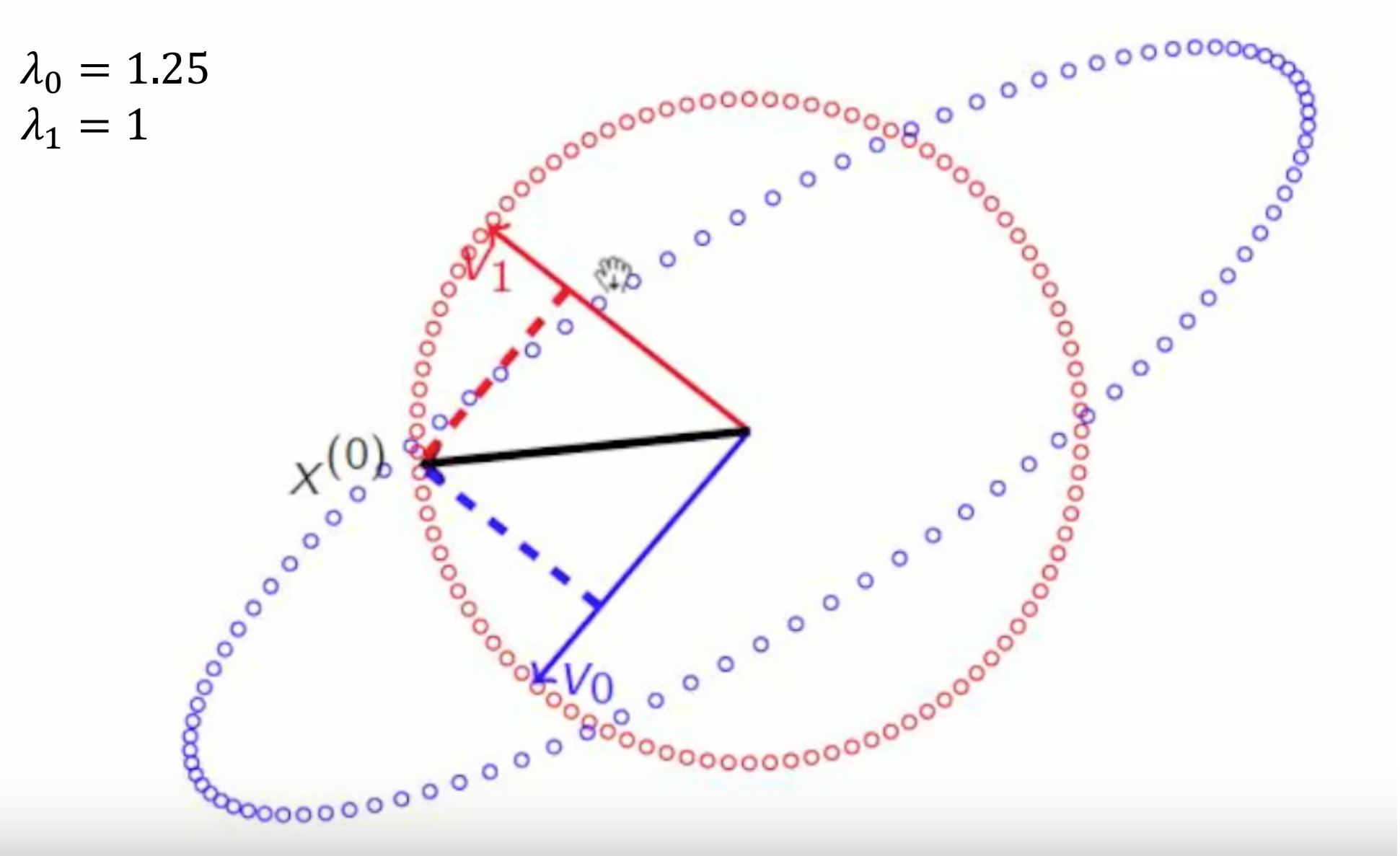


Recurrence at its Limits: Linear Case

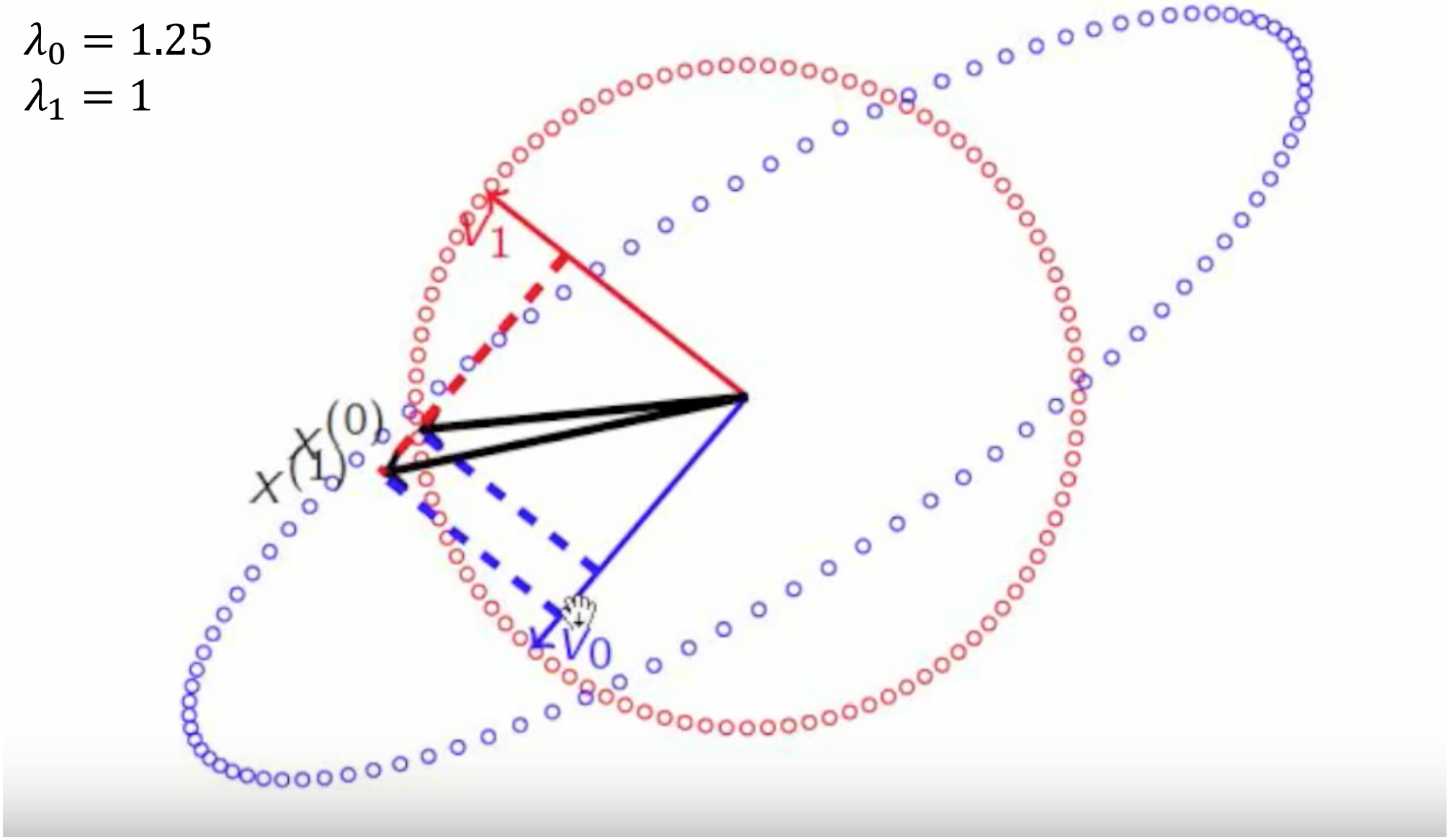
- What happens if you repeatedly apply linear transformation W?
  - x := Wx; repeat
  - y = WWW…WWWx
- Gradient:
  - $\frac{\partial y}{\partial x} = W^T W^T W^T \ldots W^T W^T W^T$
- x's direction converges to W^T's eigenvector with largest eigenvalue*
  - If W's largest eigenvalue…
    - < 1 → x's magnitude will converge to 0
    - > 1 → x's magnitude will go to infinity
    - = 1 → x's magnitude will be stable (> 0* and non-increasing)

