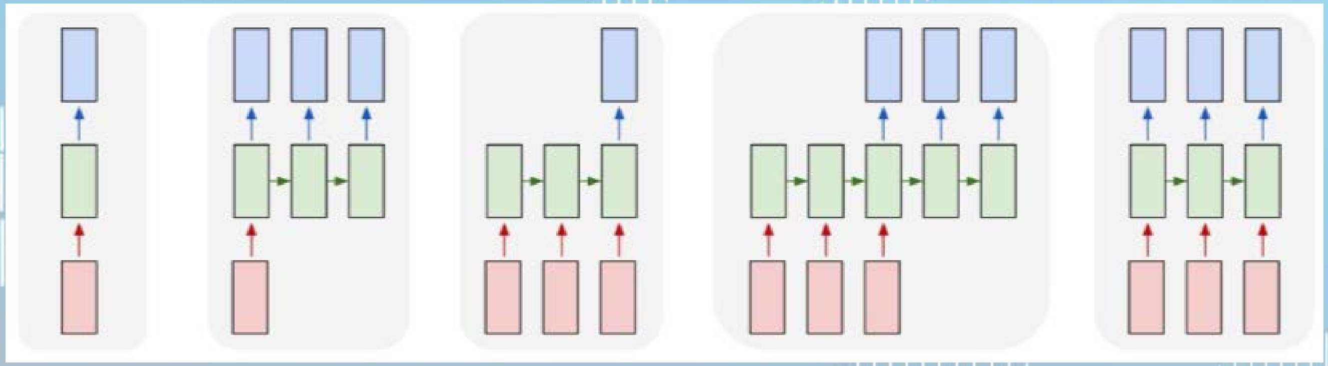
SIEMENS

LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN



Lecture-05: Recurrent Neural Networks

(Deep Learning & AI)

***Speaker**: Pankaj Gupta*

PhD Student (Advisor: Prof. Hinrich Schütze) CIS, University of Munich (LMU)

Research Scientist (NLP/Deep Learning), Machine Intelligence, Siemens AG | Nov 2018

Lecture Outline

- Motivation: Sequence Modeling
- Understanding Recurrent Neural Networks (RNNs)
- Challenges in vanilla RNNs: Exploding and Vanishing gradients. Why? Remedies?
- RNN variants:
  - Long Short Term Memory (LSTM) networks, Gated recurrent units (GRUs)
  - Bi-directional Sequence Learning
  - Recursive Neural Networks (RecNNs): TreeRNNs and TreeLSTMs
  - Deep, Multi-tasking and Generative RNNs (overview)
- Attention Mechanism: Attentive RNNs
- RNNs in Practice + Applications
- Introduction to Explainability/Interpretability of RNNs

Motivation: Need for Sequential Modeling

? Why do we need *Sequential Modeling*?

Motivation: Need for Sequential Modeling

Examples of Sequence data	Input Data	Output
Speech Recognition		*This is RNN*
Machine Translation	Hello, I am Pankaj.	Hallo, ich bin Pankaj. हैलो, मैं पंकज हूं।
Language Modeling	Recurrent neural ? based ? model	**network** **language**
Named Entity Recognition	Pankaj lives in Munich	**Pankaj** lives in **Munich** ***person*** ***location***
Sentiment Classification	There is nothing to like in this movie.	
Video Activity Analysis		Punching

Motivation: Need for Sequential Modeling

Inputs, Outputs can be different lengths in different examples

Example:

Sentence1: Pankaj lives in Munich

Sentence2: Pankaj Gupta lives in Munich DE

Motivation: Need for Sequential Modeling

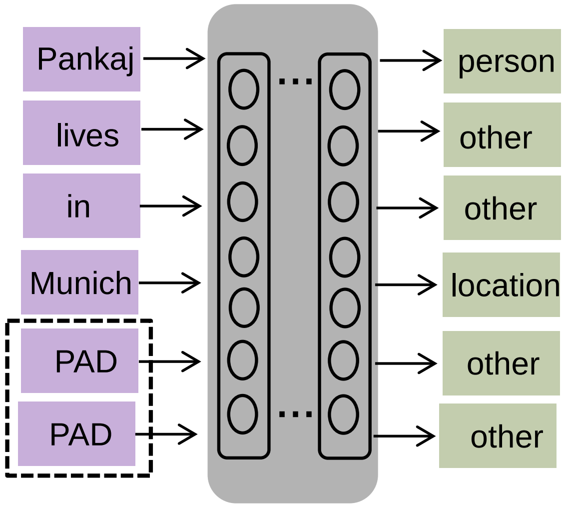
Inputs, Outputs can be different lengths in different examples

Example:

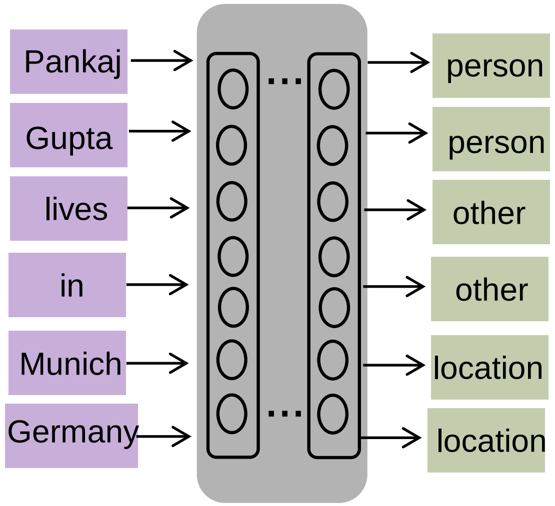
Sentence1: Pankaj lives in Munich

Sentence2: Pankaj Gupta lives in Munich DE

Additional word 'PAD' i.e., padding



FF-net / CNN



FF-net / CNN

***FF-net: Feed-forward network**

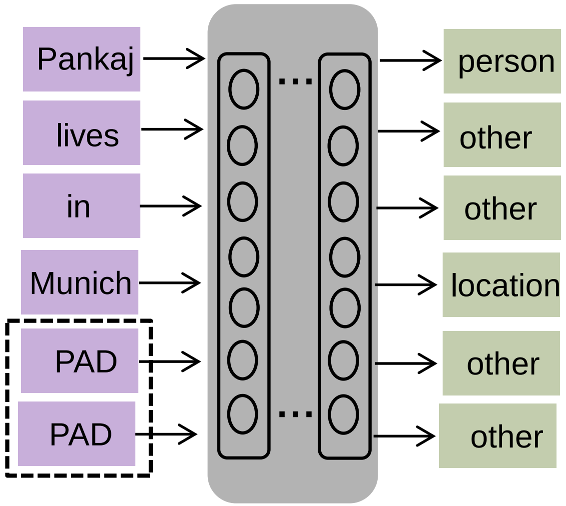
Motivation: Need for Sequential Modeling

Inputs, Outputs can be different lengths in different examples

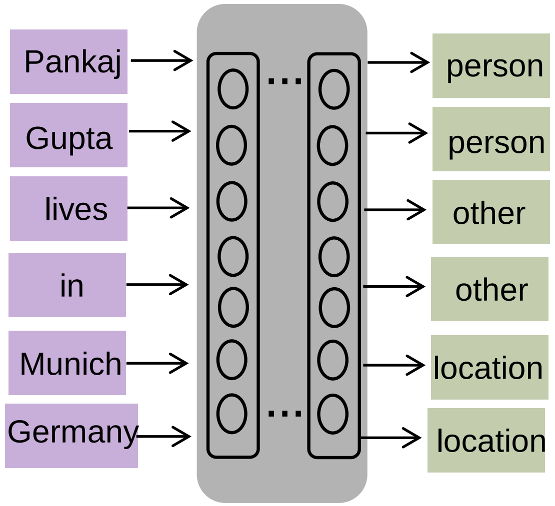
Example:

Sentence1: Pankaj lives in Munich

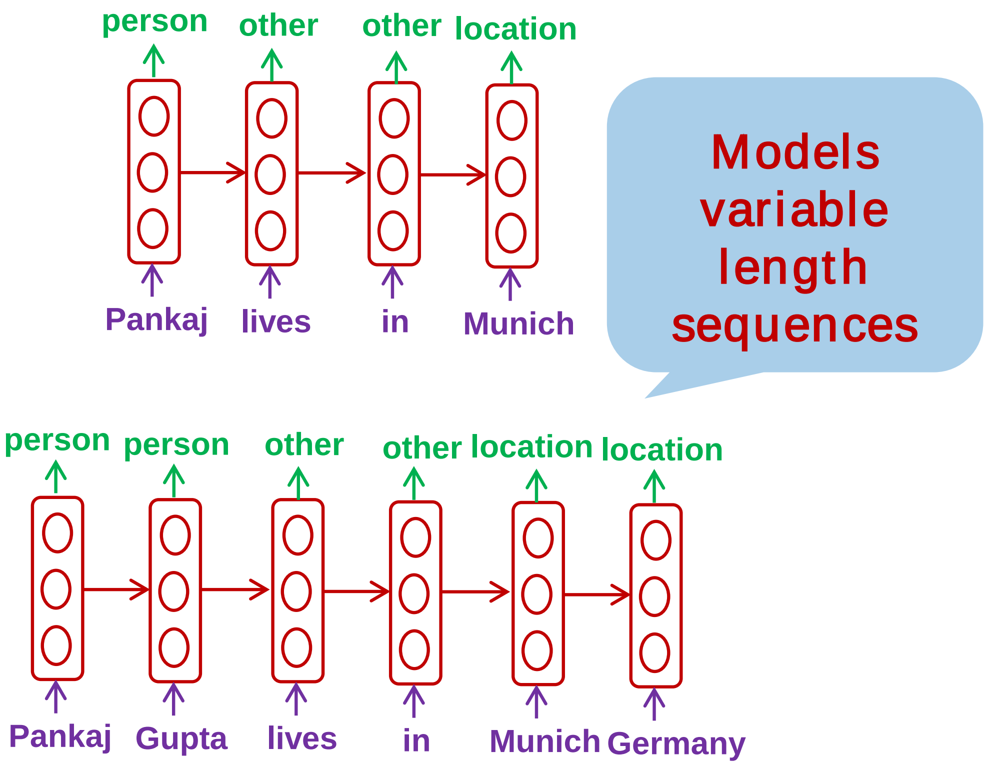
Sentence2: Pankaj Gupta lives in Munich DE



FF-net / CNN



FF-net / CNN



Sequential model: RNN

*FF-net: Feed-forward network

Motivation: Need for Sequential Modeling

Share Features learned across different positions or time steps

Example:

Sentence1: *Market falls into bear territory* → *Trading/Marketing*

Sentence2: Bear falls into market territory → *UNK*

Same uni-gram statistics

Motivation: Need for Sequential Modeling

Share Features learned across different positions or time steps

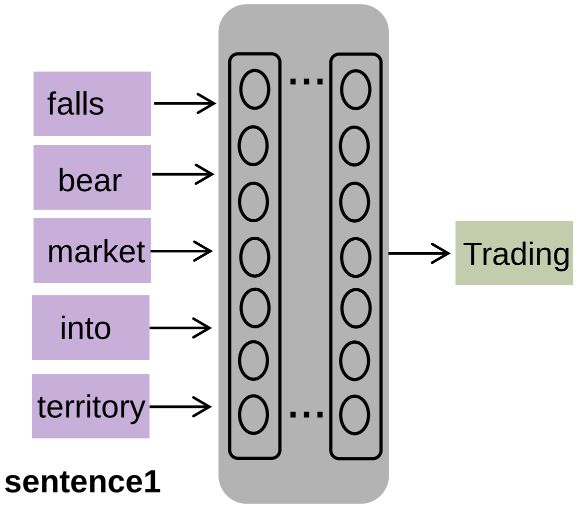
Example:

Sentence1: *Market falls into bear territory* → *Trading/Marketing*

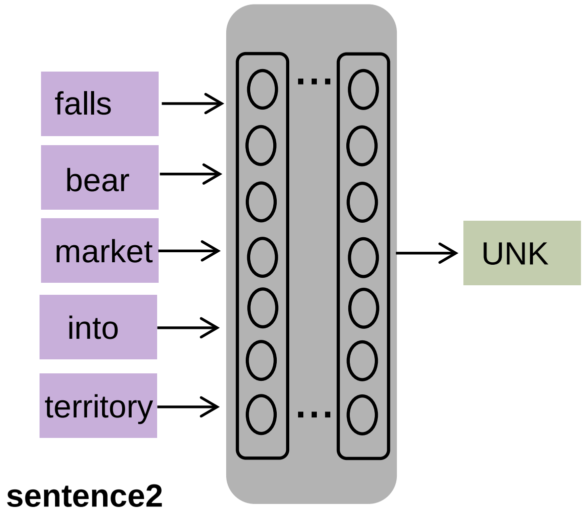
Sentence2: *Bear falls into market territory* → *UNK*

No sequential or temporal modeling, i.e., order-less

Treats the two sentences the same



FF-net / CNN



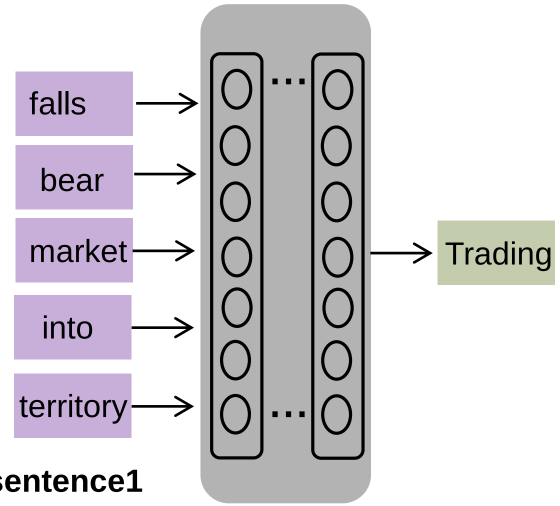
FF-net / CNN

Share Features learned across different positions or time steps

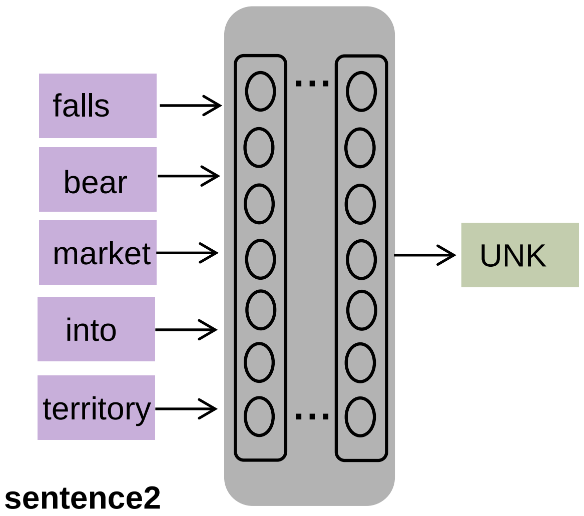
Example:

Sentence1: *Market falls into bear territory* → *Trading/Marketing*

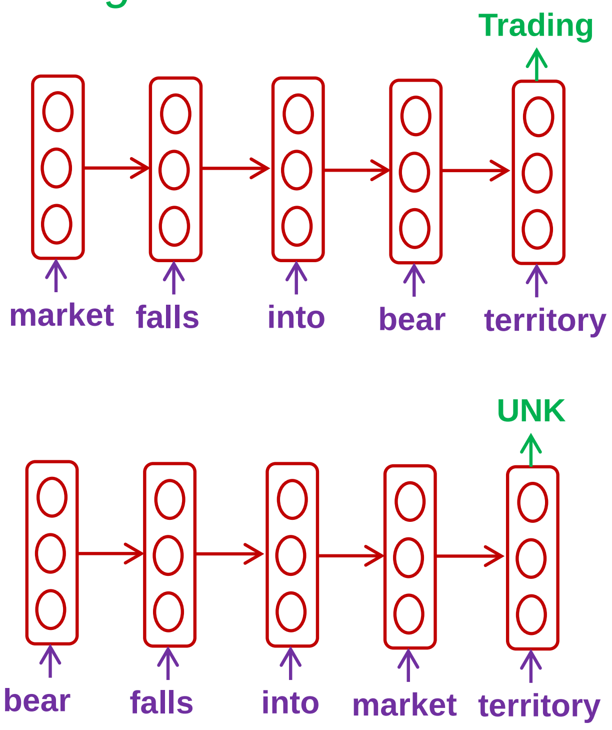
Sentence2: Bear falls into market territory → *UNK*



FF-net / CNN



FF-net / CNN



Sequential model: RNN

Language concepts,

Word ordering,

Syntactic & semantic information

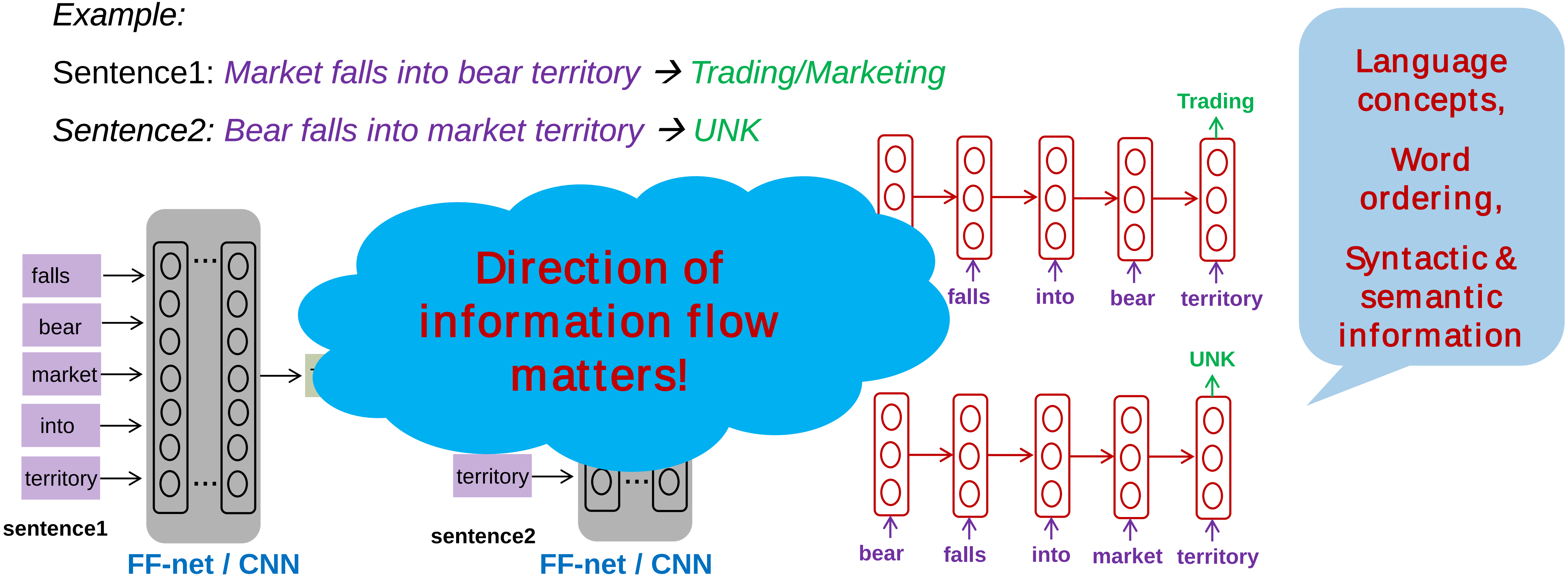
Motivation: Need for Sequential Modeling

Share Features learned across different positions or time steps

Example:

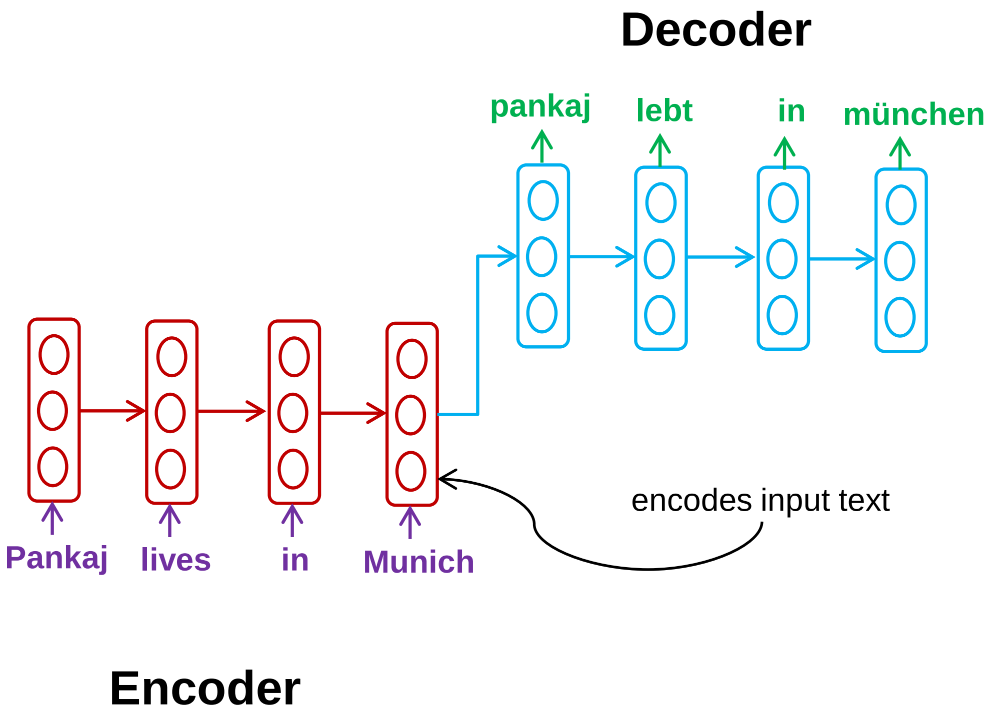
Sentence1: *Market falls into bear territory* → *Trading/Marketing*

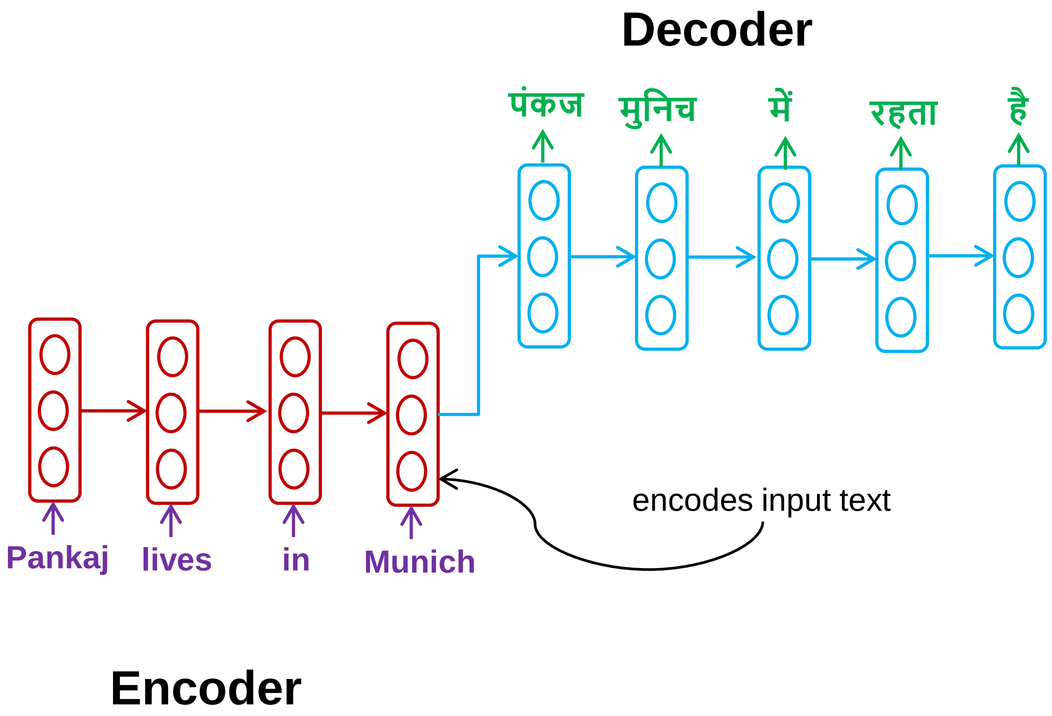
Sentence2: Bear falls into market territory → *UNK*



Sequential model: RNN

Machine Translation: Different Input and Output sizes, incurring sequential patterns





Motivation: Need for Sequential Modeling

Convolutional vs Recurrent Neural Networks

RNN

- perform well when the input data is interdependent in a sequential pattern
- correlation between previous input to the next input
- introduce bias based on your previous output

CNN/FF-Nets

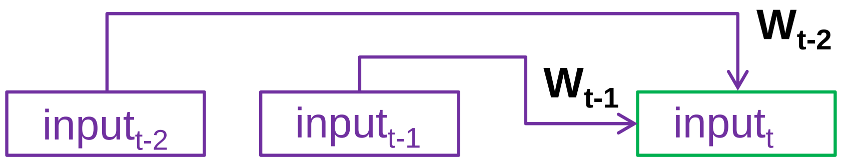
- all the outputs are self dependent
- *Feed-forward nets don't remember historic input data at test time unlike recurrent networks.*

Motivation: Need for Sequential Modeling

Memory-less Models

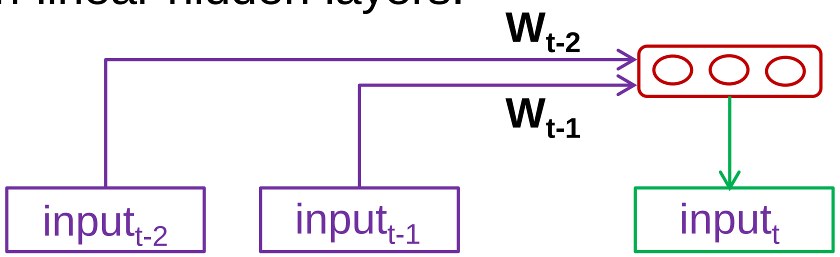
Autoregressive models:

Predict the next input in a sequence from a fixed number of previous inputs using "delay taps".



Feed-forward neural networks:

Generalize autoregressive models by using non-linear hidden layers.



Memory Networks

-possess a dynamic hidden state that can store long term information, e.g., RNNs.

Recurrent Neural Networks:

RNNs are very powerful, because they combine the following properties-

Distributed hidden state: can efficiently store a lot of information about the past.

Non-linear dynamics: can update their hidden state in complicated ways

Temporal and accumulative: can build semantics, e.g., word-by-word in sequence over time

Notations

- $\boldsymbol{h}_t$: Hidden Unit
- $\boldsymbol{x}_t$: Input
- $\boldsymbol{o}_t$: Output
- $\boldsymbol{W}_{hh}$: Shared Weight Parameter
- $\boldsymbol{W}_{ho}$: Parameter weight between hidden layer and output
- θ: parameter in general
- g_θ : non linear function
- L_t :Loss between the RNN outputs and the true output
- E_t : cross entropy loss

Long Term and Short Dependencies

Short Term Dependencies

→ need recent information to perform the present task.

For example in a language model, predict the next word based on the previous ones.

"the clouds are in the **?***"* → *'sky'*

"the clouds are in the ***sky****"*

→ Easier to predict 'sky' given the context, i.e., *short term dependency*

Long Term Dependencies

→ Consider longer word sequence "I grew up in France…..........………………………. I speak fluent ***French***."

→ Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back.

Foundation of Recurrent Neural Networks

Goal

- model long term dependencies
- connect previous information to the present task
- model sequence of events with loops, allowing information to persist

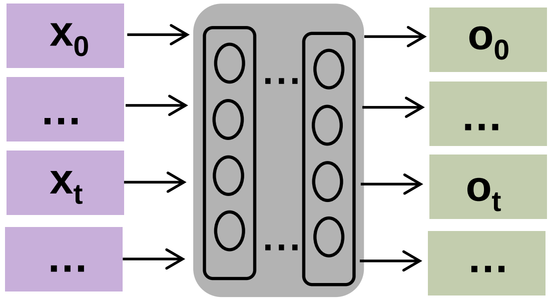
punching

Goal

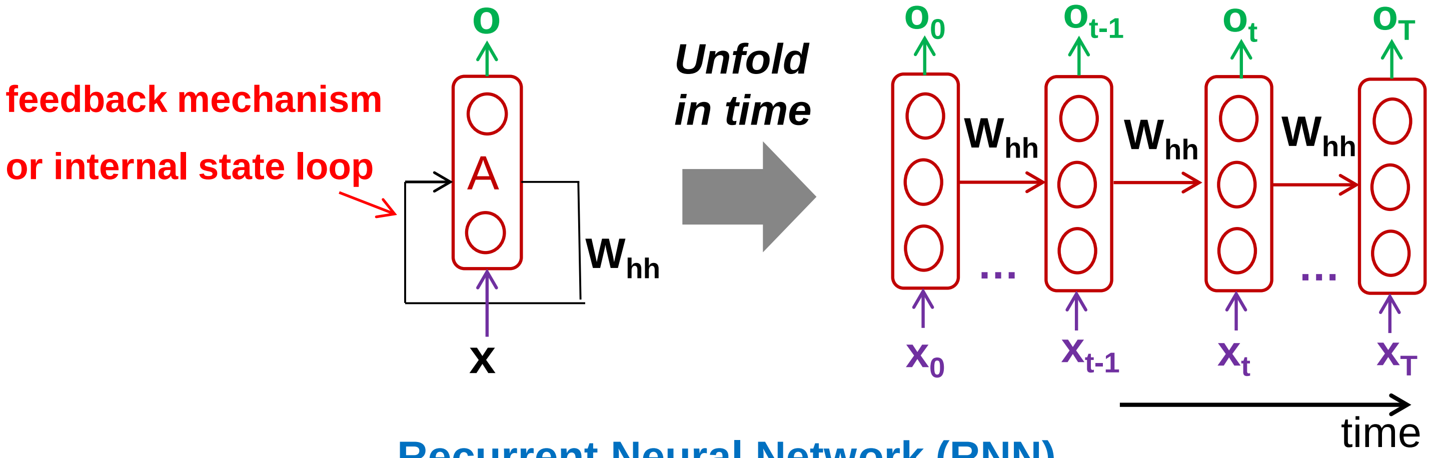
- model long term dependencies
- connect previous information to the present task
- model sequence of events with loops, allowing information to persist

Feed Forward NNets can **not** take **time dependencies** into account.

Sequential data needs a **Feedback Mechanism**.

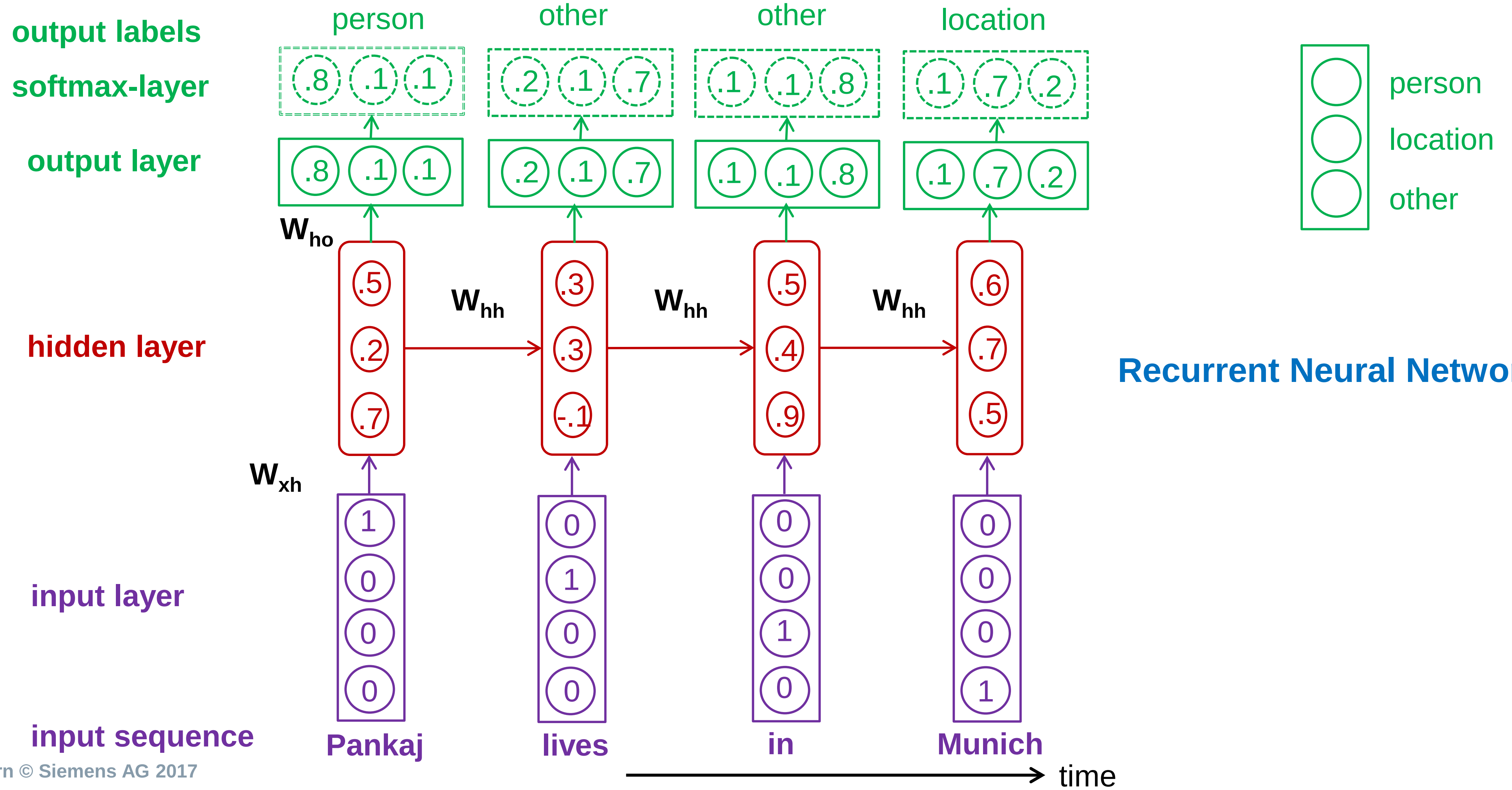


FF-net / CNN



Recurrent Neural Network (RNN)

Foundation of Recurrent Neural Networks



Recurrent Neural Network

(Vanilla) Recurrent Neural Network

Process a sequence of vectors **x** by applying a recurrence at every time step:

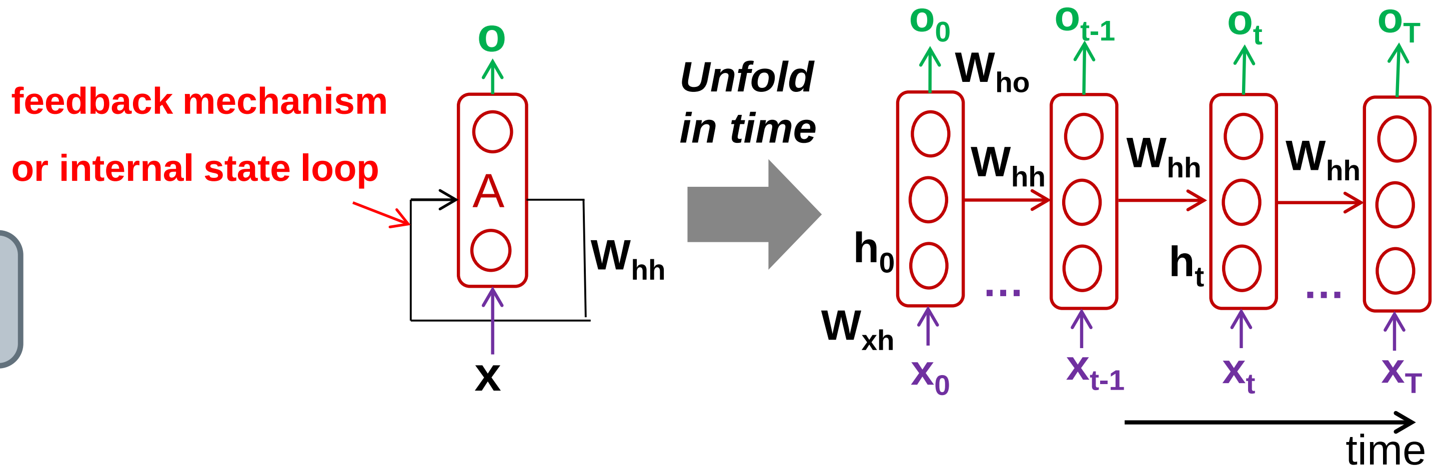
$$h_t = g_\theta(h_{t-1}, x_t)$$

new hidden state at time step, t

some function with parameters $\mathbf{W_{hh}}$ $\mathbf{W_{xh}}$

old hidden state at time step, t-1

Input vector at time step, t



Vanilla Recurrent Neural Network (RNN)

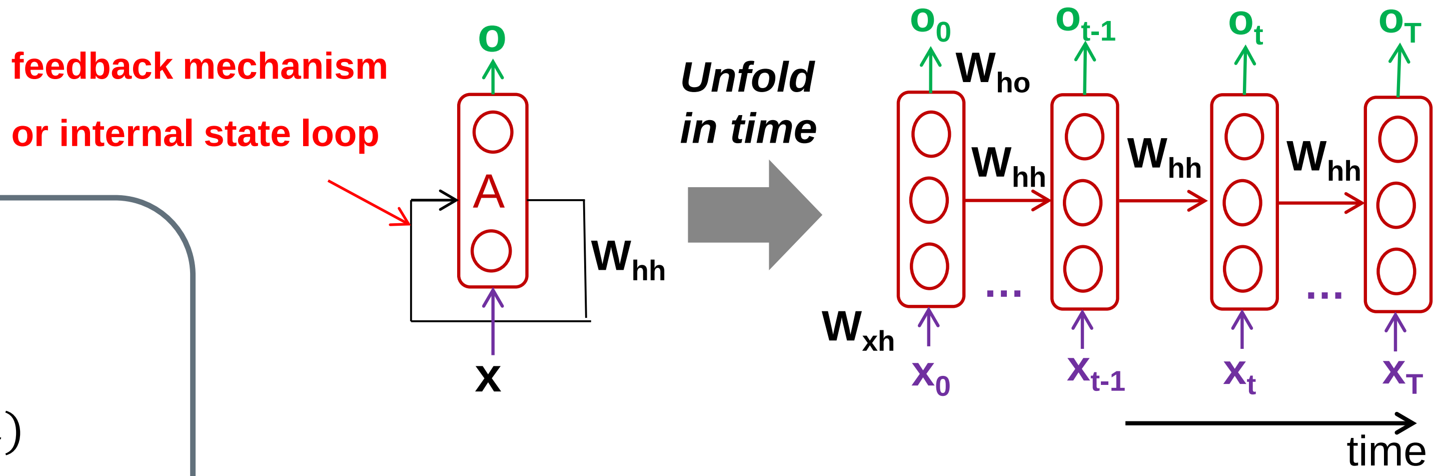
Remark: The same function *g* and same set of parameters **W** are used at every time step

(Vanilla) Recurrent Neural Network

Process a sequence of vectors **x** by applying a recurrence at every time step:

$$h_t = g_\theta(h_{t-1}, x_t)$$
$$\downarrow$$
$$h_t = \tanh(W_{hh}\, h_{t-1} + W_{xh} x_t)$$

$$o_t = softmax(W_{ho} h_t)$$



Vanilla Recurrent Neural Network (RNN)

Remark: RNN's can be seen as **selective summarization** of input sequence in a fixed-size state/hidden vector via a recursive update.