*Assuming x's component in the direction of the eigenvector of the largest eigenvalue is non-zero

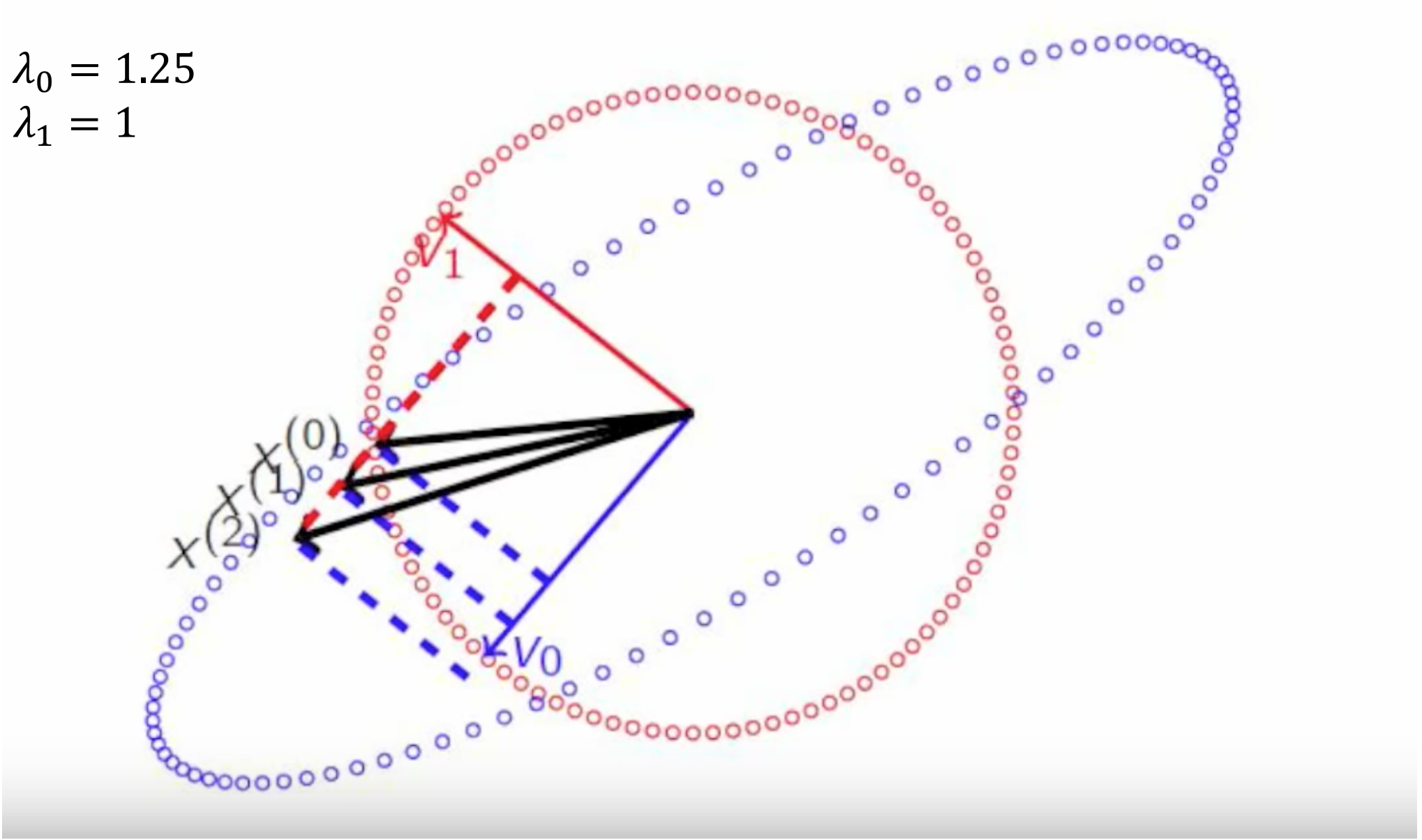
$\lambda_0 = 1.25$
$\lambda_1 = 1$



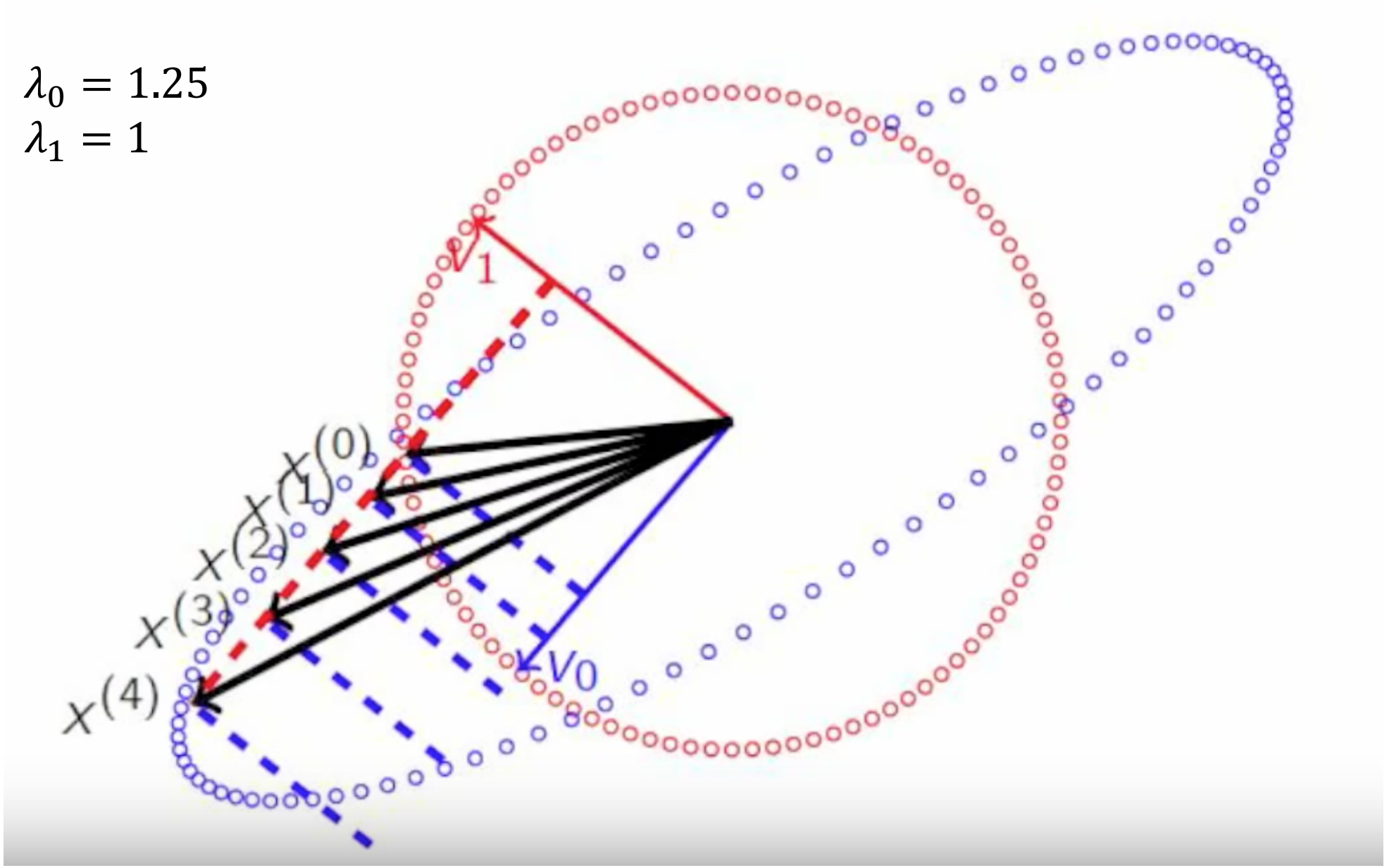
$\lambda_0 = 1.25$
$\lambda_1 = 1$



$\lambda_0 = 1.25$
$\lambda_1 = 1$

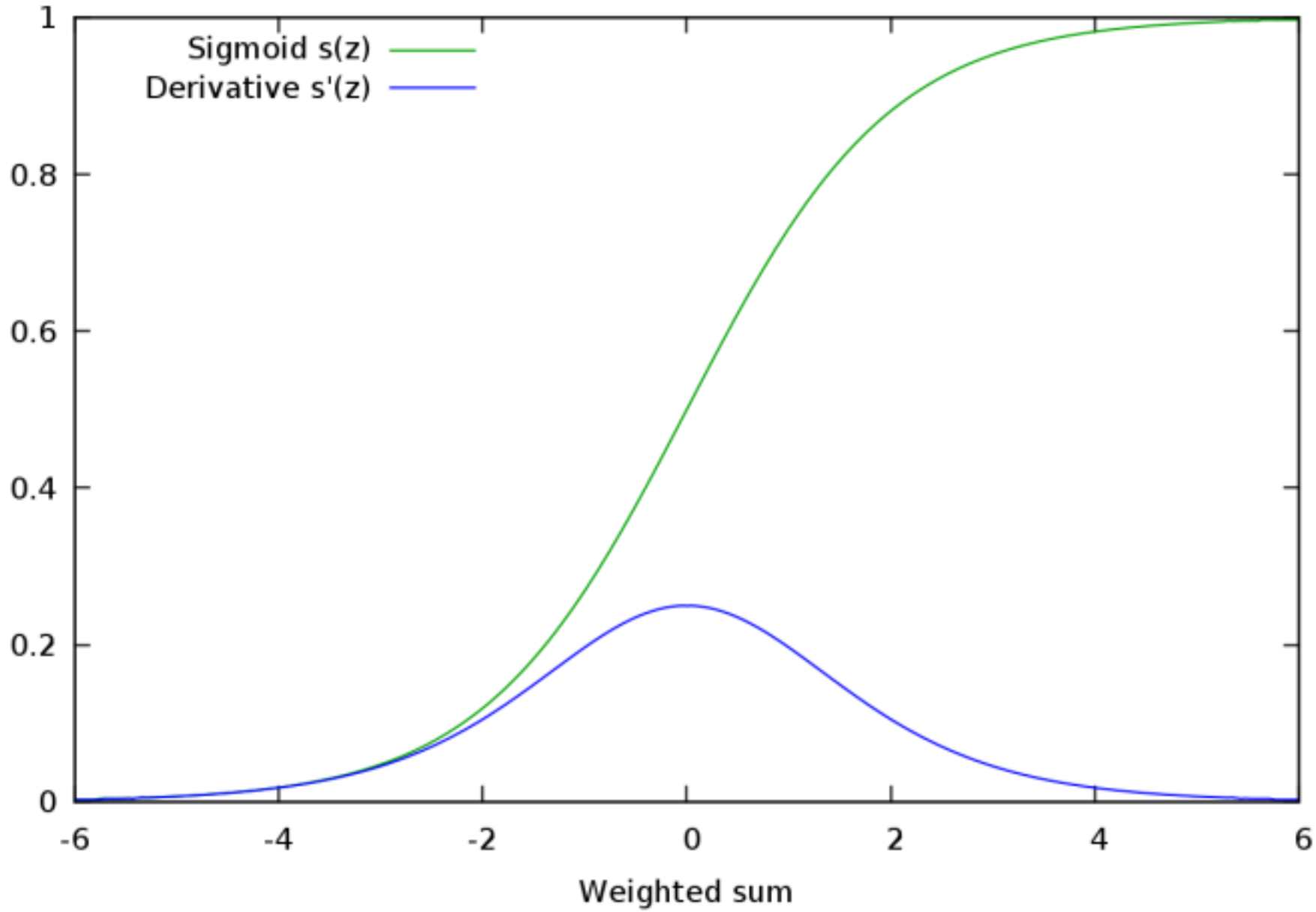


$\lambda_0 = 1.25$
$\lambda_1 = 1$



Recurrence at its Limits: Non-Linear Case

- What happens if you repeatedly apply W and non-linear function f(x)?
  - x := f(Wx); repeat
  - y = f(W(f(W…f(Wx)...)))
- Gradient
  - $\frac{\partial y_t}{\partial x_i} = f'(y_t)W^T f'(y_{t-1})W^T \dots f'(y_{i+1})W^T f'(y_i)W^T$
- Adding non-linear transformation rescales vanishing/exploding criteria by largest possible gradient of f
  - Sigmoid - largest derivative value is 0.25
  - To explode, it must be that largest eigenvalue of W^T is > 4
  - If largest eigenvalue of W^T is < 4, then vanishing gradient is guaranteed


Sigmoid s(z)
Derivative s'(z)
1
0.8
0.6
0.4
0.2
0
-6
-4
-2
0
2
4
6
Weighted sum

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$ ← GATE

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$ ← GATE

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$ ← RECURRENT CELL

$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$ ← GATE

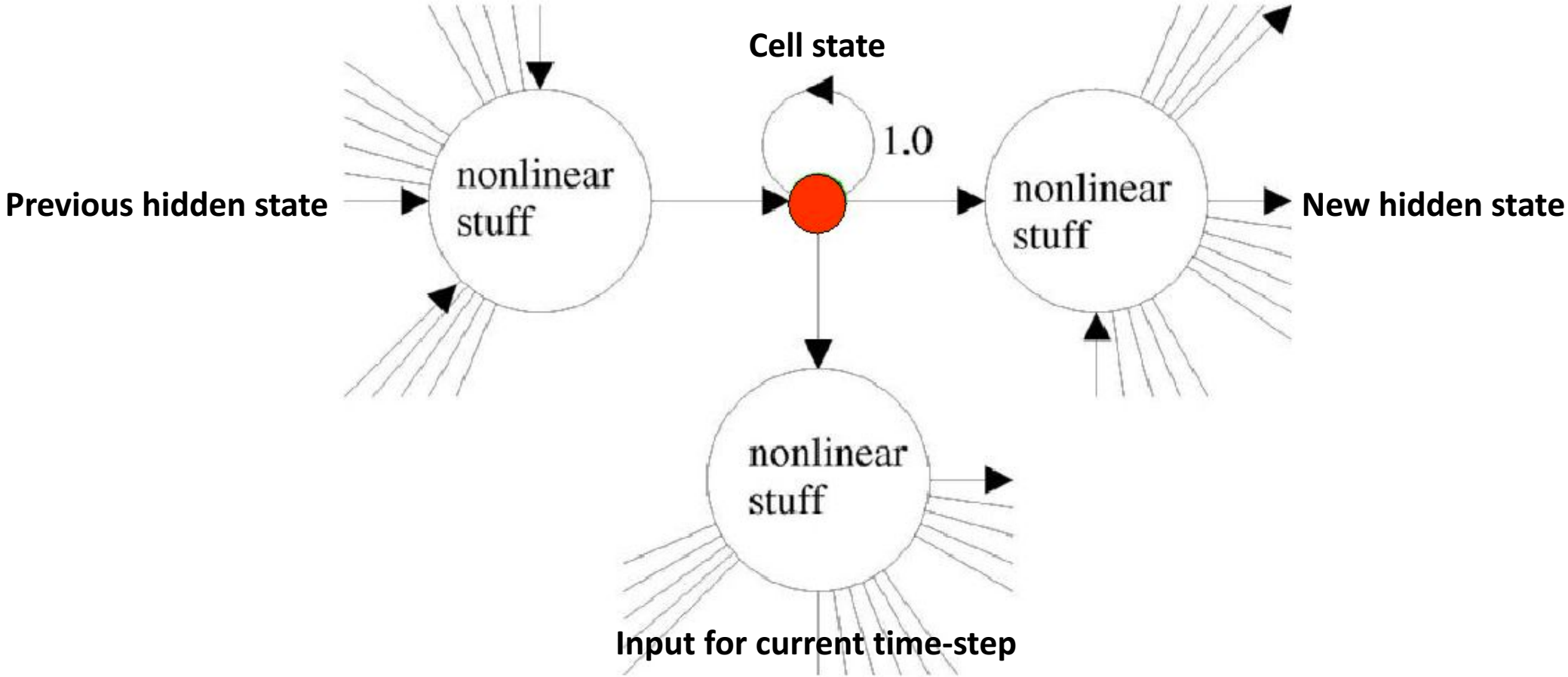
$$h_t = o_t * \tanh\left(C_t\right)$$

$$\frac{\partial C_t}{\partial C_{t-1}} = f_t$$

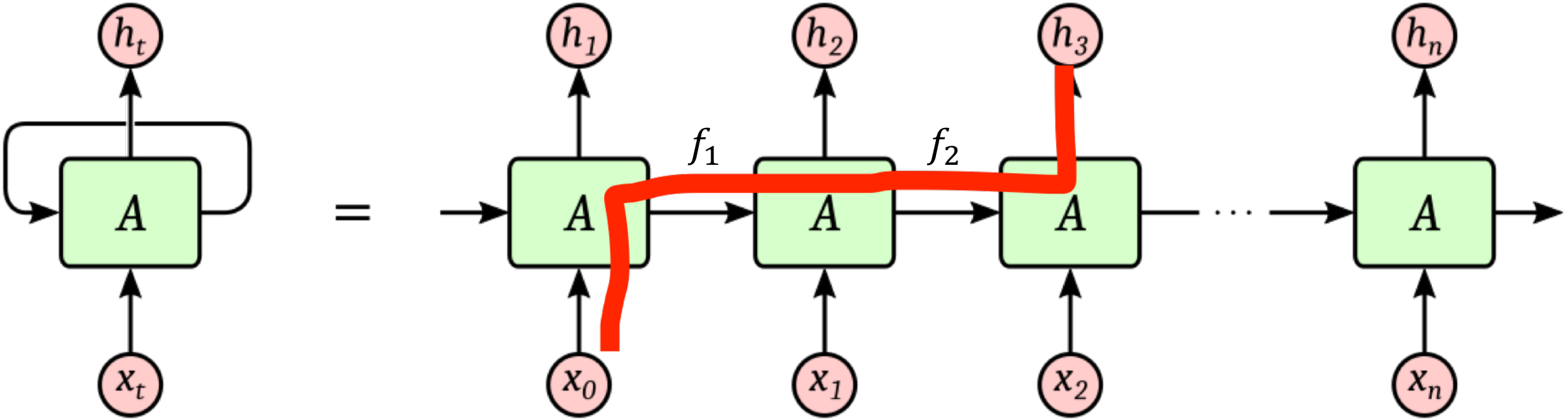
$$\frac{\partial C_t}{\partial C_{t-k}} = f_t * f_{t-1} * \cdots * f_{t-k+1}$$

Long Short-Term Memory (LSTM)