Recurrent Neural Network: Probabilistic Interpretation

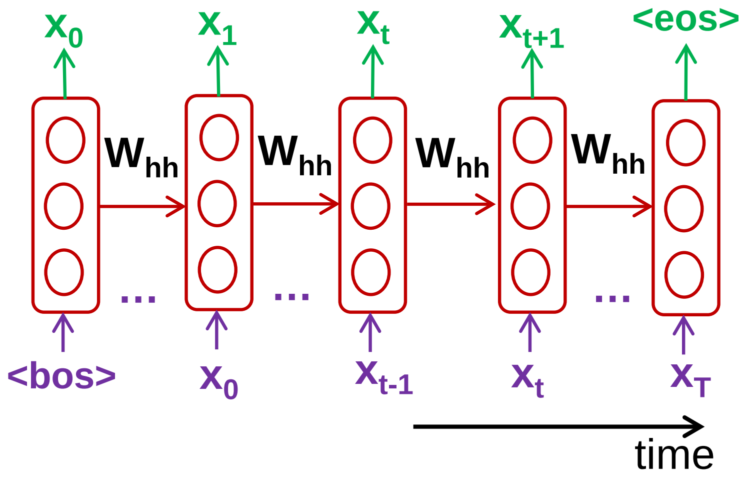
RNN as a generative model

➢ induces a set of procedures to model

the conditional distribution of $\mathbf{x}_{t+1}$ given $\mathbf{x}$<=t

for all t = 1, …,T

$$P(x) = P(x_1, \ldots x_T) = \sum_{t=1}^{T} P(x_t | x_{t-1}, x_{t-2}, .. x_1)$$



Generative Recurrent Neural Network (RNN)

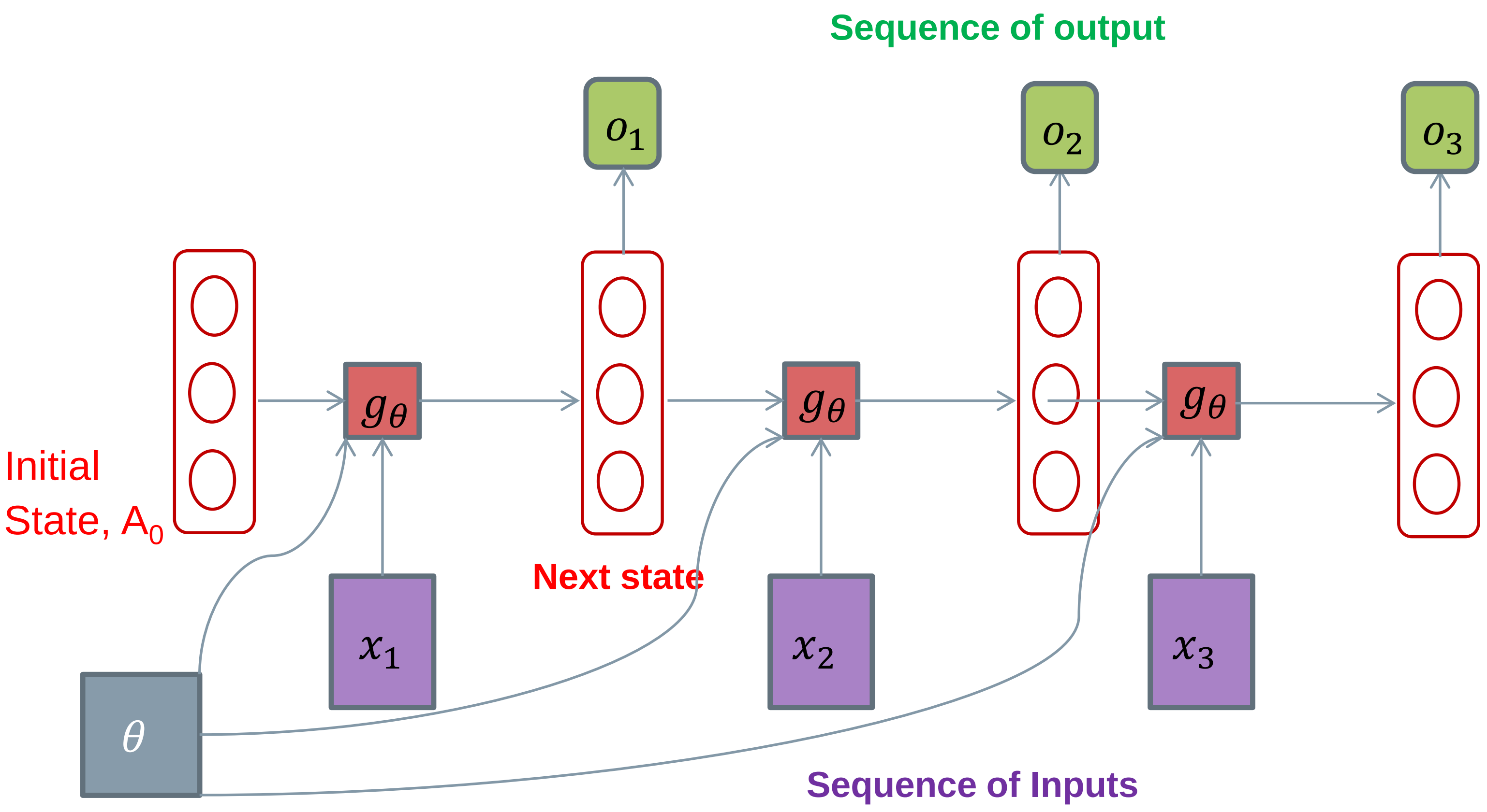
➢ Think of the output as the probability distribution of the $\mathbf{x}_t$ given the previous ones in the sequence

➢Training: Computing probability of the sequence and Maximum likelihood training

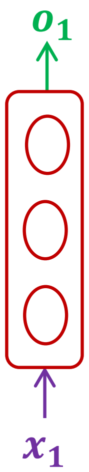
$$L_t = -\log P(x_t | x_{t-1}, x_{t-2}, .. x_1)$$

Details: https://www.cs.cmu.edu/~epxing/Class/10708-17/project-reports/project10.pdf

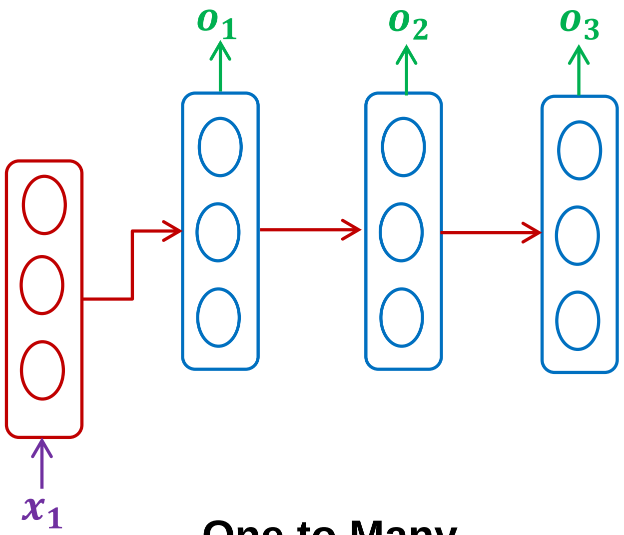
RNN: Computational Graphs



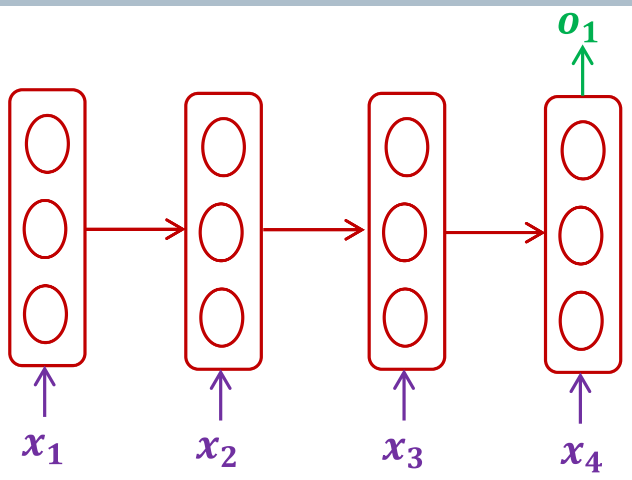
RNN: Different Computational Graphs



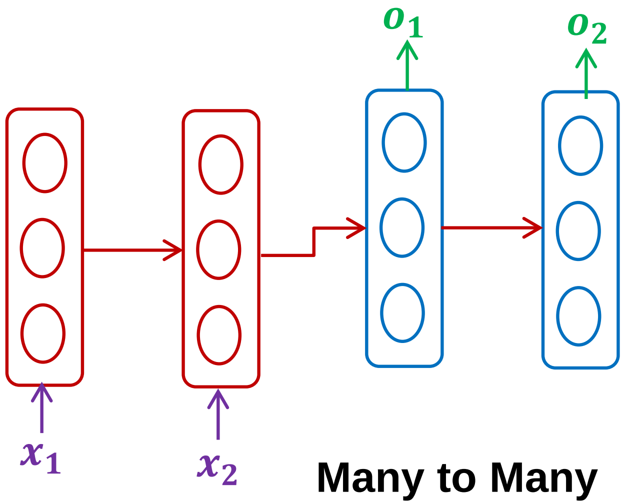
One to one



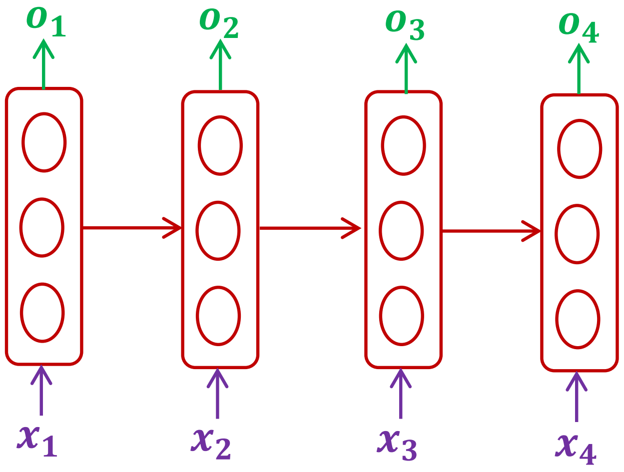
One to Many



Many to One



Many to Many



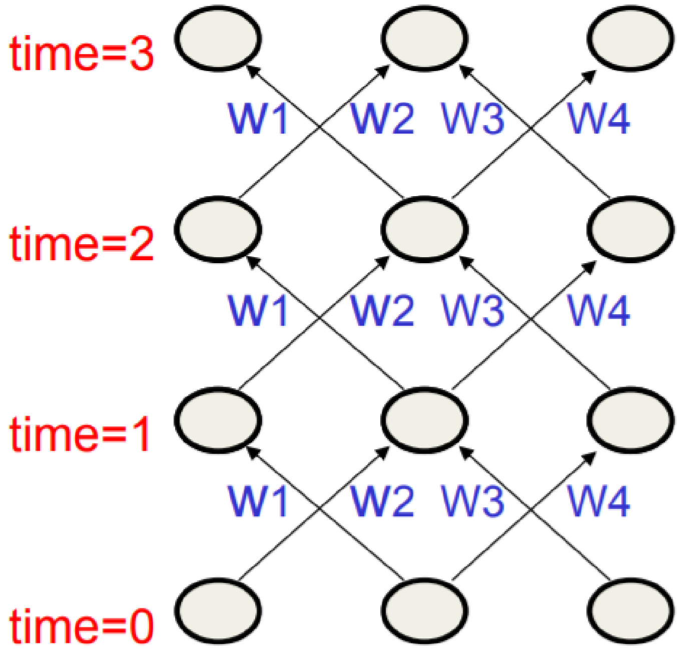
Many to Many

Backpropogation through time (BPTT) in RNN

- Training recurrent networks via BPTT
- Compute gradients via backpropagation on the (multi-layer) unrolled model

Backpropogation through time (BPTT) in RNN

- Training recurrent networks via BPTT
- Compute gradients via backpropagation on the (multi-layer) unrolled model
- Think of the recurrent net as a *layered*, *feed-forward net* with shared weights and then train the feed-forward net in time domain



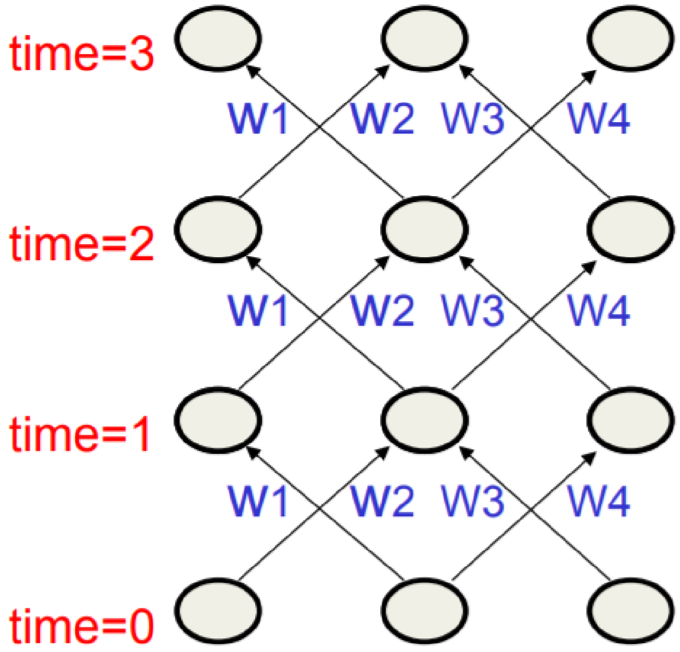
Lecture from the course Neural Networks for Machine Learning

Backpropogation through time (BPTT) in RNN

- Training recurrent networks via BPTT

- Compute gradients via backpropagation on the (multi-layer) unrolled model

- Think of the recurrent net as a *layered*, *feed-forward net* with shared weights and then train the feed-forward net in time domain



Training algorithm in time domain:

- The forward pass builds up a stack of the activities of all the units at each time step
- The backward pass peels activities off the stack to compute the error derivatives at each time step.
- After the backward pass we add together the derivatives at all the different times for each weight.

Lecture from the course Neural Networks for Machine Learning

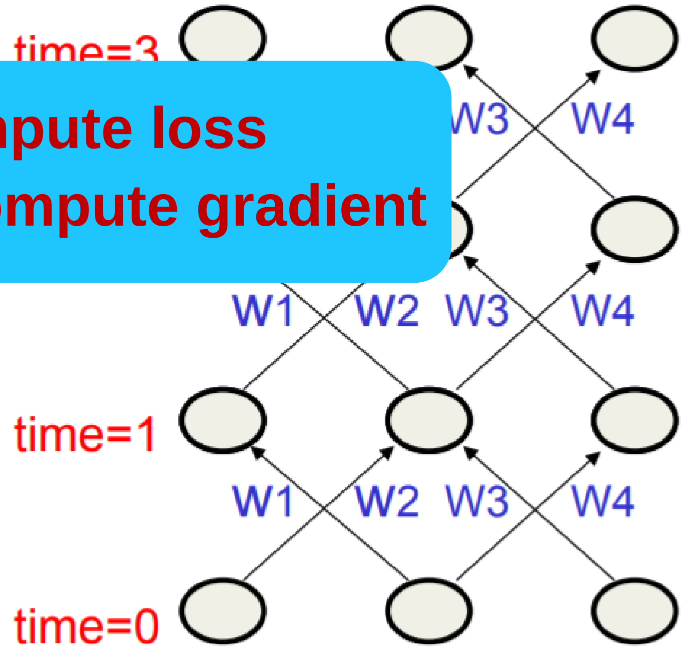
Backpropogation through time (BPTT) in RNN

- Training recurrent networks via BPTT

- Compute gradients via backpropagation on the (multi-layer) unrolled model

- Think of the recurrent net as a *layered*
  feed-forward ne
  then train the fe

Forward through entire sequence → compute loss
Backward through entire sequence → compute gradient



Training algorithm in time domain:

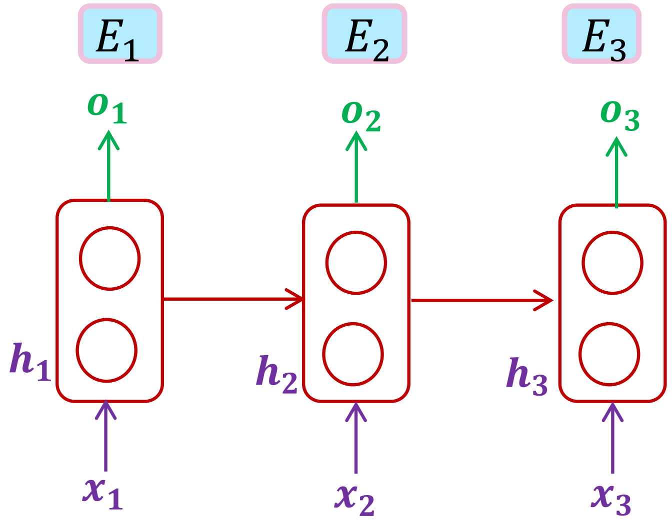
- The forward pass builds up a stack of the activities of all the units at each time step
- The backward pass peels activities off the stack to compute the error derivatives at each time step.
- After the backward pass we add together the derivatives at all the different times for each weight.