Constant Error Carousel

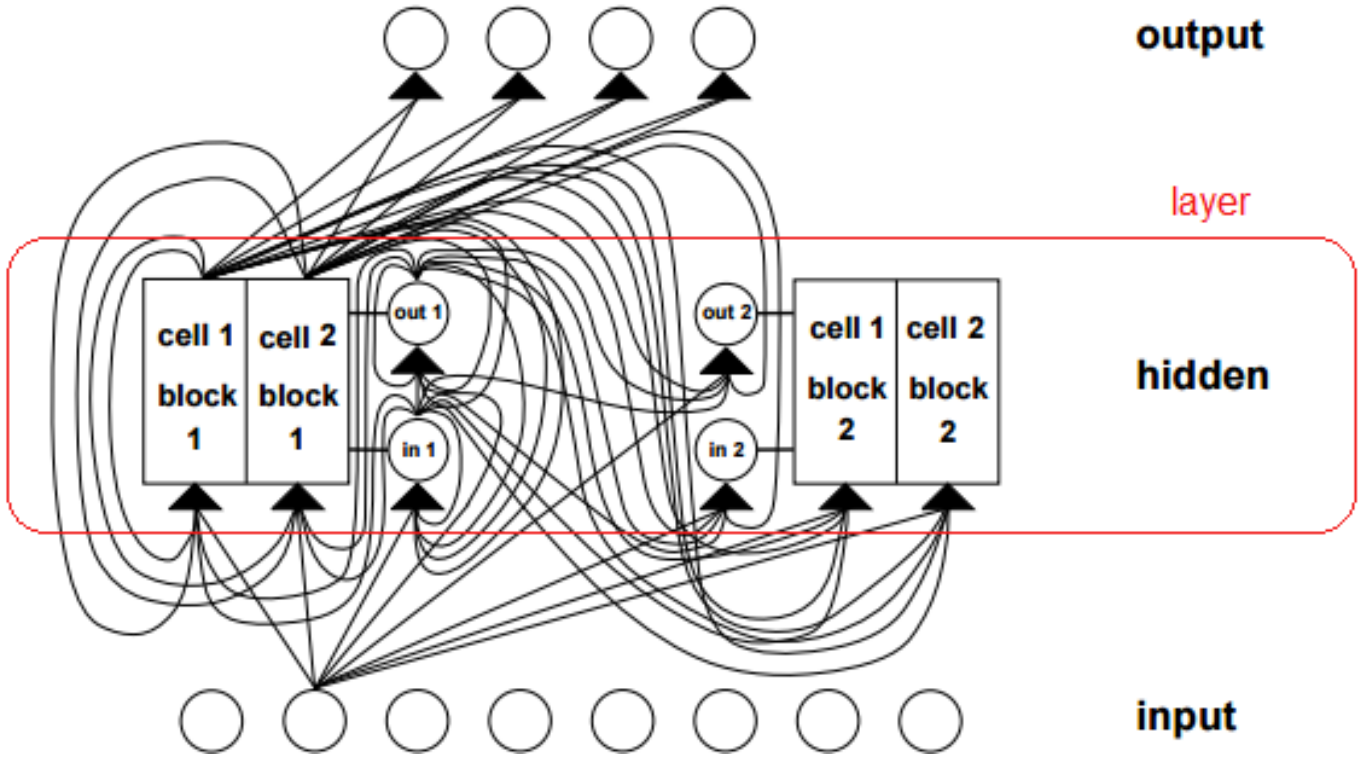


Error Flow through Cell State



Multiple Memory Blocks

- Similar to learning multiple different LSTM cells



← LSTM output is a function of each block's hidden state output

(From my understanding of the paper, the hidden states are added; let me know if I'm mistaken)

←Each block has its own:

- Cell and hidden states
- Weights (f, i, & o gates, candidate)