Lecture from the course Neural Networks for Machine Learning

Backpropogation through time (BPTT) in RNN

- Training recurrent networks via BPTT

The **output** at time t=T is **dependent** on the inputs from **t=T to t=1**

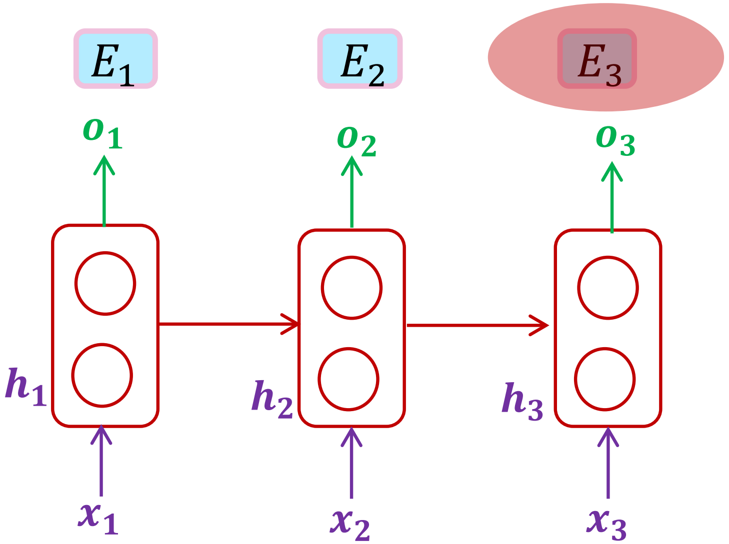


Direction of Forward pass

Backpropogation through time (BPTT) in RNN

- Training recurrent networks via BPTT

The **output** at time t=T is **dependent** on the inputs from **t=T to t=1**

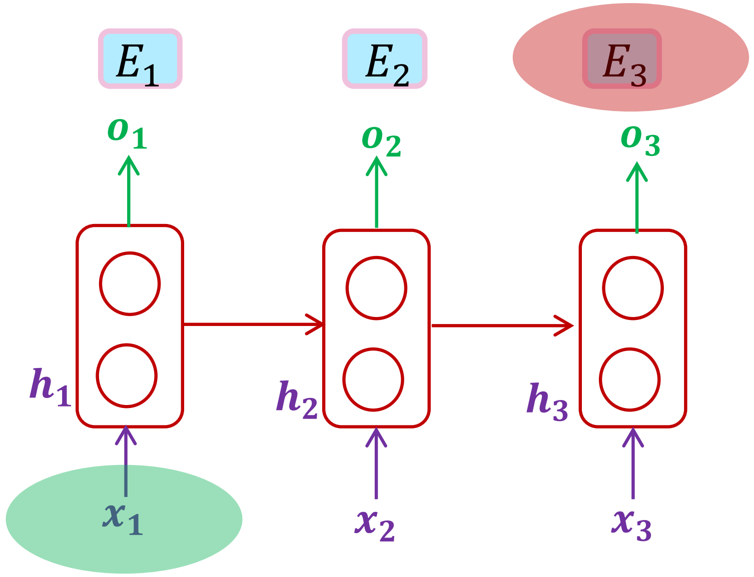


Direction of Forward pass

Backpropogation through time (BPTT) in RNN

➢ Training recurrent networks via BPTT

The **output** at time t=T is **dependent** on the inputs from **t=T to t=1**

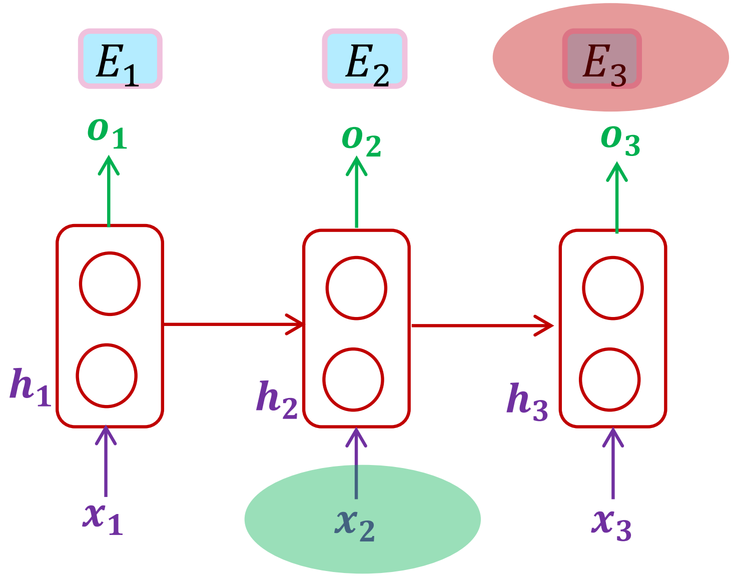


Direction of Forward pass

Backpropogation through time (BPTT) in RNN

- Training recurrent networks via BPTT

The **output** at time t=T is **dependent** on the inputs from **t=T to t=1**

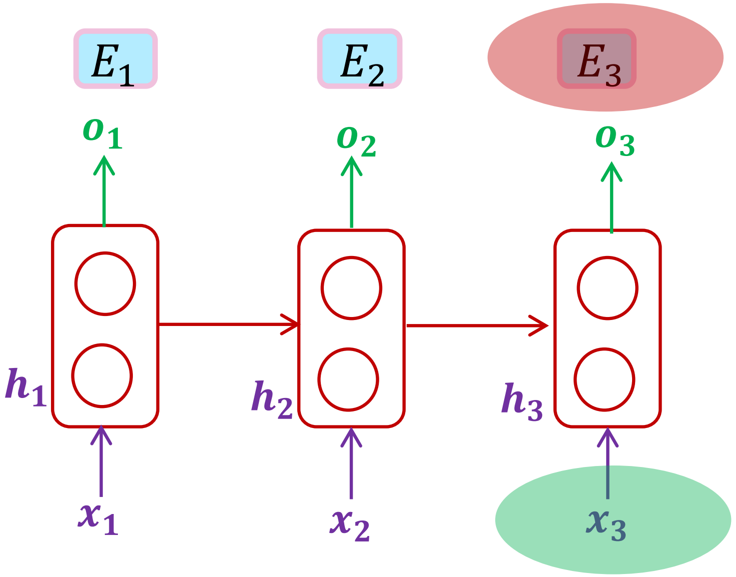


Direction of Forward pass

Backpropogation through time (BPTT) in RNN

- Training recurrent networks via BPTT

The **output** at time t=T is **dependent** on the inputs from **t=T to t=1**

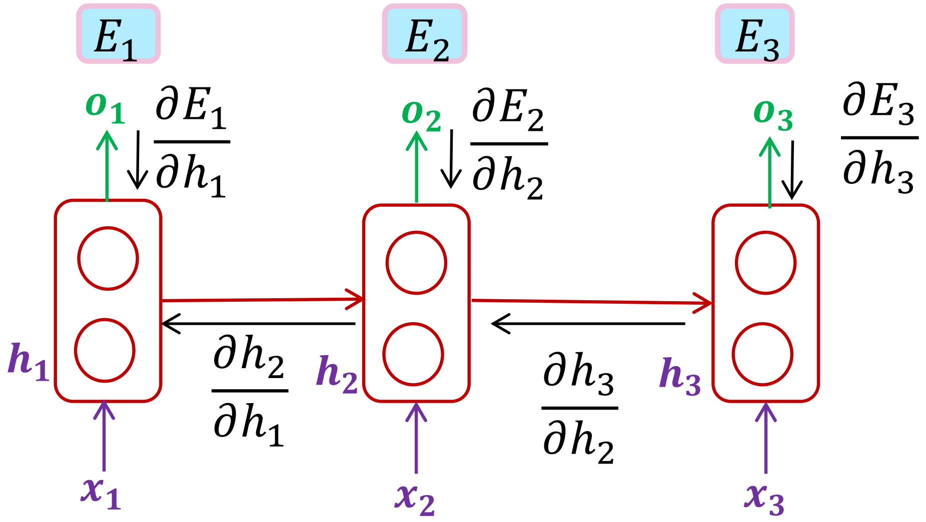


Direction of Forward pass

Backpropogation through time (BPTT) in RNN

- Training recurrent networks via BPTT

The **output** at time t=T is **dependent** on the inputs from **t=T to t=1**



Direction of *Backward* pass (via partial derivatives)
--- gradient flow ---

Backpropogation through time (BPTT) in RNN

➢ Training recurrent networks via BPTT

The **output** at time t=T is **dependent** on the inputs from **t=T to t=1**

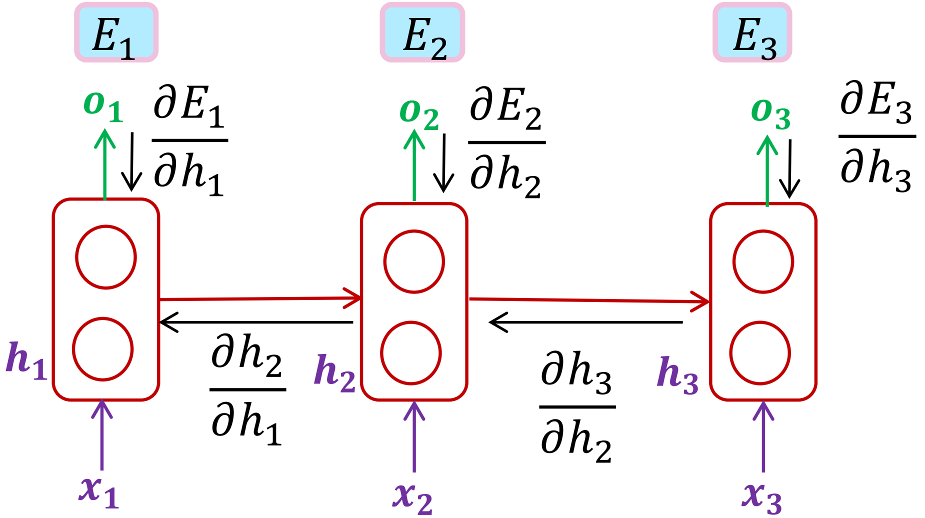
➢ Let us take our loss/error function to be cross entropy:

$$E_t(o_t', o_t) = -o_t' \log o_t$$

$$E(o_t', o_t) = \sum_t E_t(o_t', o_t)$$

$$E(o_t', o_t) = -\sum_t o_t' \log o_t$$

Where o_t' are the truth values



Direction of *Backward* pass (via partial derivatives)
--- gradient flow ---

Backpropogation through time (BPTT) in RNN

The **output** at time t=3 is **dependent** on the inputs from **t=3 to t=1**

Writing gradients in a sum-of-products form

$$\frac{\partial E}{\partial \theta} = \sum_{1 \le t \le 3} \frac{\partial E_t}{\partial \theta}$$

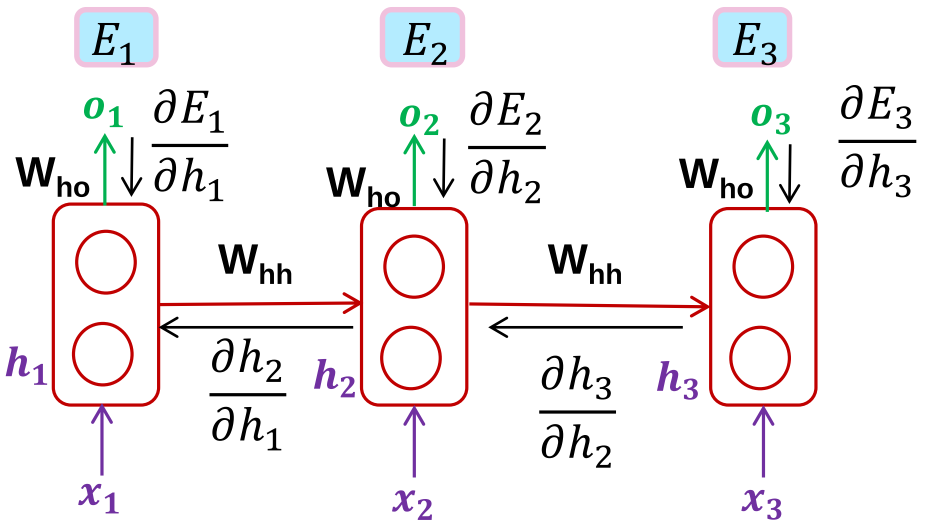
$$\frac{\partial E_3}{\partial W_{ho}} = \frac{\partial E_3}{\partial o_3}\frac{\partial o_3}{\partial W_{ho}} = \frac{\partial E_3}{\partial o_3}\frac{\partial o_3}{\partial z_3}\frac{\partial z_3}{\partial W_{ho}}$$

$where, z_3 = W_{ho} h_3$ i.e., o_3 with softmax

$$\frac{\partial E_3}{\partial W_{ho}} = {o_3}'(o_3 - 1) \times (h_3)$$

$where, \quad \times = outer\ product$

$$\boxed{\frac{\partial E_3}{\partial W_{ho}}\ depends\ only\ on\ o_3\ , o_3'\ and\ h_3}$$



Direction of *Backward* pass (via partial derivatives)
--- gradient flow ---

Backpropogation through time (BPTT) in RNN

offline

$$\frac{\partial E_3}{\partial W_{ho}} = {o_3}'(o_3 - 1) \times (h_3)$$

How ?

Proof

$$E_3({o_3}', o_3) = -{o_3}' \log o_3$$

$$o_3 = softmax(z_3), \qquad and\ z_3 = W_{ho} h_3$$

$$\frac{\partial E_3}{\partial z_3} = -{o_3}' \frac{\partial \log(o_3)}{\partial z_3}$$

$$o_3 = \frac{1}{\Omega} e^{z_3} \text{ and }, \Omega = \sum_i e^{z_i} \quad \log(o_3) = z_3 - \log(\Omega)$$

$$\frac{\partial \log(o_3)}{\partial z_3} = 1 - \frac{1}{\Omega} \frac{\partial \Omega}{\partial z_3}$$

$$\frac{\partial \Omega}{\partial z_3} = \sum_i e^{zi} \delta_{i3} = e^{zk}$$

$$\frac{\partial \log(o_3)}{\partial z_3} = 1 - o_3$$

$$\frac{\partial o_3}{\partial z_3} = o_3(1 - o_3)$$

$$\frac{\partial E_3}{\partial z_3} = -{o_3}'(1 - o_3) = o_3'(o_3 - 1)$$

$$\boxed{\frac{\partial E_3}{\partial W_{ho}} = \frac{\partial E_3}{\partial o_3} \frac{\partial o_3}{\partial z_3} \frac{\partial z_3}{\partial W_{ho}} = \frac{\partial E_3}{\partial z_3} \frac{\partial z_3}{\partial W_{ho}} = o_3'(o_3 - 1) \times (h_3)}$$

1. http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/
2. https://stats.stackexchange.com/questions/235528/backpropagation-with-softmax-cross-entropy

Backpropogation through time (BPTT) in RNN

The **output** at time t=3 is **dependent** on the inputs from **t=3 to t=1**

Writing gradients in a sum-of-products form

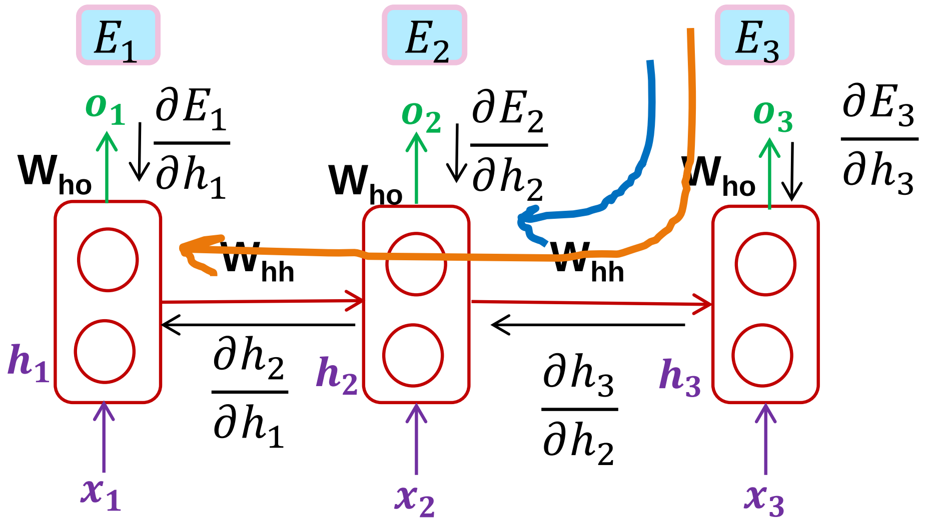
$$\frac{\partial E}{\partial \theta} = \sum_{1 \le t \le 3} \frac{\partial E_t}{\partial \theta} \qquad \frac{\partial E_3}{\partial \mathrm{W}_{hh}} = \frac{\partial E_3}{\partial h_3} \frac{\partial h_3}{\partial \mathrm{W}_{hh}}$$

Since h_3 depends on h_2 **and** h_2 depends on h_1, therefore

$$\frac{\partial E_3}{\partial \mathrm{W}_{hh}} = \sum_{k=1}^{3} \frac{\partial E_3}{\partial h_3} \frac{\partial h_3}{\partial h_k} \frac{\partial h_k}{\partial \mathrm{W}_{hh}}$$

e.g., $\frac{\partial h_3}{\partial \mathrm{h}_1} = \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1}$

In general, $$\frac{\partial E_t}{\partial \mathrm{W}_{hh}} = \sum_{1 \le k \le t} \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial \mathrm{W}_{hh}}$$



Direction of *Backward* pass (via partial derivatives)
--- gradient flow ---

Backpropogation through time (BPTT) in RNN

The **output** at time t=3 is **dependent** on the inputs from **t=3 to t=1**

Writing gradients in a sum-of-products form

$$\frac{\partial E}{\partial \theta} = \sum_{1 \le t \le 3} \frac{\partial E_t}{\partial \theta} \qquad \frac{\partial E_3}{\partial W_{hh}} = \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial W_{hh}}$$

Since h_3 depends on h_2 **and** h_2 depends on h_1, therefore

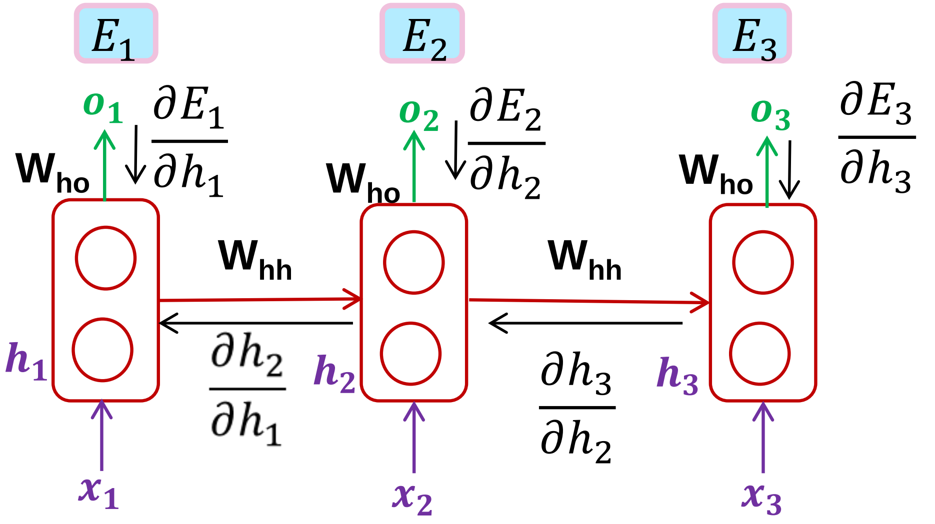
$$\frac{\partial E_3}{\partial W_{hh}} = \sum_{k=1}^{3} \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_k}\frac{\partial h_k}{\partial W_{hh}} \quad \text{e.g.,} \quad \frac{\partial h_3}{\partial h_1} = \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}$$

In general,

$$\frac{\partial E_t}{\partial W_{hh}} = \sum_{1 \le k \le t} \frac{\partial E_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W_{hh}}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{t \ge i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \ge i > k} {W_{hh}}^T diag[g'(h_{i-1})]$$

Jacobian matrix $\frac{\partial h_t}{\partial h_k}$



Direction of *Backward* pass (via partial derivatives)
--- gradient flow ---

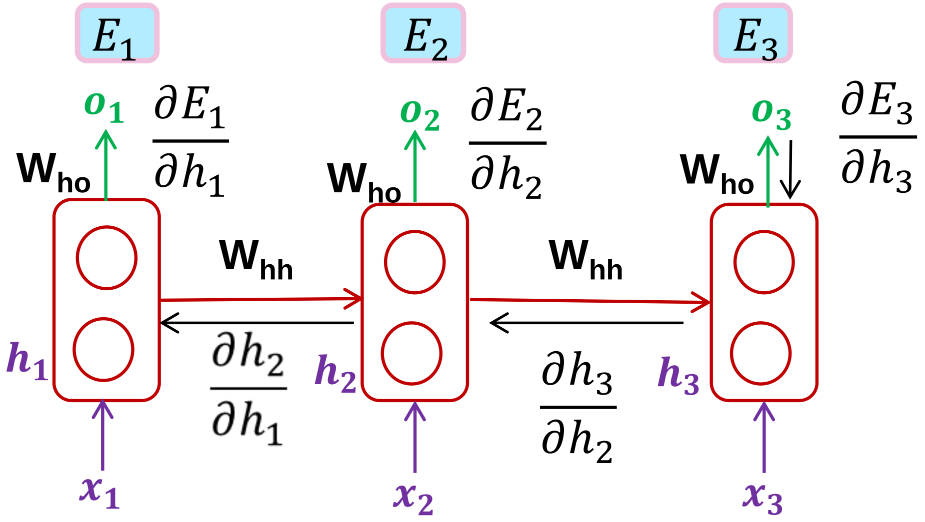
Transport error in time from step t back to step k

Backpropogation through time (BPTT) in RNN

The **output** at time t=3 is **dependent** on the inputs from **t=3 to t=1**

Writing gradients in a sum-of-products form

$$\frac{\partial E}{\partial \theta} = \sum_{1 \le t \le 3} \frac{\partial E_t}{\partial \theta} \qquad \frac{\partial E_3}{\partial W_{hh}} = \frac{\partial E_3}{\partial h_3} \frac{\partial h_3}{\partial W_{hh}}$$



Since h_3 depends on h_2 **and** h_2 depends on h_1, therefore

$$\frac{\partial E_3}{\partial W_{hh}} = \sum_{k=1}^{3} \frac{\partial E_3}{\partial h_3} \frac{\partial h_3}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}} \qquad \text{e.g.,} \quad \frac{\partial h_3}{\partial h_1} = \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1}$$

In general, $$\frac{\partial E_t}{\partial W_{hh}} = \sum_{1 \le k \le t} \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}}$$

Weight matrix

Direction of *Backward* pass (via partial derivatives)

--- gradient flow ---

$$\frac{\partial h_t}{\partial h_k} = \prod_{t \ge i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \ge i > k} W_{hh}^{T} \, diag[g'(h_{i-1})]$$

Derivative of activation function

Transport error in time from step t back to step k

Jacobian matrix $\frac{\partial h_t}{\partial h_k}$

Backpropogation through time (BPTT) in RNN

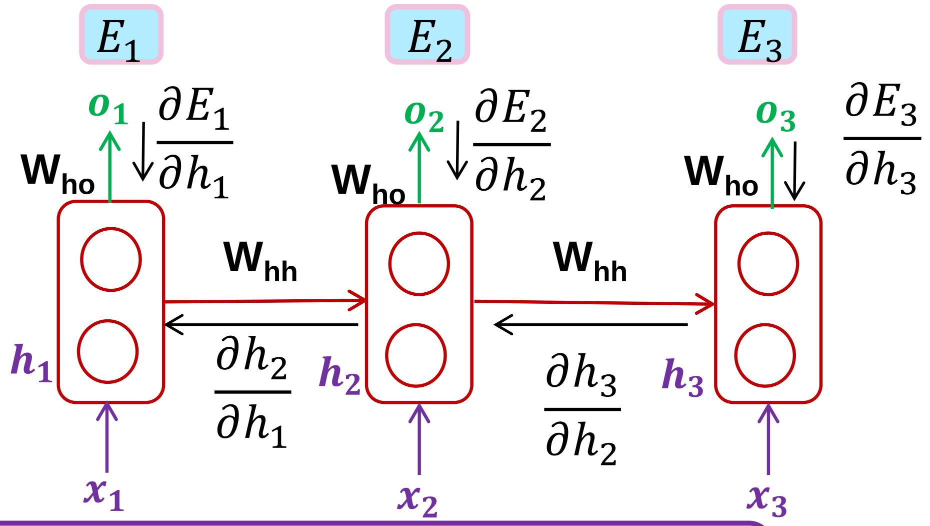
The **output** at time t=3 is **dependent** on the inputs from **t=3 to t=1**

Writing gradients in a sum-of-products form

$$\frac{\partial E}{\partial \theta} = \sum_{1 \le t \le 3} \frac{\partial E_t}{\partial \theta} \qquad \frac{\partial E_3}{\partial W_{hh}} = \frac{\partial E_3}{\partial h_3} \frac{\partial h_3}{\partial W_{hh}}$$

Since h_3 depends on h_2 **and** h_2 depends on h_1, therefore

$$\frac{\partial E_3}{\partial W_{hh}} = \sum_{k=1}^{3} \frac{\partial E_3}{\partial h_3} \frac{\partial h_3}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}} \quad \text{e.g.,} \quad \frac{\partial h_3}{\partial h_1} = \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1}$$



Repeated matrix multiplications leads to vanishing and exploding gradients

erivatives)

$1 \le k \le t$

$$\frac{\partial h_t}{\partial h_k} = \prod_{t \ge i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \ge i > k} W_{hh}^T \, diag[g'(h_{i-1})]$$

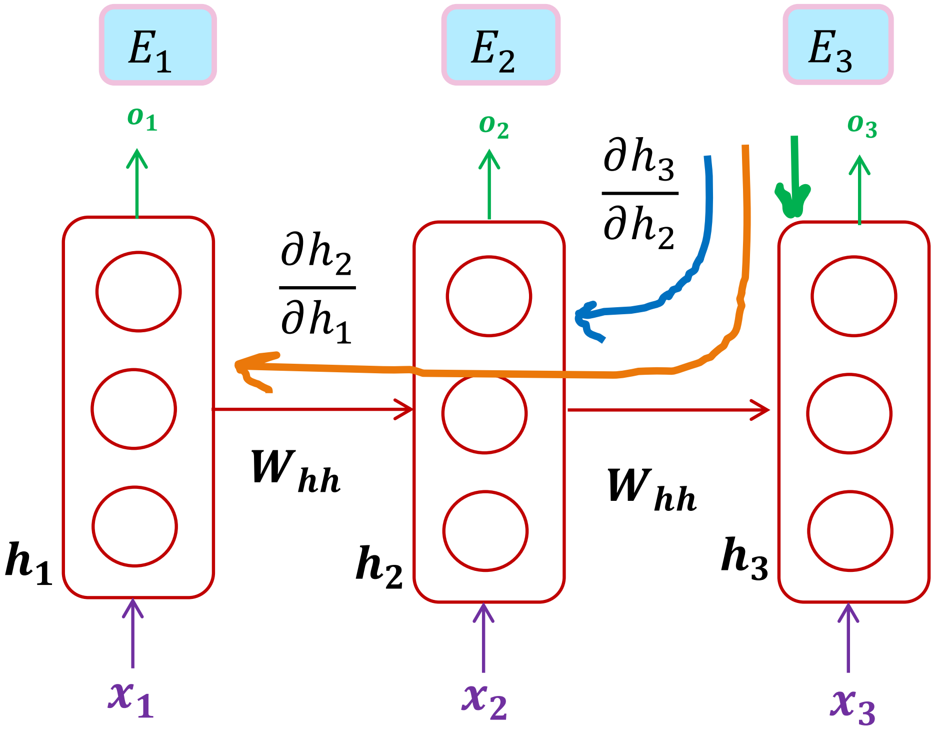
Transport error in time from step t back to step k

Jacobian matrix $\frac{\partial h_t}{\partial h_k}$

BPTT: Gradient Flow

$$\frac{\partial E_3}{\partial W_{hh}} = \sum_{k=1}^{3} \frac{\partial E_3}{\partial h_3} \frac{\partial h_3}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}}$$

$$= \frac{\partial E_3}{\partial o_3} \frac{\partial o_3}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial W_{hh}} + \frac{\partial E_3}{\partial o_3} \frac{\partial o_3}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial W_{hh}} + \frac{\partial E_3}{\partial o_3} \frac{\partial o_3}{\partial h_3} \frac{\partial h_3}{\partial h_3} \frac{\partial h_3}{\partial W_{hh}}$$



Backpropogation through time (BPTT) in RNN

offline

Code snippet for forward-propagation is shown below (Before going for BPTT code)

```
def forward_propagation(self, x):
    # The total number of time steps
    T = len(x)
    # During forward propagation we save all hidden states in s because need them later.
    # We add one additional element for the initial hidden, which we set to 0
    h = np.zeros((T + 1, self.hidden_dim))
    h[-1] = np.zeros(self.hidden_dim)
    # The outputs at each time step. Again, we save them for later.
    o = np.zeros((T, self.word_dim))
    # For each time step...
    for t in np.arange(T):
        # Note that we are indxing W_xh by x[t]. This is the same as multiplying W_xh with a one-hot vector.
        h[t] = np.tanh(self.W_xh[:,x[t]] + self.W_hh.dot(h[t-1]))
        o[t] = softmax(self.W_xo.dot(h[t]))
    return [o, h]
```

https://cs224d.stanford.edu/lectures/CS224d-Lecture8.pdf

Backpropogation through time (BPTT) in RNN

Code snippet for backpropagation w.r.t. time is shown below

```
def back_prop_through_time(self, x, y):
    T=len(y)
    #perform a forward propogation
    o,h = self.forward_propogation(x)

    #Following variables accumulate gradient
    #wrt W_hh, W_xx and W_xy
    dLdW_xh=np.zeros(self.W_xh.shape)
    dLdW_hy=np.zeros(self.W_hy.shape)
    dLdW_hh=np.zeros(self.W_hh.shape)
    delta_o= 0
    delta_o[np.arrange(len(y)),y]=1

    #For each output backwards...

    for t in np.arrange(T)[::-1]:
        dLdW_hy += np.outer(delta_o[t],h[t].T)

        # Next for calculating backpropogation through time, i.e.
        #dL_dW_hh, we need delta_t
        #Initial Delta Calculation: dL_dz
        delta_t= self.W_hy.T.dot(delta_o[t])*(1-(h[t]**2))

        #Backpropogation through time for (atmost self.bptt_truncate_steps)
        # Have a look at Equation-2 in the slides
        for bptt_step in np.arrange(max(0,t-self.bptt.truncate),t+1)[::-1]:
            # print Backpropagation step
            #Add the gradient to each previous step
            dLdW_hh += np.outer(delta_t, h[bptt_step-1])
            dLdW_xh[:,x[bptt_step]] += delta_t

            #Update delta for the next time step dL/dz at t-1
            delta_t =self.W_hh.T.dot(delta_t)*(1-h[bptt_step_1]**2)
    return [dLdW_xh,dLdW_hy, dLdW_hh]
```

offline

$$\frac{\partial E_t}{\partial W_{hh}} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W_{hh}}$$

$$= \cdots + \frac{\partial E_t}{\partial o_t}\frac{\partial o_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}}\frac{\partial h_{t-2}}{\partial W_{hh}} + \frac{\partial E_t}{\partial o_t}\frac{\partial o_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial W_{hh}} + \frac{\partial E_t}{\partial o_t}\frac{\partial o_t}{\partial h_t}\frac{\partial h_t}{\partial W_{hh}}$$

$$\frac{\partial E_t}{\partial W_{ho}} = -(o_t - o_t{}')(h_t)$$

$$\frac{\partial E_t}{\partial h_t} = -(o_t - o_t{}')W_{ho} \ and \ \frac{\partial h_t}{\partial W_{hh}} = (1 - h_t^2)(h_{t-1})$$

$$\frac{\partial E}{\partial h_t}\frac{\partial h_t}{\partial W_{hh}} = \underbrace{-(o_t - o_t{}')W_{ho}(1 - h_t^2)}_{\textit{Initial delta\_t}}(h_{t-1}) \longrightarrow \mathbf{A}$$

$$\frac{\partial E_t}{\partial o_t}\frac{\partial o_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial W_{hh}} = \underbrace{-(o_t - o_t')(W_{ho})[1 - h_t^2](W_{hh})(1 - h_{t-1}^2)}_{\textit{delta\_t}}(h_{t-2}) \rightarrow \mathbf{B}$$

$$\frac{\partial E}{\partial W_{hh}} = \mathrm{A} + \mathrm{B} + \cdots \text{(till the end of dependency)}$$

Break (10 minutes)

Challenges in Training an RNN: Vanishing Gradients

Short Term Dependencies

→ need recent information to perform the present task.

For example in a language model, predict the next word based on the previous ones.

“the clouds are in the ***?****”* → *‘sky’*

→ Easier to predict ‘sky’ given the context, i.e., *short term dependency* → (vanilla) **RNN Good so far.**

Long Term Dependencies

→ Consider longer word sequence “I grew up in France…..........……………………. I speak fluent *French*.”

→ Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back.

→ As the gap increases → **practically difficult for RNN to learn from the past information**

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Challenges in Training an RNN: Vanishing Gradients

Long Term dependencies

Assume an RNN of 5 time steps:

Let's look at the **Jacobian matrix** while BPTT:

$$\frac{\partial E_5}{\partial \theta} = \frac{\partial E_5}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial \theta} + \frac{\partial E_5}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial \theta} + \frac{\partial E_5}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial \theta} + \dots$$

$$A = \begin{bmatrix} -1.70\text{e}{-10} & 4.94\text{e}{-10} & 2.29\text{e}{-10} \\ -1.73\text{e}{-10} & 5.56\text{e}{-10} & 2.55\text{e}{-10} \\ -1.81\text{e}{-10} & 4.40\text{e}{-10} & 2.08\text{e}{-10} \end{bmatrix}$$

$\|A\| =$ **1.00e−09**

$$B = \begin{bmatrix} -1.13\text{e}{-08} & 2.61\text{e}{-09} & 1.50\text{e}{-08} \\ -1.11\text{e}{-08} & 5.70\text{e}{-09} & 1.51\text{e}{-08} \\ -1.33\text{e}{-08} & 9.11\text{e}{-09} & 1.83\text{e}{-08} \end{bmatrix}$$

$\|B\| =$ **1.53e−07**

$$C = \begin{bmatrix} -1.70\text{e}{-06} & 8.70\text{e}{-06} & 9.40\text{e}{-06} \\ -2.51\text{e}{-07} & 7.30\text{e}{-06} & 8.98\text{e}{-06} \\ 7.32\text{e}{-07} & 7.85\text{e}{-06} & 1.05\text{e}{-05} \end{bmatrix}$$

$\|C\| =$ **2.18e−05**

- $\frac{\partial E_5}{\partial \theta}$ is dominated by short-term dependencies(e.g., C), but
- Gradient **vanishes** in long-term dependencies i.e. $\frac{\partial E_5}{\partial \theta}$ is updated much less due to A as compared to updated by C

Long Term dependencies

Assume an RNN of 5 time steps:

Let‘s look at the **Jacobian matrix** while BPTT:

$$\frac{\partial E_5}{\partial \theta} = \frac{\partial E_5}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial \theta} + \frac{\partial E_5}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial \theta} + \frac{\partial E_5}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial \theta} + \ldots$$

$$A = \begin{bmatrix} -1.70e{-10} & 4.94e{-10} & 2.29e{-10} \\ -1.73e{-10} & 5.56e{-10} & 2.55e{-10} \\ -1.81e{-10} & 4.40e{-10} & 2.08e{-10} \end{bmatrix}$$

$\|\boldsymbol{A}\| =$ **1.00e−09**

$$B = \begin{bmatrix} -1.13e{-08} & 2.61e{-09} & 1.50e{-08} \\ -1.11e{-08} & 5.70e{-09} & 1.51e{-08} \\ -1.33e{-08} & 9.11e{-09} & 1.83e{-08} \end{bmatrix}$$

$\|\boldsymbol{B}\| =$ **1.53e−07**

$$C = \begin{bmatrix} -1.70e{-06} & 8.70e{-06} & 9.40e{-06} \\ -2.51e{-07} & 7.30e{-06} & 8.98e{-06} \\ 7.32e{-07} & 7.85e{-06} & 1.05e{-05} \end{bmatrix}$$

$\|\boldsymbol{C}\| =$ **2.18e−05**

- $\frac{\partial E_5}{\partial \theta}$ is dom
- Gradient A as compared to updated by C

Long Term Components goes exponentially fast to norm 0
→ no correlation between temporally distant events

Challenges in Training an RNN: Exploding Gradients

Assume an RNN of 5 time steps:

Let's look at the **Jacobian matrix** while BPTT:

Long Term dependencies

$$\frac{\partial E_5}{\partial \theta} = \frac{\partial E_5}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial \theta} + \frac{\partial E_5}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial \theta} + \frac{\partial E_5}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial \theta} + \ \ldots$$

$$A = \begin{bmatrix} -1.70e+10 & 4.94e+10 & 2.29e-10 \\ -1.73e+10 & 5.56e+10 & 2.55e-10 \\ -1.81e+10 & 4.40e+10 & 2.08e-10 \end{bmatrix}$$

$\|A\|$ = **1.00e+109**

$$B = \begin{bmatrix} -1.13e+08 & 2.61e+09 & 1.50e+08 \\ -1.11e+08 & 5.70e+09 & 1.51e+08 \\ -1.33e+08 & 9.11e+09 & 1.83e+08 \end{bmatrix}$$

$\|B\|$ = **1.53e+107**

$$C = \begin{bmatrix} -1.70e+06 & 8.70e+06 & 9.40e+06 \\ -2.51e+07 & 7.30e+06 & 8.98e+06 \\ 7.32e+07 & 7.85e+06 & 1.05e+05 \end{bmatrix}$$

$\|C\|$ = **2.18e+105**

- ➢ $\frac{\partial E_5}{\partial \theta}$, gradient explodes, i.e., **NaN** due to very large numbers

Challenges in Training an RNN: Exploding Gradients

Assume an RNN of 5 time steps:

Let‘s look at the **Jacobian matrix** while BPTT:

Long Term dependencies

$$\frac{\partial E_5}{\partial \theta} = \frac{\partial E_5}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial \theta} + \frac{\partial E_5}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial \theta} + \frac{\partial E_5}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial \theta} + \ldots$$

$$A = \begin{bmatrix} -1.70e+10 & 4.94e+10 & 2.29e{-10} \\ -1.73e+10 & 5.56e+10 & 2.55e{-10} \\ -1.81e+10 & 4.40e+10 & 2.08e{-10} \end{bmatrix}$$

$\|A\|$ = **1.00e+109**

$$B = \begin{bmatrix} -1.13e+06 & 2.61e+06 & 1.50e+06 \\ -1.11e+06 & 5.70e+06 & 1.51e+06 \\ -1.33e+06 & 9.11e+06 & 1.83e+06 \end{bmatrix}$$

$\|B\|$ = **1.53e+97**

$$C = \begin{bmatrix} -1.70e+04 & 8.70e+04 & 9.40e+04 \\ -2.51e+04 & 7.30e+04 & 8.98e+04 \\ 7.32e+04 & 7.85e+04 & 1.05e+04 \end{bmatrix}$$

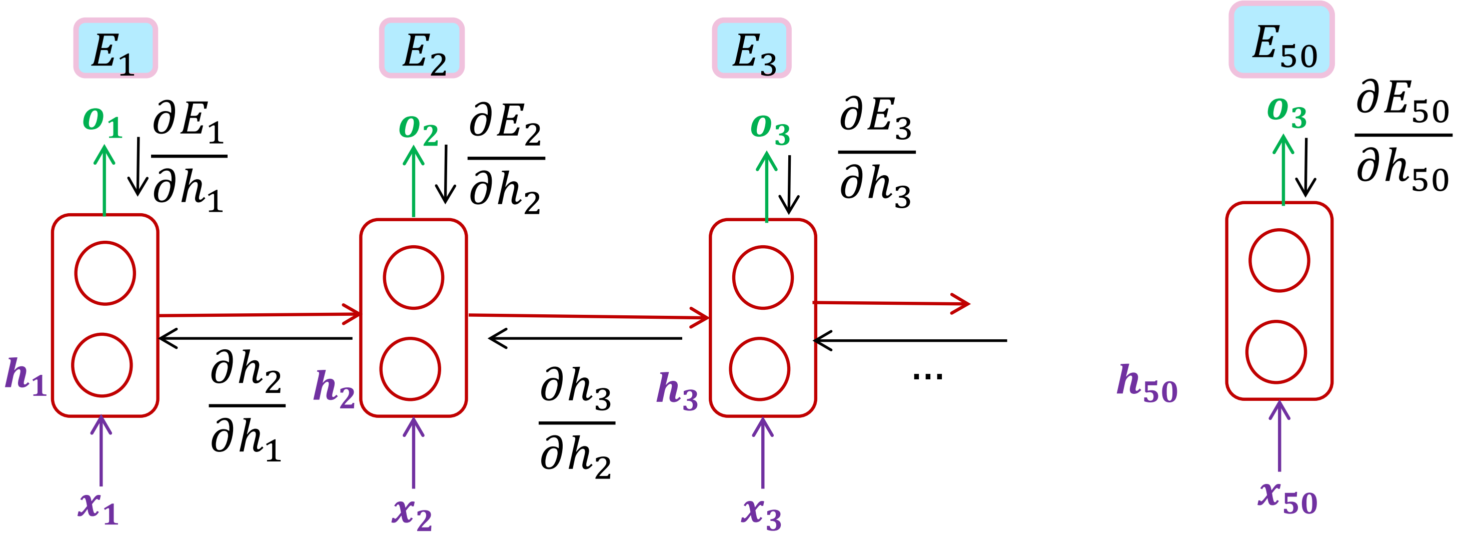
$\|C\|$ = **2.18e+85**

- $\frac{\partial E_5}{\partial \theta}$, gra

Large increase in the norm of the gradient during training → due to explosion of long term components

Vanishing Gradient in Long-term Dependencies

Often, the length of sequences are long….e.g., documents, speech, etc.