← Shared Input Representation

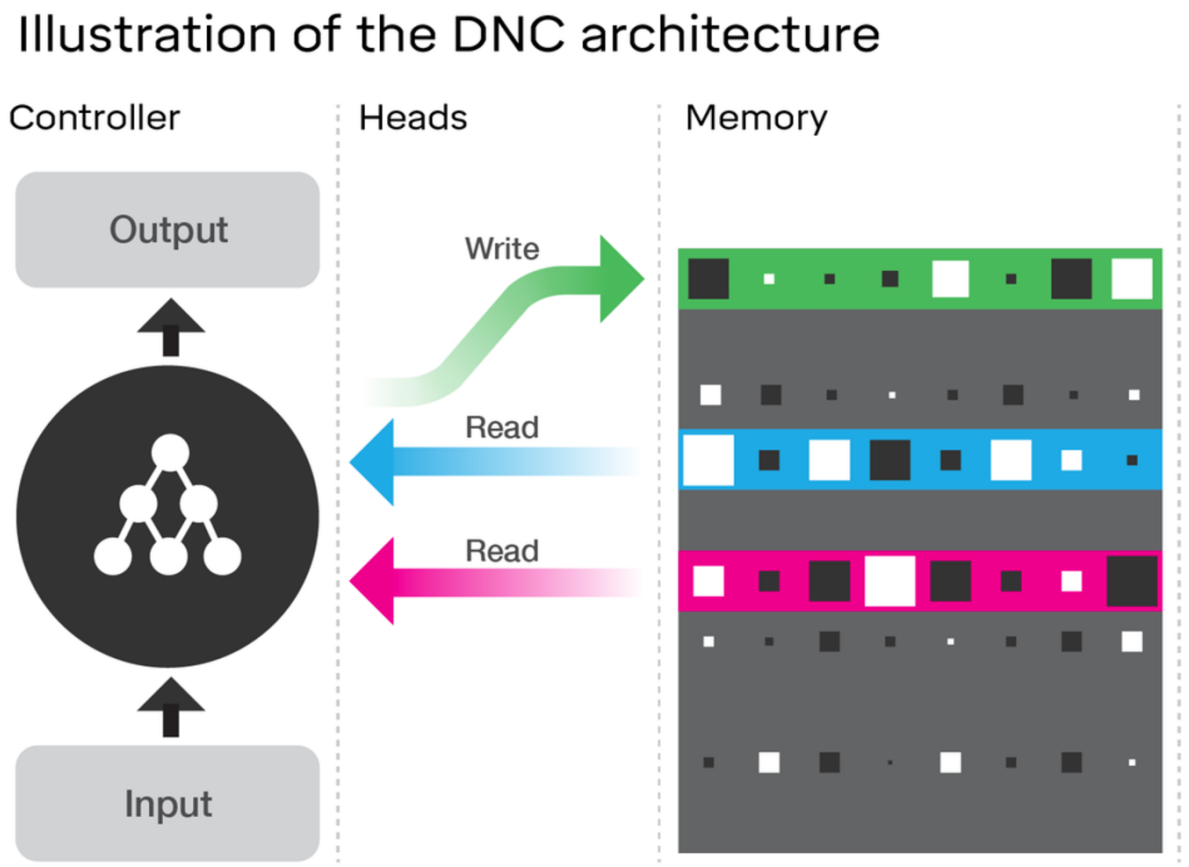
Sounds like Recurrent Entity Networks

Techniques to Prevent Memory Redundancy

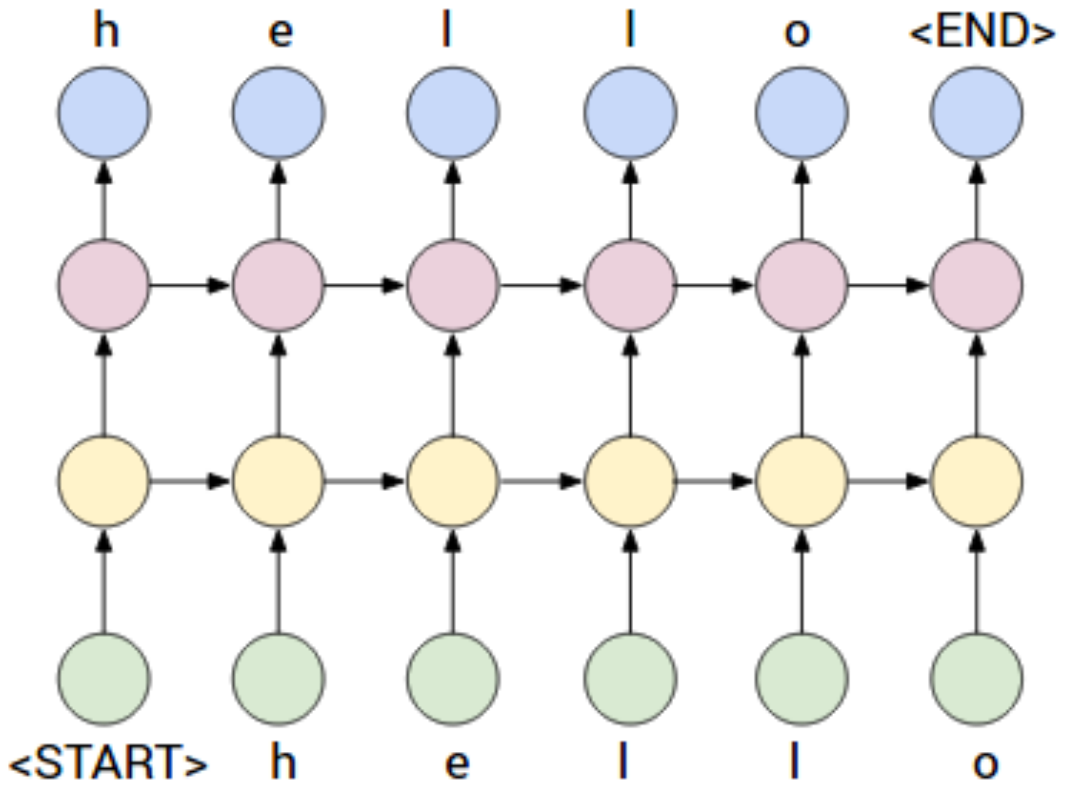
- Original LSTM:
  - 1) Add a new memory block whenever the performance of the model stops improving
  - 2) Initialize the bias of the output gate for memory cells to negative values → encourages the activations of the output (hidden state) toward zero initially
    - The model can learn to “allocate” these memory cells over time by changing these bias values
- Recurrent Entity Networks (Henaff et al. 2016):
  - Each memory block (LSTM unit) has a key, weights are tied with the word embeddings of certain chosen entities
  - Biases inputs related to the key to be written to that block

Wormholes: Sparsity in Memory Reads

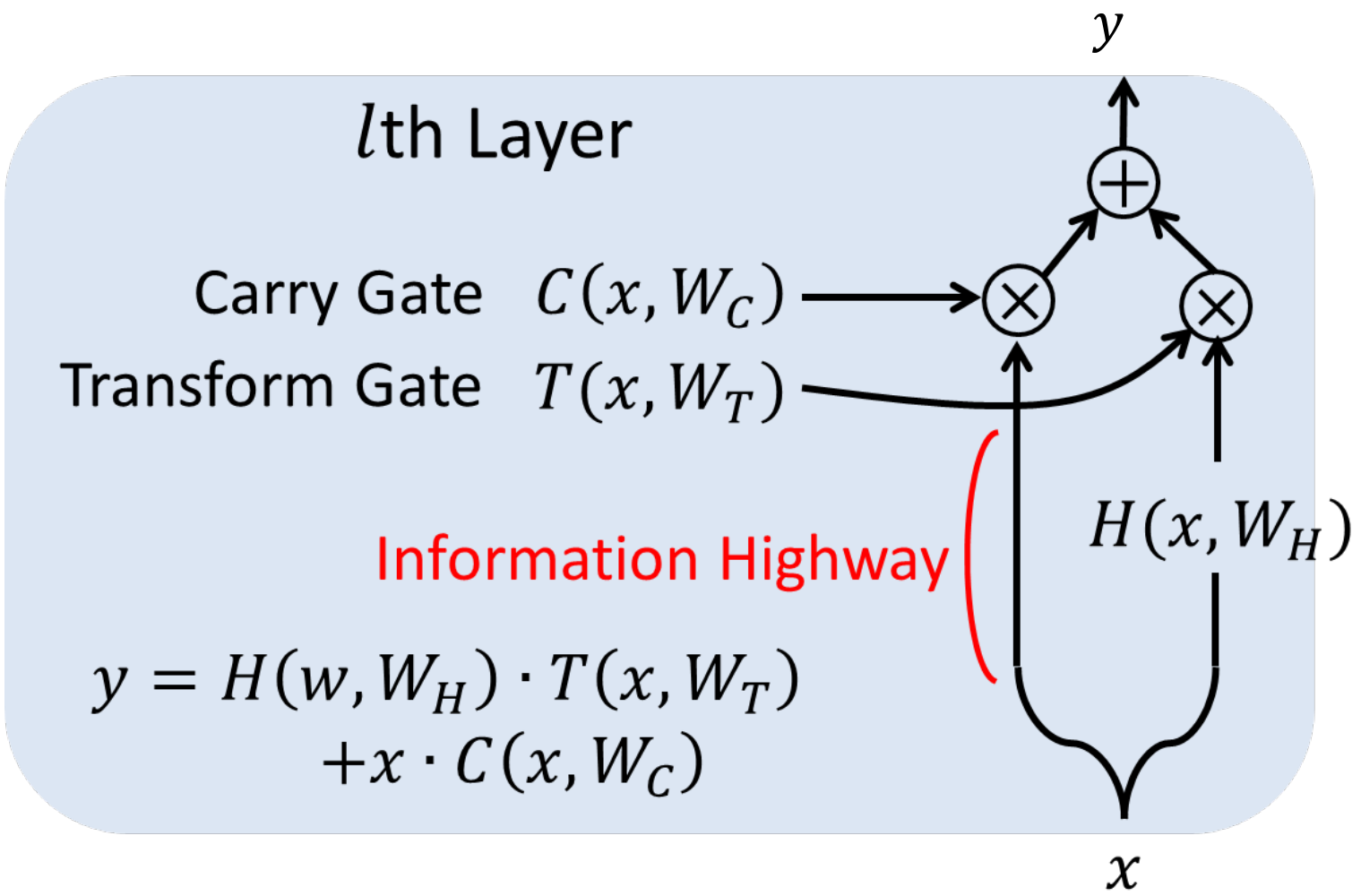
- Using REINFORCE for memory addressing
  - Scaling Memory-Augmented Neural Networks with Sparse Reads and Writes (Rae et al. 2016)
- Using K-nearest neighbors for memory addressing
  - Memory Augmented Neural Networks with Wormhole Connections (Gulcehre et al. 2017)



Depth through Gates and Highways:
A Mini Exploration



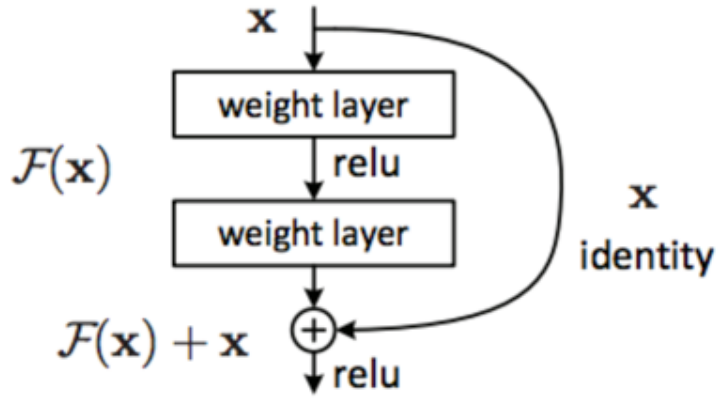
Highway Networks: Deeper Through Gates



Srivastava et al. 2015

- Can train 900 layer deep models with SGD
- Implicit version of residual networks