In practice as the **length** of the sequence **increases**, the probability of **training** being successful **decrease** drastically.

? **Why**

❓ Why

Let us look at the recurrent part of our RNN equation:

$$h_t = g_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}\, h_{t-1} + W_{xh} x_t)$$

$$o_t = softmax(W_{ho} h_t)$$

tanh Expansion

$$h_t = W_{hh} \mathrm{f}(h_{\mathrm{t}-1}) + \text{some other terms}$$

$$h_t = W_{hh} h_0 + \text{some other terms}$$

Vanishing Gradient in Long-term Dependencies

Writing gradients in a sum-of-products form

$$\frac{\partial E}{\partial \theta} = \sum_{1 \le t \le 3} \frac{\partial E_t}{\partial \theta} \qquad \frac{\partial E_3}{\partial W_{hh}} = \frac{\partial E_3}{\partial h_3} \frac{\partial h_3}{\partial W_{hh}}$$

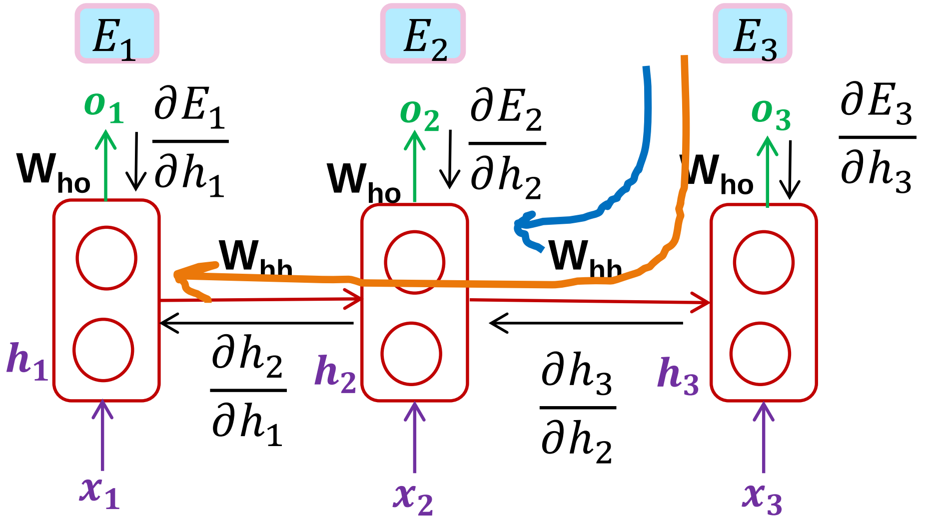
Since h_3 depends on h_2 **and** h_2 depends on h_1, therefore

$$\frac{\partial E_3}{\partial W_{hh}} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial h_3} \frac{\partial h_3}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}} \quad \text{e.g.,} \quad \frac{\partial h_3}{\partial h_1} = \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1}$$

In general, $$\frac{\partial E_t}{\partial W_{hh}} = \sum_{1 \le k \le t} \frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}}$$

$$\frac{\partial h_t}{\partial h_k} = \underbrace{\prod_{t \ge i > k} \frac{\partial h_i}{\partial h_{i-1}}}_{\text{This term is the product of Jacobian matrix .}} = \prod_{t \ge i > k} W_{hh}{}^{T} diag[g'(h_{i-1})]$$

Jacobian matrix $\frac{\partial h_t}{\partial h_k}$



Direction of *Backward* pass (via partial derivatives)
--- gradient flow ---

$h_t = W_{hh} f(h_{t-1}) + \text{some terms}$

Transport error in time from step t back to step k

Vanishing Gradient in Long-term Dependencies

Writing gradients in a sum-of-products form

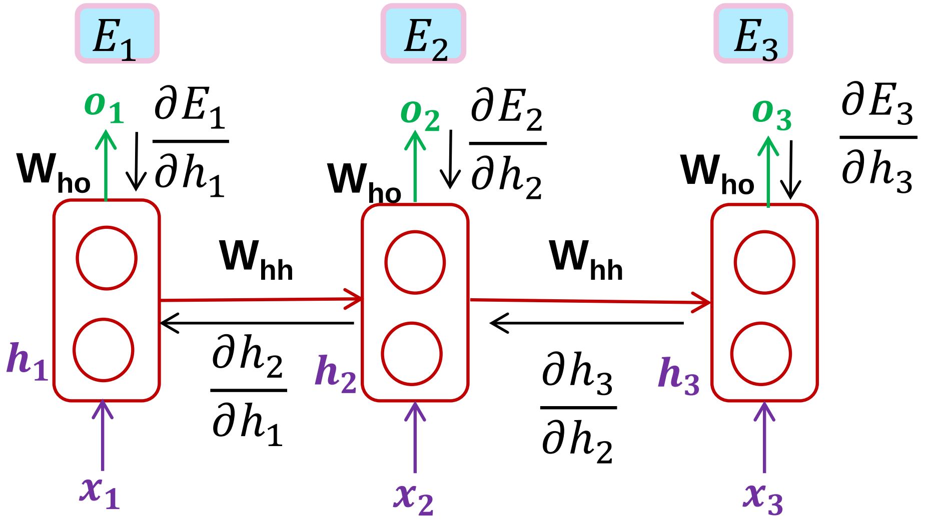
$$\frac{\partial E}{\partial \theta} = \sum_{1 \le t \le 3} \frac{\partial E_t}{\partial \theta}$$

$$\frac{\partial E_3}{\partial W_{hh}} = \frac{\partial E_3}{\partial h_3} \frac{\partial h_3}{\partial W_{hh}}$$

$$\frac{\partial E_t}{\partial W_{hh}} = \sum_{1 \le k \le t} \frac{\partial E_3}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{t \ge i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \ge i > k} W_{hh}{}^T diag[g'(h_{i-1})]$$

Jacobian matrix $\frac{\partial h_t}{\partial h_k}$



Direction of *Backward* pass (via partial derivatives)
--- gradient flow ---

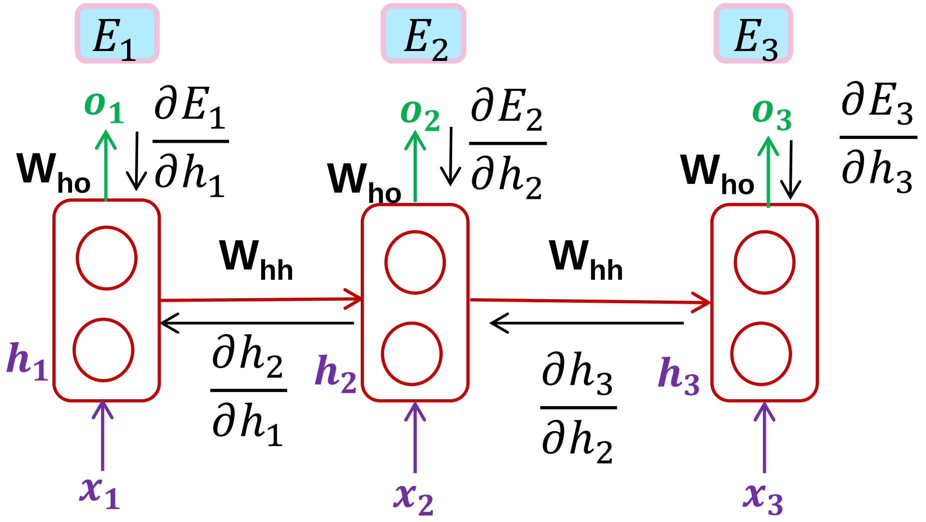
Vanishing Gradient in Long-term Dependencies

Writing gradients in a sum-of-products form

$$\frac{\partial E}{\partial \theta} = \sum_{1\le t\le 3} \frac{\partial E_t}{\partial \theta} \qquad \frac{\partial E_3}{\partial W_{hh}} = \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial W_{hh}}$$

$$\frac{\partial E_t}{\partial W_{hh}} = \sum_{1\le k\le t} \frac{\partial E_3}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W_{hh}}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{t\ge i>k}\frac{\partial h_i}{\partial h_{i-1}} = \prod_{t\ge i>k} W_{hh}^T\, diag[g'(h_{i-1})]$$



Direction of *Backward* pass (via partial derivatives)
--- gradient flow ---

Repeated matrix multiplications leads to vanishing gradients !!!

Mechanics behind Vanishing and Exploding Gradients

$$\frac{\partial h_t}{\partial \mathrm{h}_k} = \prod_{t \geq i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \geq i > k} \mathrm{W}_{hh}^{T} diag[g'(h_{i-1})]$$

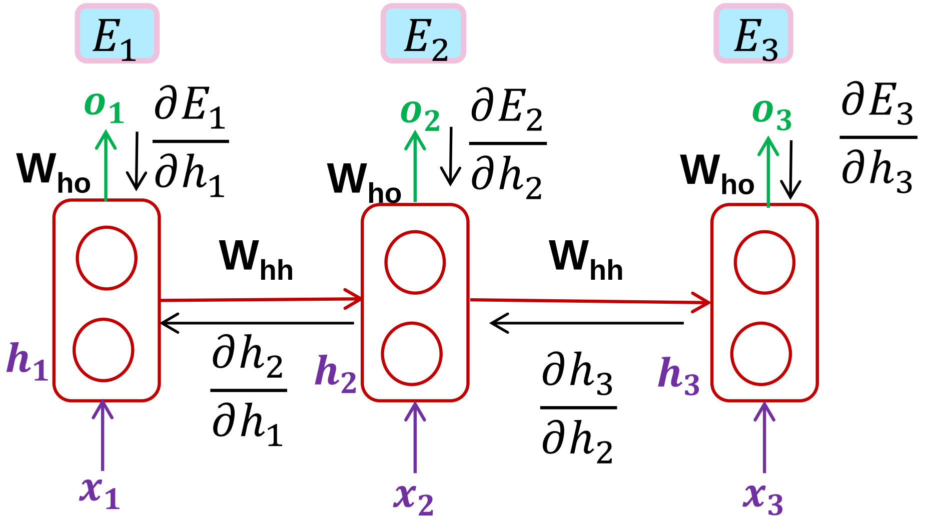
Consider identity activation function

If recurrent matrix W_{hh} is a diagonalizable:

$$\boldsymbol{W_{hh} = Q^{-1} * \nabla * Q}$$

matrix composed of eigenvectors of W_{hh}

diagonal matrix with eigenvalues placed on the diagonals



Direction of *Backward* pass (via partial derivatives)
--- gradient flow ---

Using power iteration method, computing powers of W_{hh} :

$$\boldsymbol{W_{hh}^{n} = Q^{-1} * \nabla^{n} * Q}$$

Bengio et al, "On the difficulty of training recurrent neural networks." (2012)

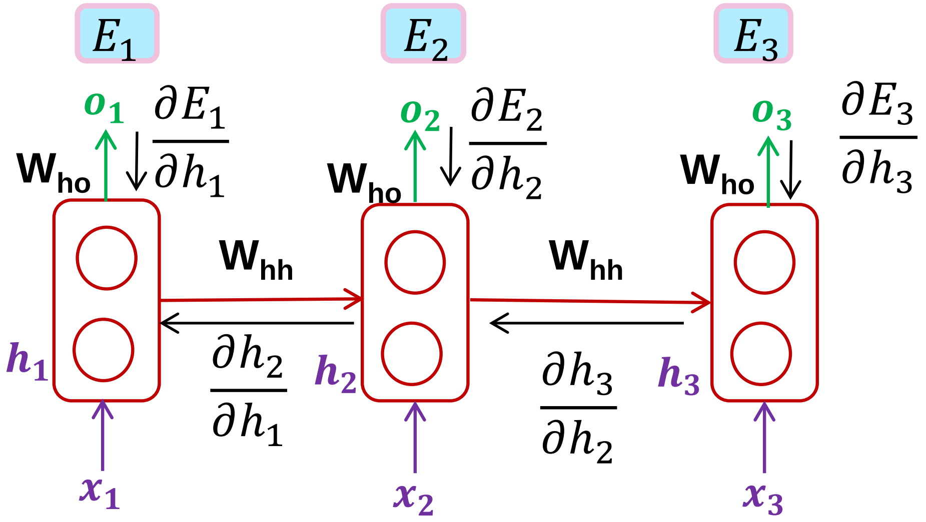
Mechanics behind Vanishing and Exploding Gradients

$$\frac{\partial h_t}{\partial \mathrm{h}_k} = \prod_{t \geq i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \geq i > k} W_{hh}^T diag[g'(h_{i-1})]$$

Consider identity activation function

computing powers of W_{hh} :

$$\boldsymbol{W_{hh}^{n} = Q^{-1} * \nabla^{n} * Q}$$



Direction of *Backward* pass (via partial derivatives)

Vanishing gradients

$$\boldsymbol{\nabla} = \begin{bmatrix} -0.618 & 0 \\ 0 & 1.618 \end{bmatrix} \quad \Rightarrow \quad \boldsymbol{\nabla}^{10} = \begin{bmatrix} \boxed{-0.0081} & 0 \\ 0 & \boxed{122.99} \end{bmatrix}$$

Eigen values on the diagonal

Exploding gradients

Bengio et al, "On the difficulty of training recurrent neural networks." (2012)

Mechanics behind Vanishing and Exploding Gradients

$$\frac{\partial h_t}{\partial \mathrm{h}_k} = \prod_{t \geq i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \geq i > k} \mathrm{W}_{hh}^T \, diag[g'(h_{i-1})]$$

Consider identity activation function

Need for tight conditions on eigen values during training to prevent gradients to vanish or explode

computing powers of W_{hh} :

$$\boldsymbol{W_{hh}^{n} = Q^{-1} * \nabla^{n} * Q}$$

Vanishing gradients

$$\boldsymbol{\nabla} = \begin{bmatrix} -0.618 & 0 \\ 0 & 1.618 \end{bmatrix} \Rightarrow \boldsymbol{\nabla}^{\mathbf{10}} = \begin{bmatrix} \boxed{-0.0081} & 0 \\ 0 & \boxed{122.99} \end{bmatrix}$$

Eigen values on the diagonal

Exploding gradients

Bengio et al, "On the difficulty of training recurrent neural networks." (2012)

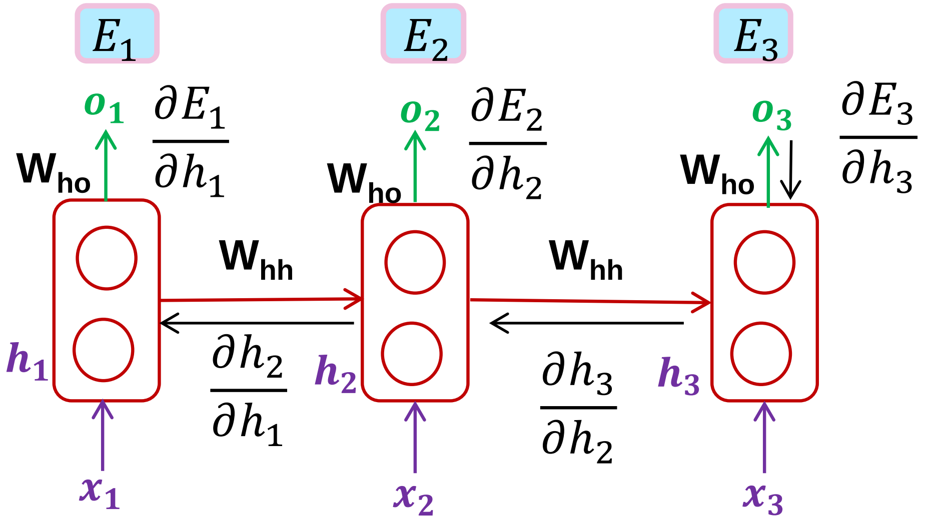
Mechanics behind Vanishing and Exploding Gradients

Writing gradients in a sum-of-products form

$$\frac{\partial E}{\partial \theta} = \sum_{1 \le t \le 3} \frac{\partial E_t}{\partial \theta} \qquad \frac{\partial E_3}{\partial W_{hh}} = \frac{\partial E_3}{\partial h_3} \frac{\partial h_3}{\partial W_{hh}}$$

$$\frac{\partial E_t}{\partial W_{hh}} = \sum_{1 \le k \le t} \frac{\partial E_3}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{t \ge i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \ge i > k} W_{hh}{}^{T} diag[g'(h_{i-1})]$$



Direction of *Backward* pass (via partial derivatives)
--- gradient flow ---

Find Sufficient condition for when gradients vanish → compute an upper bound for $\frac{\partial h_t}{\partial h_k}$ term

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\| \le \left\| W_{hh}{}^{T} \right\| \left\| diag([g'(h_{i-1})]) \right\|$$

→ find out an upper bound for the norm of the jacobian!

Mechanics behind Vanishing and Exploding Gradients

Lets find an upper bound for the term: $\|\boldsymbol{W^T}\|\|\boldsymbol{diag}([\boldsymbol{g}'(\boldsymbol{h_{i-1}})])\|$

Property of matrix norm

- **Proof**: $\|M\|_2 = \sqrt{\lambda_{max}(M^*M)} = \gamma_{max}(\mathrm{M})$

where the spectral norm $\|M\|_2$ of a complex matrix M is defined as $max\{\|Mx\|_2 : \|x\| = 1\}$

offline

The norm of a matrix is equal to the largest singular value of the matrix and is related to the largest Eigen value (spectral radius)

Put $B = M * M$ which is a Hermitian matrix. As a linear transformation of Euclidean vector space E is Hermite iff there exists an orthonormal basis of E consisting of all the eigenvectors of B

Let $\lambda_1, \lambda_2, \lambda_3 \ldots \lambda_n$ be the eigenvalues of B and $\{e_1, e_2 \ldots \ldots . e_n\}$ be an orthonormal basis of E

Let $x = a_1 e_1 + \ldots \ldots a_n e_n$ (linear combination of eigen vectors)

The specttal norm of x:

$$\|x\| = \langle \textstyle\sum_{i=1}^{n} a_i e_i \sum_{i=1}^{n} a_i e_i \rangle^{1/2} = \sqrt{\textstyle\sum_{i=1}^{n} a_i^2}$$

Mechanics behind Vanishing and Exploding Gradients

Using **characteristic equation** to find a matrix's eigenvalues,

$$Bx = B\left(\sum_{i=1}^{n} a_i e_i\right) = \sum_{i=1}^{n} a_i B(e_i) = \sum_{i=1}^{n} \lambda_i a_i e_i$$

offline

Therefore,

$$\|Mx\| = \langle Mx, Mx\rangle = \langle x, M^{*}Mx\rangle = \langle x, Bx\rangle = \left\langle \sum_{i=1}^{n} a_i e_i \sum_{i=1}^{n} \lambda_i a_i e_i \right\rangle = \sqrt{\sum_{i=1}^{n} a_i \overline{\lambda_i a_i}} \leq \max_{(1\leq j\leq n)} \sqrt{\lambda_j} \times (\|x\|)$$

Thus,

If $\|M\| = max\{\|Mx\|: \|x\| = 1\}$, $then$ $\|M\| \leq \max_{1\leq j\leq n} \sqrt{|\lambda_j|}$ equation (1)

Mechanics behind Vanishing and Exploding Gradients

Consider,

$$x_0 = e_{j_0} \Rightarrow \|x\| = 1, so\ that\ \|M\| \geq \langle x, Bx \rangle = \left\langle e_{j_0}, B\left(e_{j_0}\right)\right\rangle = \left\langle e_{j_0}, \lambda_{j_0} e_{j_0}\right\rangle = \sqrt{\left|\lambda_{j_0}\right|}$$ … equation (2)

offline

where, j_0 is the largest eigen value.

Combining (1) and (2) give us $\|M\| = \max_{1 \leq j \leq n} \sqrt{\left|\lambda_j\right|}\ where, \lambda_j\ is\ the\ eigen\ value\ of\ B = M^* M$

Conclusion : $\|M\|_2 = \sqrt{\lambda_{max}(M^* M)} = \gamma_{max}(\mathrm{M})$ …. equation (3)

Remarks:

- **The spectral norm of a matrix** is **equal** to the **largest singular value** of the matrix and is **related** to the **largest Eigen value (spectral radius)**
- If the matrix is square symmetric, the singular value = spectral Radius

Mechanics behind Vanishing and Exploding Gradients

Let's use these properties:

$$\frac{\partial h_t}{\partial \mathrm{h}_k} = \prod_{t \geq i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \geq i > k} \mathrm{W}_{hh}{}^T diag[g'(h_{i-1})]$$

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\| \leq \left\| \mathrm{W}_{hh}{}^T \right\| \| diag([g'(h_{i-1})]) \|$$

Mechanics behind Vanishing and Exploding Gradients

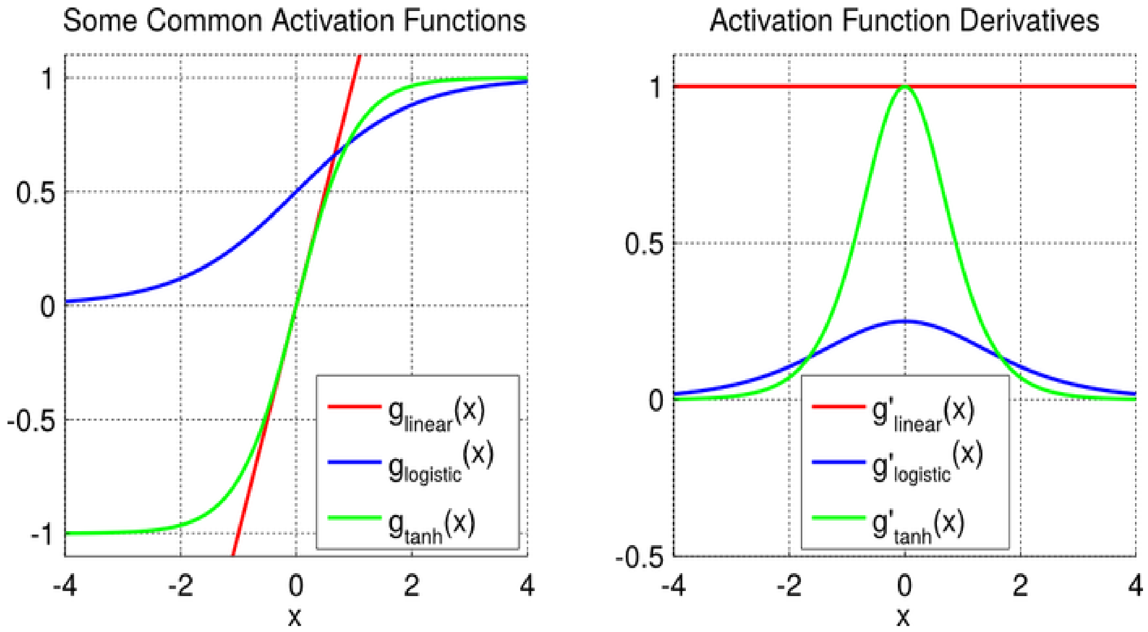
Let's use these properties:

$$\frac{\partial h_t}{\partial \mathrm{h}_k} = \prod_{t \geq i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \geq i > k} \mathrm{W}_{hh}{}^{T} diag[g'(h_{i-1})]$$

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\| \leq \left\| \mathrm{W}_{hh}{}^{T} \right\| \left\| diag\big(g'(h_{i-1})\big) \right\|$$

Gradient of the nonlinear function (sigmoid or tanh) $g'(h_{i-1})$ is bounded by constant, .i.e., $\left\| diag\big(g'(h_{i-1})\big) \right\| \leq \gamma_g$

an upper bound for the norm of the gradient of activation



γ_g = ¼ for sigmoid

γ_g = 1 for tanh

constant

Mechanics behind Vanishing and Exploding Gradients

Let's use these properties:

$$\frac{\partial h_t}{\partial \mathrm{h}_k} = \prod_{t \geq i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \geq i > k} \mathrm{W}_{hh}{}^{T} diag[g'(h_{i-1})]$$

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\| \leq \left\| \mathrm{W}_{hh}{}^{T} \right\| \left\| diag\left(g'(h_{i-1})\right) \right\|$$

Largest Singular value of W_{hh}

$$\leq \boxed{\gamma_W} \gamma_g$$

$\gamma_W \gamma_g =$ **an upper bound for the norm of jacobian!**

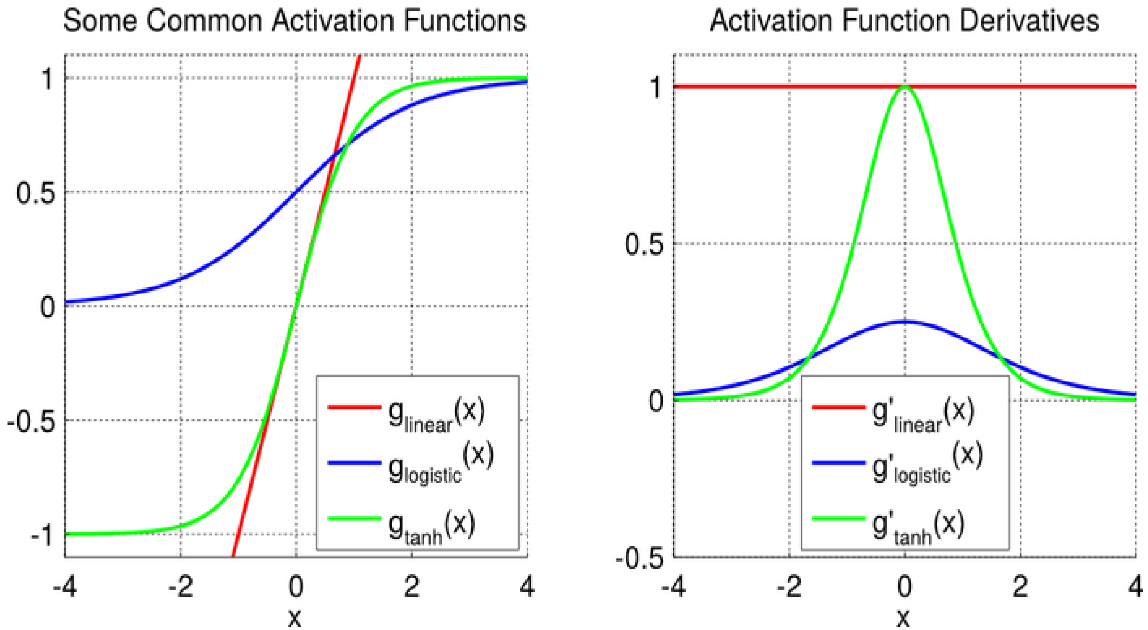
γ_g = ¼ for sigmoid

γ_g = 1 for tanh

constant

Gradient of the nonlinear function (sigmoid or tanh) $g'(h_{i-1})$ is bounded by constant, .i.e., $\left\|diag\left(g'(h_{i-1})\right)\right\| \leq \gamma_g$

an upper bound for the norm of the gradient of activation



Mechanics behind Vanishing and Exploding Gradients

Let's use these properties:

$$\frac{\partial h_t}{\partial \mathrm{h}_k} = \prod_{t \geq i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \geq i > k} \mathrm{W}_{hh}{}^{T} diag[g'(h_{i-1})]$$

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\| \leq \left\| \mathrm{W}_{hh}{}^{T} \right\| \left\| diag\big(g'(h_{i-1})\big) \right\|$$

Largest Singular value of W_{hh}

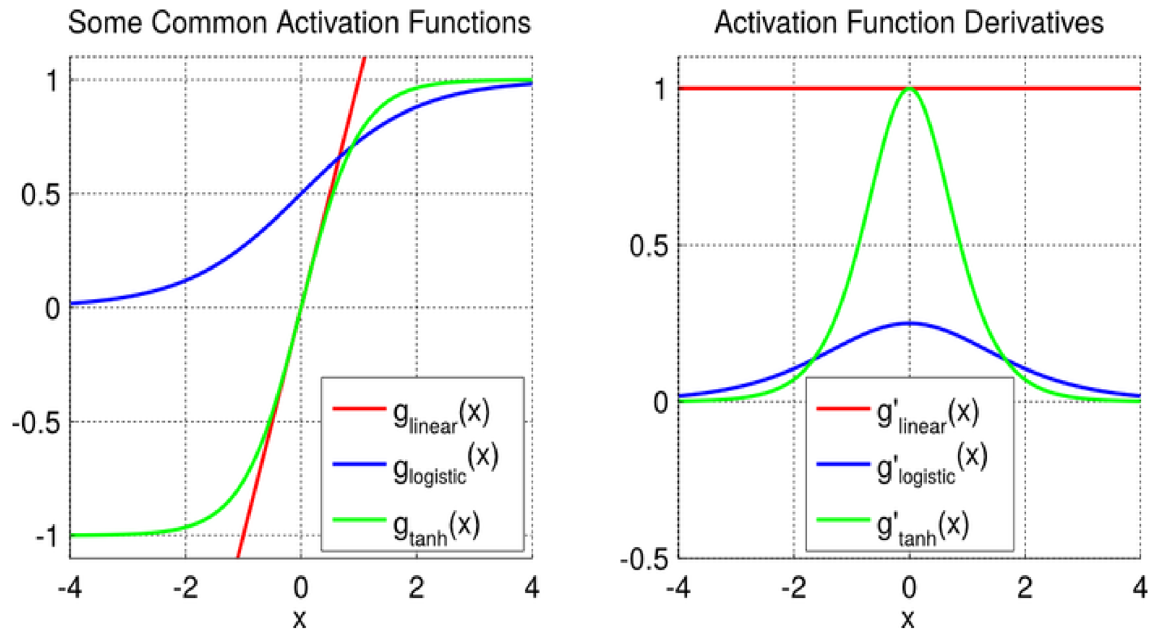
$$\leq \boxed{\gamma_W} \gamma_g$$

$\gamma_W \gamma_g$ = **an upper bound for the norm of jacobian!**

$$\boxed{\left\| \frac{\partial h_3}{\partial h_k} \right\| \leq \left(\gamma_W \gamma_g\right)^{t-k}}$$

Gradient of the nonlinear function (sigmoid or tanh) $g'(h_{i-1})$ is bounded by constant, .i.e., $\left\| diag\big(g'(h_{i-1})\big) \right\| \leq \gamma_g$

an upper bound for the norm of the gradient of activation



Mechanics behind Vanishing and Exploding Gradients

Let's use these properties:

$$\frac{\partial h_t}{\partial \mathrm{h}_k} = \prod_{t \geq i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \geq i > k} \mathrm{W}_{hh}{}^{T} diag[g'(h_{i-1})]$$

$$\left\|\frac{\partial h_i}{\partial h_{i-1}}\right\| \leq \|\mathrm{W}_{hh}{}^{T}\|\|diag(g'(h_{i-1}))\|$$

Largest Singular value of W_{hh}

$$\leq \boxed{\gamma_W}\gamma_g$$

$\gamma_W \gamma_g =$ an upper bound for the norm of jacobian!

$$\left\|\frac{\partial h_3}{\partial h_k}\right\| \leq (\gamma_W \gamma_g)^{t-k}$$

Sufficient Condition for Vanishing Gradient

As $\gamma_W \gamma_g < 1$ and (t-k)→∞ then long term contributions go to 0 exponentially fast with t-k (*power iteration method*).

Therefore,

sufficient condition for vanishing gradient to occur:

$$\boxed{\gamma_W < 1/\gamma_g}$$

i.e. for sigmoid, $\gamma_W < 4$

i.e., for tanh, $\gamma_W < 1$

Mechanics behind Vanishing and Exploding Gradients

Let's use these properties:

$$\frac{\partial h_t}{\partial \mathrm{h}_k} = \prod_{t \geq i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \geq i > k} \mathrm{W}_{hh}{}^T diag[g'(h_{i-1})]$$

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\| \leq \left\| \mathrm{W}_{hh}{}^T \right\| \left\| diag\left(g'(h_{i-1})\right) \right\|$$

Largest Singular value of W_{hh}

$$\leq \boxed{\gamma_W} \gamma_g$$

$\gamma_W \gamma_g =$ **an upper bound for the norm of jacobian!**

$$\boxed{\left\| \frac{\partial h_3}{\partial h_k} \right\| \leq \left(\gamma_W \gamma_g\right)^{t-k}}$$

Necessary Condition for Exploding Gradient

As $\gamma_W \gamma_g > 1$ and (t-k)→∞ then gradient explodes!!!

Therefore,

Necessary condition for exploding gradient to occur:

$$\boxed{\gamma_W > 1/\gamma_g}$$

i.e. for sigmoid, $\gamma_W > 4$

i.e., for tanh, $\gamma_W > 1$

Vanishing Gradient in Long-term Dependencies

? **What have we concluded with the upper bound of derivative from recurrent step?**

$$\frac{\partial h_t}{\partial \mathrm{h}_k} = \prod_{t \geq i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \geq i > k} \mathrm{W}_{hh}{}^{T} diag[g'(h_{i-1})]$$

$$\left\|\frac{\partial h_3}{\partial h_k}\right\| \leq \left(\gamma_W \gamma_g\right)^{t-k}$$

$$\left\|\frac{\partial h_i}{\partial h_{i-1}}\right\| \leq \left\|\mathrm{W}_{hh}{}^{T}\right\| \left\|diag([g'(h_{i-1})])\right\| \leq \gamma_W \gamma_g$$

If we multiply the same term $\gamma_W \gamma_g < 1$ again and again, the overall number becomes very small(i.e almost equal to zero)

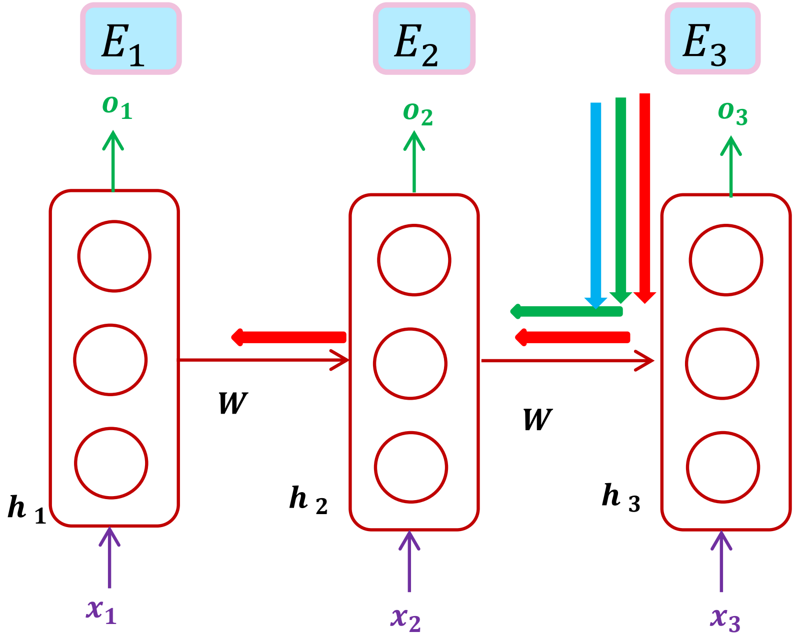
HOW ?

Repeated matrix multiplications leads to vanishing and exploding gradients

Vanishing Gradient in Long-term Dependencies

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W} + \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W} + \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_3}\frac{\partial h_3}{\partial W}$$

$$= \lll 1 \quad + \quad \ll 1 \quad + \quad < 1$$



The gradients no longer depend on the past inputs…

since, the near past inputs dominate the gradient !!!

Total Gradient = **Gradient due to long term dependencies** + **Gradient due to short term dependencies**

Remark: *The gradients due to short term dependencies (just previous dependencies) dominates the gradients due to long-term dependencies.*

This means network will tend to focus on short term dependencies which is often not desired

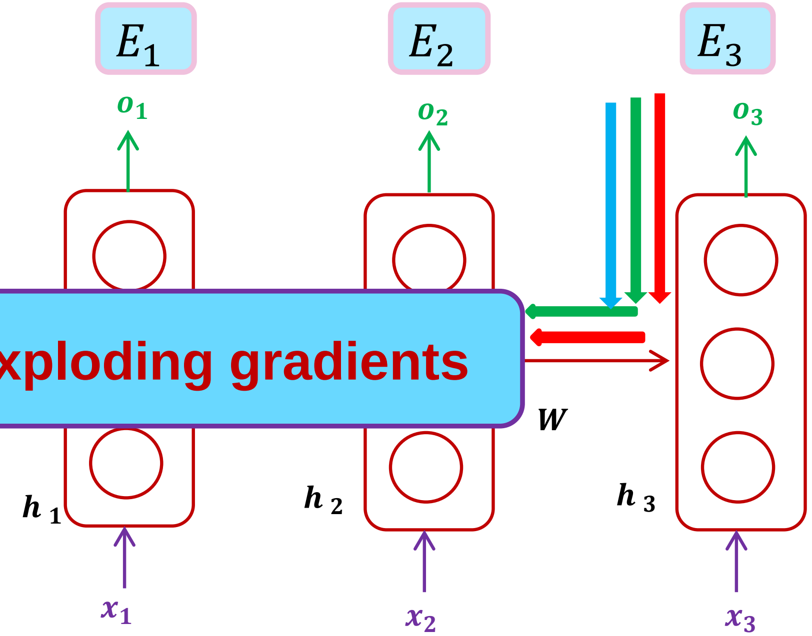
Problem of Vanishing Gradient

Vanishing Gradient in Long-term Dependencies

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W} + \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W} + \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_3}\frac{\partial h_3}{\partial W}$$

$$= \lll 1 \quad + \quad \ll 1 \quad + \quad < 1$$

Repeated matrix multiplications leads to vanishing and exploding gradients



Total Gradient = **Gradient due to long term dependencies** + **Gradient due to short term dependencies**

Remark: *The gradients due to short term dependencies (just previous dependencies) dominates the gradients due to long-term dependencies.*

This means network will tend to focus on short term dependencies which is often not desired

Problem of Vanishing Gradient

Exploding Gradient in Long-term Dependencies

? **What have we concluded with the upper bound of derivative from recurrent step?**

$$\frac{\partial h_t}{\partial \mathrm{h}_k} = \prod_{t \geq i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \geq i > k} \mathrm{W}_{hh}{}^T diag[g'(h_{i-1})]$$

$$\boxed{\left\| \frac{\partial h_3}{\partial h_k} \right\| \leq \left(\gamma_W \gamma_g\right)^{t-k}}$$

$$\left\| \frac{\partial h_i}{\partial h_{i-1}} \right\| \leq \left\| \mathrm{W}_{hh}{}^T \right\| \| diag([g'(h_{i-1})]) \| \leq \gamma_W \, \gamma_g$$

If we multiply the same term $\gamma_W \, \gamma_g > 1$ again and again, the overall number explodes and hence the gradient explodes

HOW ?