Residual Network



He et al. 2015

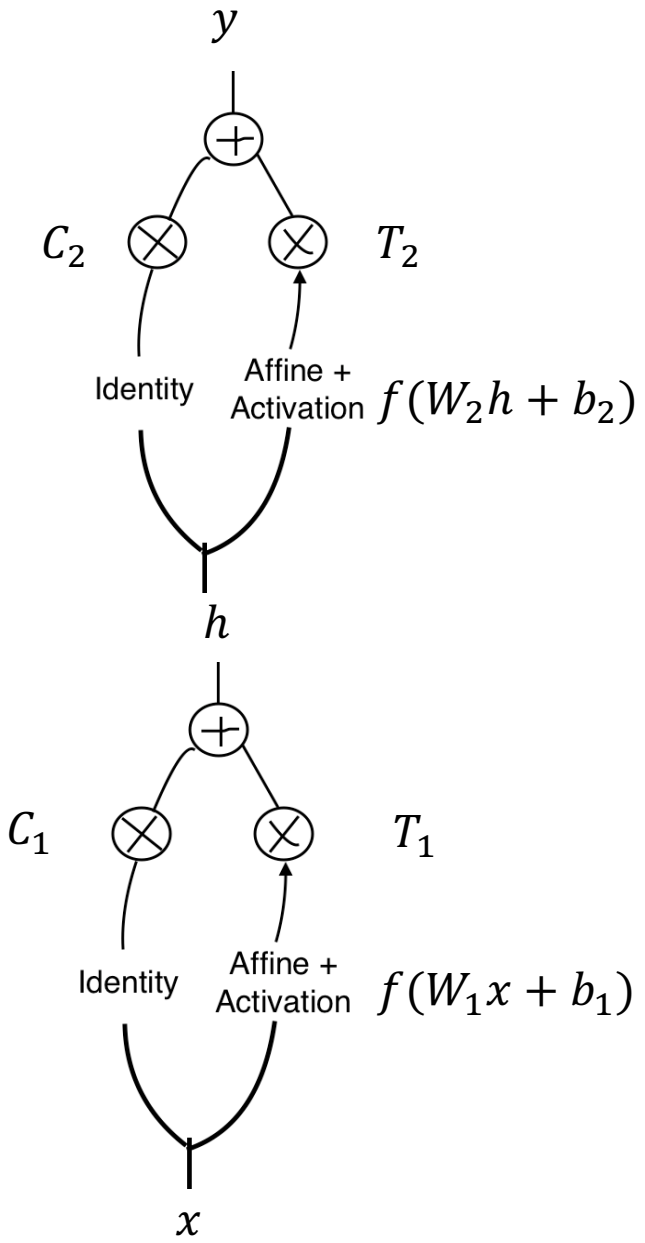
Highway Gradients

$$\frac{\partial y}{\partial x} = C_2 * C_1 +$$

$$C_2 * T_1 * f'(W_1 x + b_1) W_1^T +$$

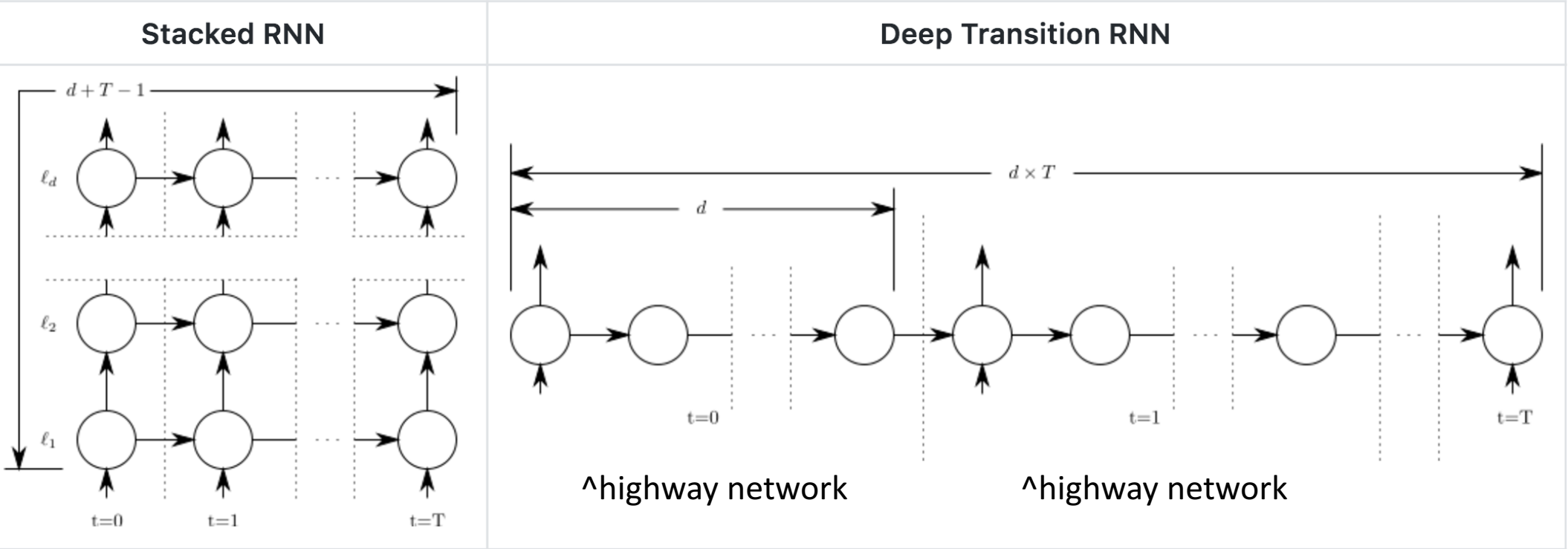
$$T_2 * f'(W_2 h + b_2) W_2^T * C_1 +$$

$$T_2 * f'(W_2 h + b_2) W_2^T * T_1 * f'(W_1 x + b_1) W_1^T$$



Recurrent Highway Networks

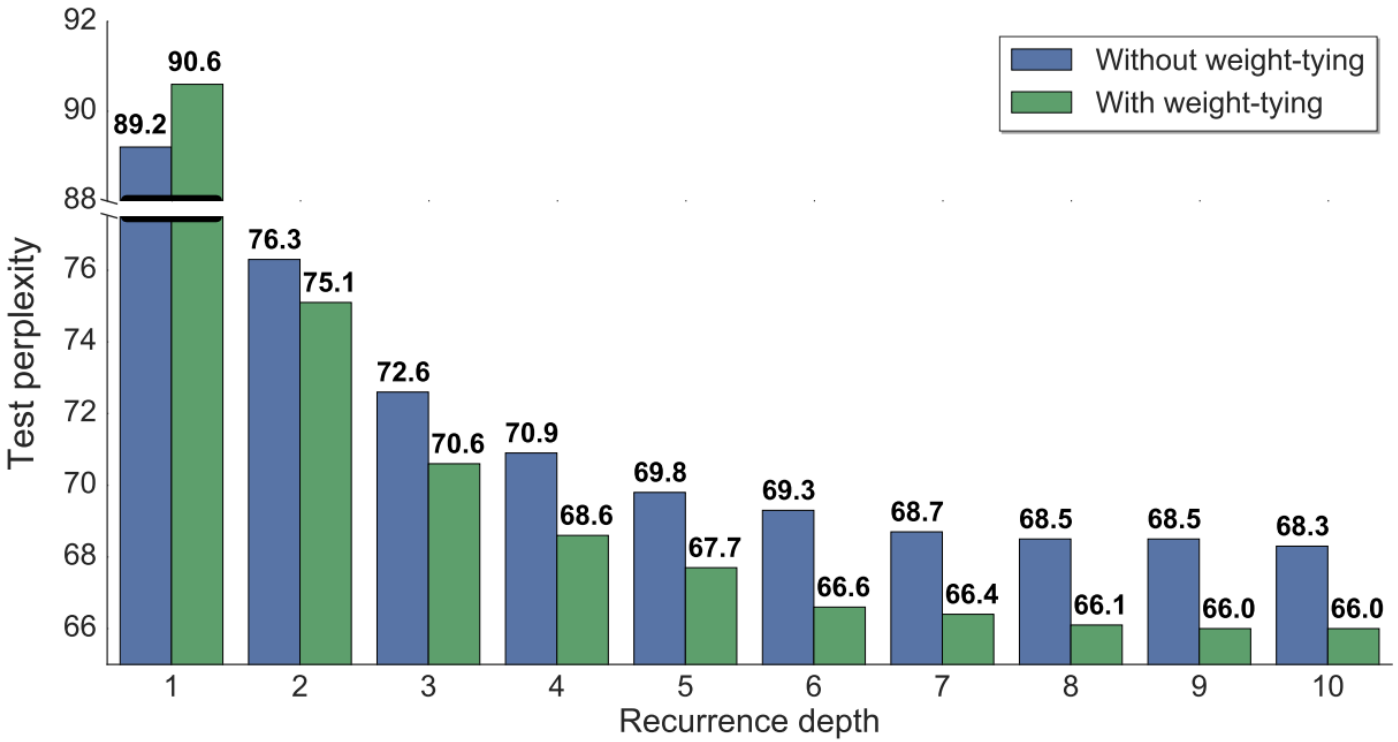
Adaptive Computation Time



$$\mathbf{s}_\ell^{[t]} = \mathbf{h}_\ell^{[t]} \cdot \mathbf{t}_\ell^{[t]} + \mathbf{s}_{\ell-1}^{[t]} \cdot \mathbf{c}_\ell^{[t]}$$

$$\begin{aligned}
\mathbf{h}_\ell^{[t]} &= tanh(\mathbf{W}_H \mathbf{x}^{[t]} \mathbb{I}_{\{\ell=1\}} + \mathbf{R}_{H_\ell} \mathbf{s}_{\ell-1}^{[t]} + \mathbf{b}_{H_\ell}), \\
\mathbf{t}_\ell^{[t]} &= \quad \sigma(\mathbf{W}_T \mathbf{x}^{[t]} \mathbb{I}_{\{\ell=1\}} + \mathbf{R}_{T_\ell} \mathbf{s}_{\ell-1}^{[t]} + \mathbf{b}_{T_\ell}), \\
\mathbf{c}_\ell^{[t]} &= \quad \sigma(\mathbf{W}_C \mathbf{x}^{[t]} \mathbb{I}_{\{\ell=1\}} + \mathbf{R}_{C_\ell} \mathbf{s}_{\ell-1}^{[t]} + \mathbf{b}_{C_\ell}),
\end{aligned}$$

Penn Treebank Language Modeling



Model	Size	Best Val.	Test
RNN-LDA + KN-5 + cache (Mikolov & Zweig, 2012)	9 M	–	92.0
Conv.+Highway+LSTM+dropout (Kim et al., 2015)	19 M	–	78.9
LSTM+dropout (Zaremba et al., 2014)	66 M	82.2	78.4
Variational LSTM (Gal, 2015)	66 M	77.3	75.0
Variational LSTM + WT (Press & Wolf, 2016)	51 M	75.8	73.2
Pointer Sentinel-LSTM (Merity et al., 2016)	21 M	72.4	70.9
Variational LSTM + WT + augmented loss (Inan et al., 2016)	51 M	71.1	68.5
Variational RHN	**32 M**	**71.2**	**68.5**
Neural Architecture Search with base 8 (Zoph & Le, 2016)	32 M	–	67.9
Variational RHN + WT	**23 M**	**67.9**	**65.4**
Neural Architecture Search with base 8 + WT (Zoph & Le, 2016)	25 M	–	64.0
Neural Architecture Search with base 8 + WT (Zoph & Le, 2016)	54 M	–	62.4

Character level language modeling:
Hutter Prize (Compressing Wikipedia)

Model	BPC	Size
Grid-LSTM (Kalchbrenner et al., 2015)	1.47	17 M
MI-LSTM (Wu et al., 2016)	1.44	17 M
mLSTM (Krause et al., 2016)	1.42	21 M
LN HyperNetworks (Ha et al., 2016)	1.34	27 M
LN HM-LSTM (Chung et al., 2016)	1.32	35 M
RHN - Rec. depth 5	**1.31**	**23** M
RHN - Rec. depth 10	**1.30**	**21** M
Large RHN - Rec. depth 10	**1.27**	**46** M

Enwiki8 (27 unicode symbols)

Model	BPC	Size
MI-LSTM (Wu et al., 2016)	1.44	17 M
mLSTM (Krause et al., 2016)	1.40	10 M
BN LSTM (Cooijmans et al., 2016)	1.36	16 M
HM-LSTM (Chung et al., 2016)	1.32	35 M
LN HM-LSTM (Chung et al., 2016)	1.29	35 M
RHN - Rec. depth 10	**1.29**	**20** M
Large RHN - Rec. depth 10	**1.27**	**45** M

Text8 (27 unicode symbols)

(Performance is measured in entropy bits per character.
I am not sure what bits mean here exactly...)

Questions about Depth in RNNs

- Is there a way to effectively add “depth” to tree / recursive neural networks?
- What are stacked LSTM’s learning? Are they learning hierarchical structure? Interpretable?
- How can we make neural networks explicitly learn hierarchical structure / chunking in sequences?
  - Adaptive Sequence Chunker (Schmidhuber 1992)
  - Hierarchical Multiscale Recurrent Neural Networks (Chung et al. 2017)
  - Learning to Compose (Yogatama et al. 2016) / Andrew & Sam