Repeated matrix multiplications leads to vanishing and exploding gradients

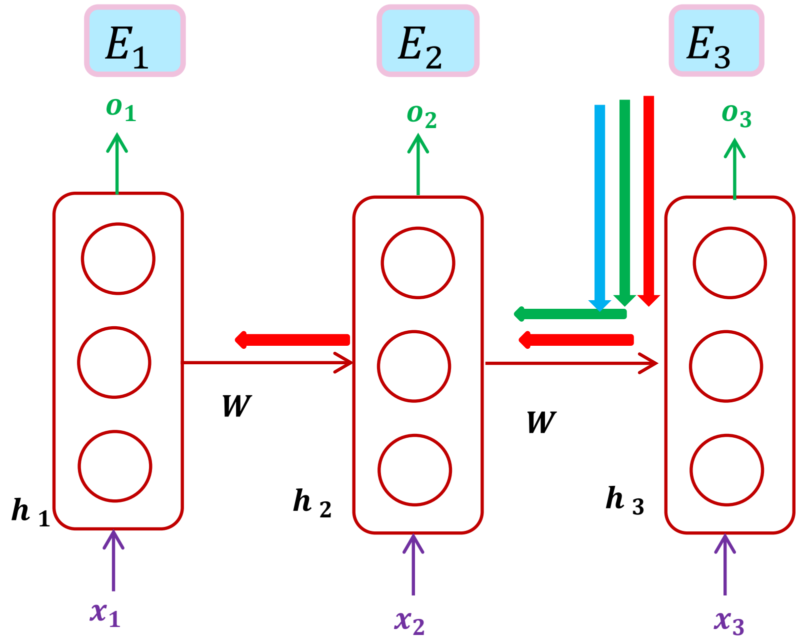
Vanishing Gradient in Long-term Dependencies

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W} + \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W} + \frac{\partial E_3}{\partial h_3}\frac{\partial h_3}{\partial h_3}\frac{\partial h_3}{\partial W}$$

$$= \gg \gg 1 \quad + \quad \gg\gg 1 \quad + \quad \gg\gg 1$$

= Very large number, i.e., NaN

Problem of Exploding Gradient



Vanishing vs Exploding Gradients

$$\left\|\frac{\partial h_3}{\partial h_k}\right\| \leq \left(\gamma_W \gamma_g\right)^{t-k}$$

For tanh or linear activation

$\gamma_W \gamma_g$ **greater than** 1

Gradient *Expodes* !!!

$\gamma_W \gamma_g$ **less than** 1

Gradient *Vanishes* *!!!*

Remark: This problem of exploding/vanishing gradient occurs because the same number is multiplied in the gradient repeatedly.

Dealing With Exploding Gradients

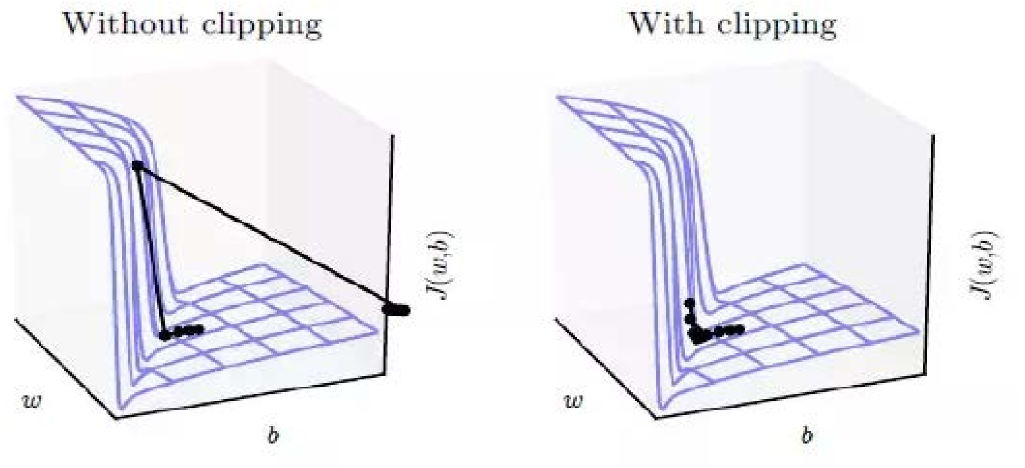
Dealing with Exploding Gradients: Gradient Clipping

Scaling down the gradients

- rescale norm of the gradients whenever it goes over a threshold

Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if



- Proposed clipping is simple and computationally efficient,
- introduce an additional hyper-parameter, namely the threshold

Pascanu et al., 2013. On the difficulty of training recurrent neural networks.

Dealing With Vanishing Gradients

Dealing with Vanishing Gradient

- As discussed, the gradient vanishes due to the recurrent part of the RNN equations.

$$h_t = W_{hh}\, \mathrm{h}_{\mathrm{t}-1} + \text{some other terms}$$

- **What if Largest Eigen value of the parameter matrix becomes 1, but in this case, memory just grows.**
- **We need to be able to decide when to put information in the memory**

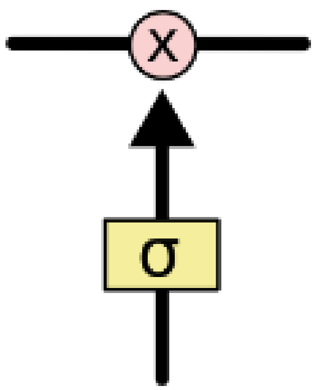
Gates :

→ way to optionally let information through.

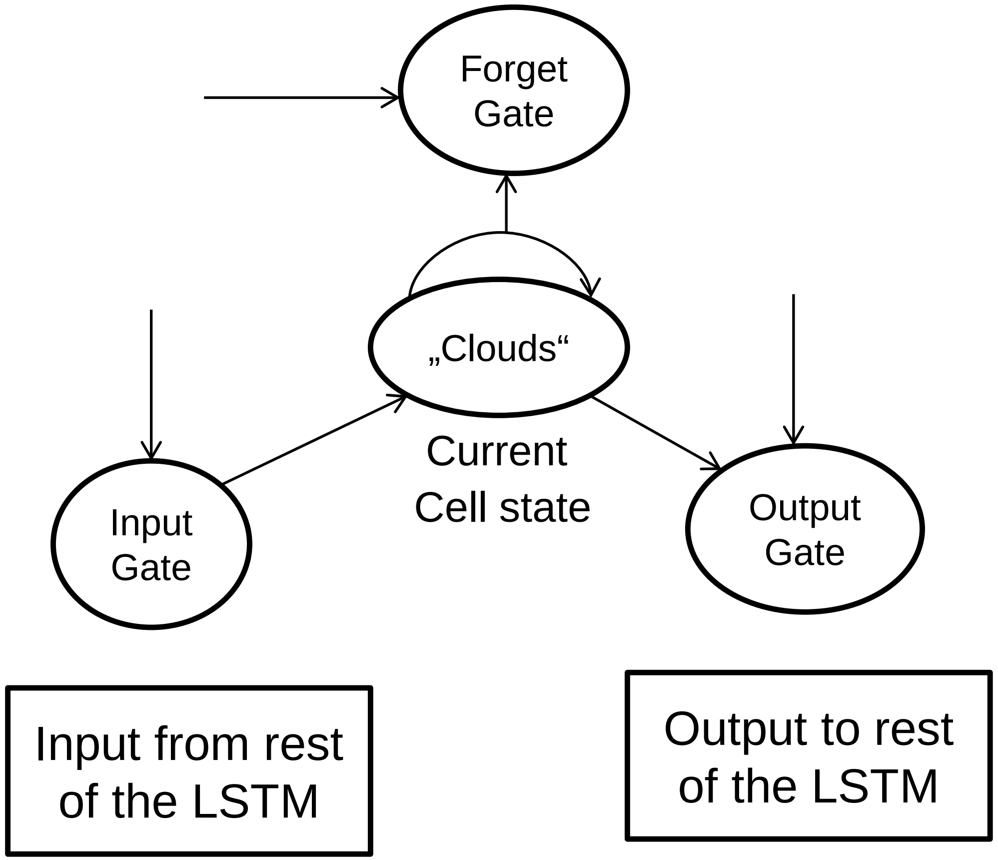
→ composed out of a sigmoid neural net layer and a pointwise multiplication operation.

→ remove or add information to the cell state

→ 3 gates in LSTM



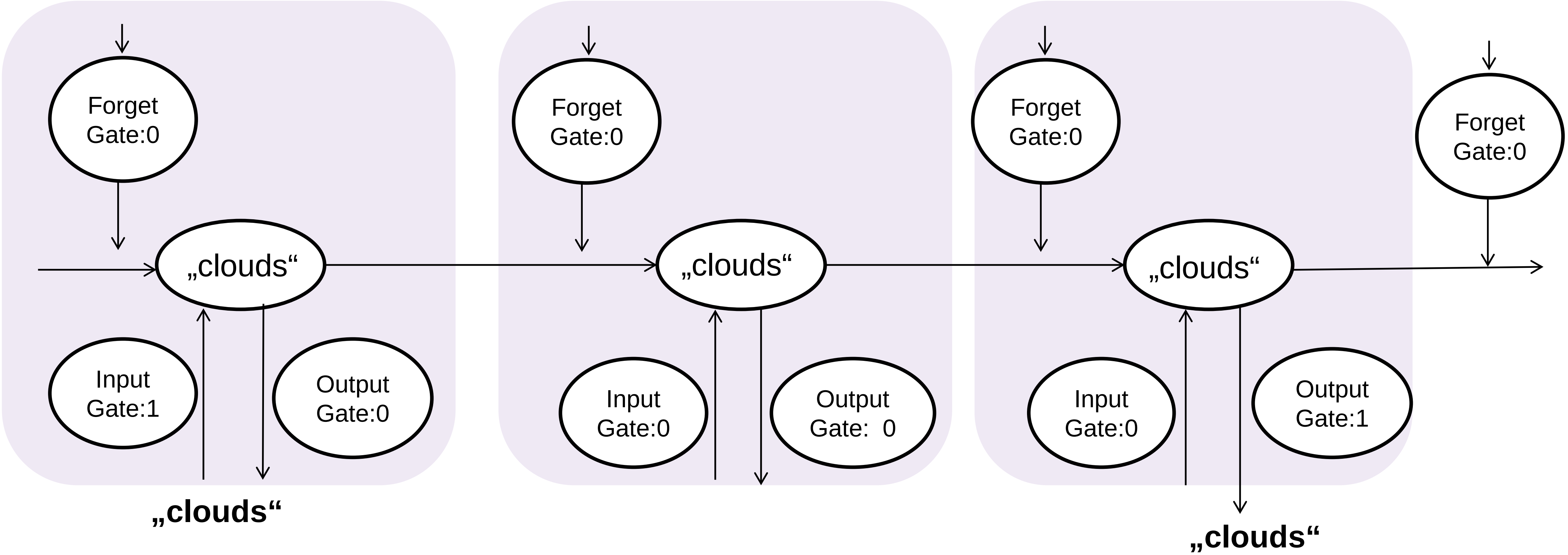
→ gates to protect and control the cell state.



http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short Term Memory (LSTM): Gating Mechanism

Remember the word „ **clouds**“ over time….

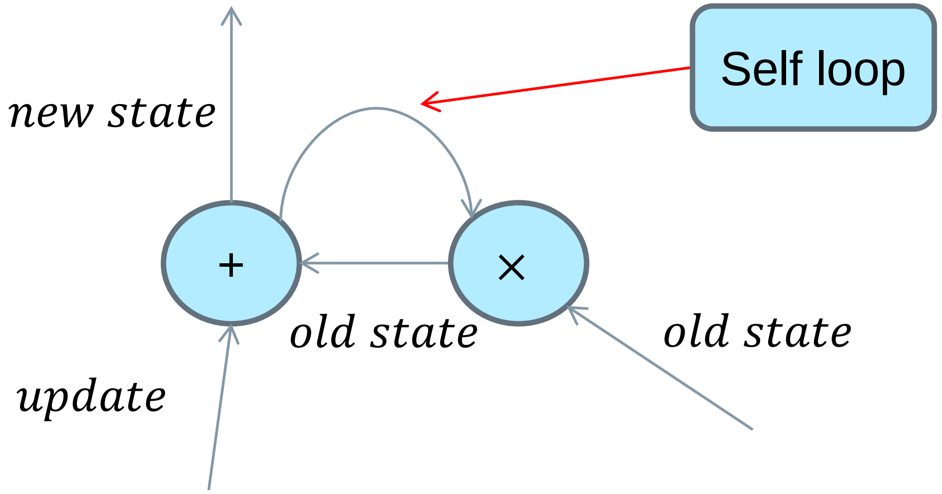


Lecture from the course Neural Networks for Machine Learning by Greff Hinton

Long Short Term Memory (LSTM)

Motivation:

- **Create a self loop path from where gradient can flow**
- **self loop corresponds to an eigenvalue of Jacobian to be slightly less than 1**



$$new\ state = old\ state + update$$

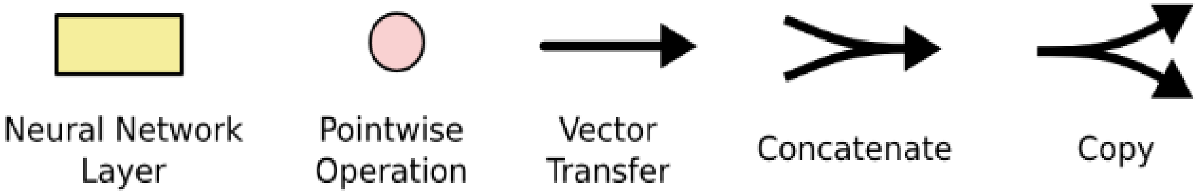
$$\frac{\partial new\ state}{\partial old\ state} \cong Identity$$

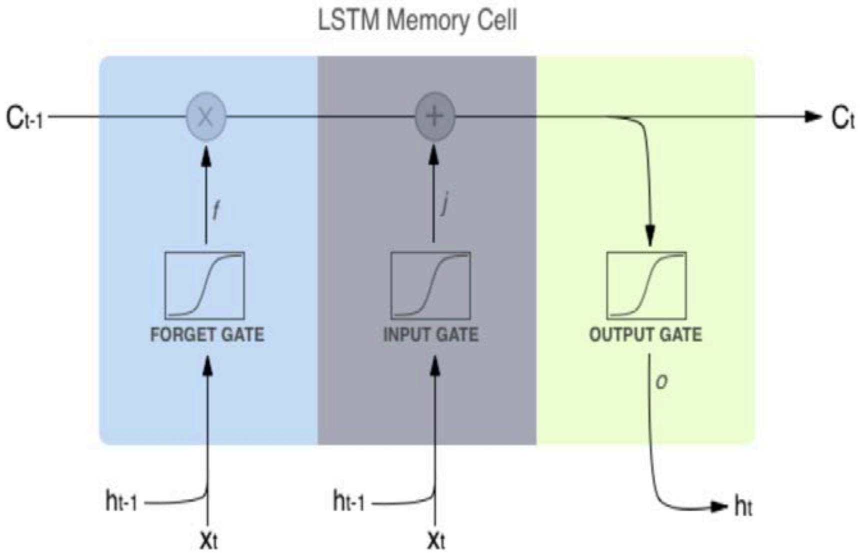
LONG SHORT-TERM MEMORY, Sepp Hochreiter and Jürgen Schmidhuber

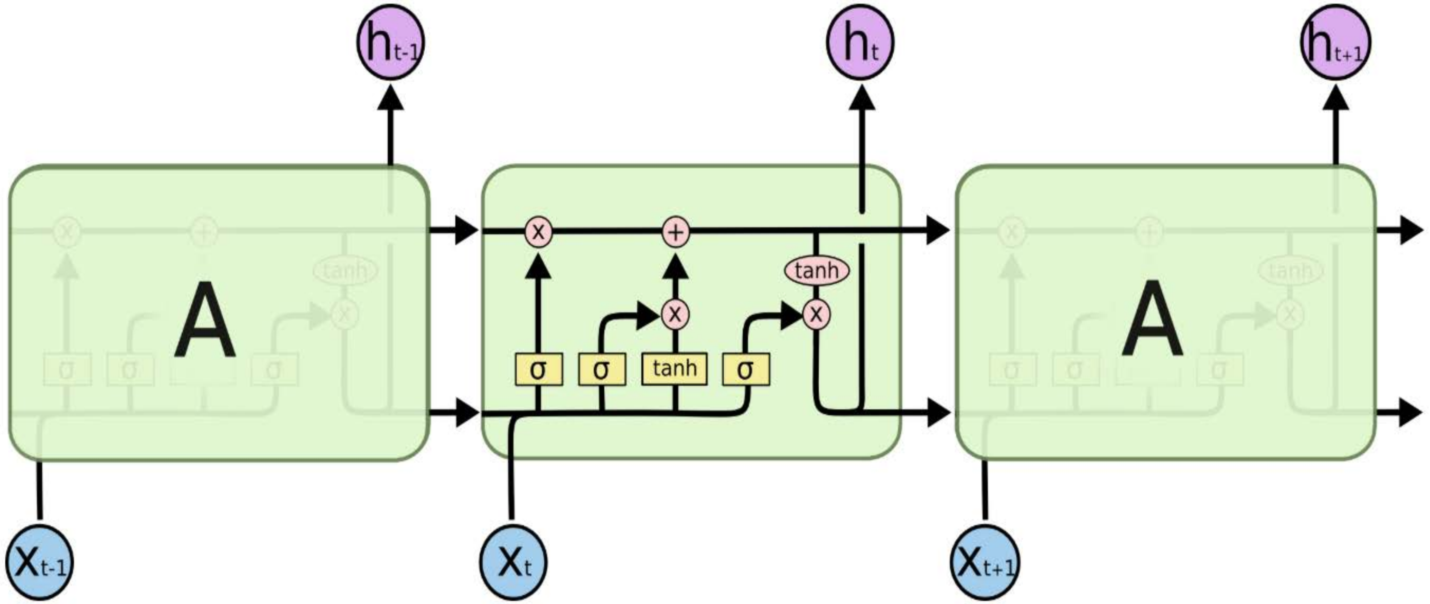
Key Ingredients

Cell state - transport the information through the units

Gates – optionally allow information passage







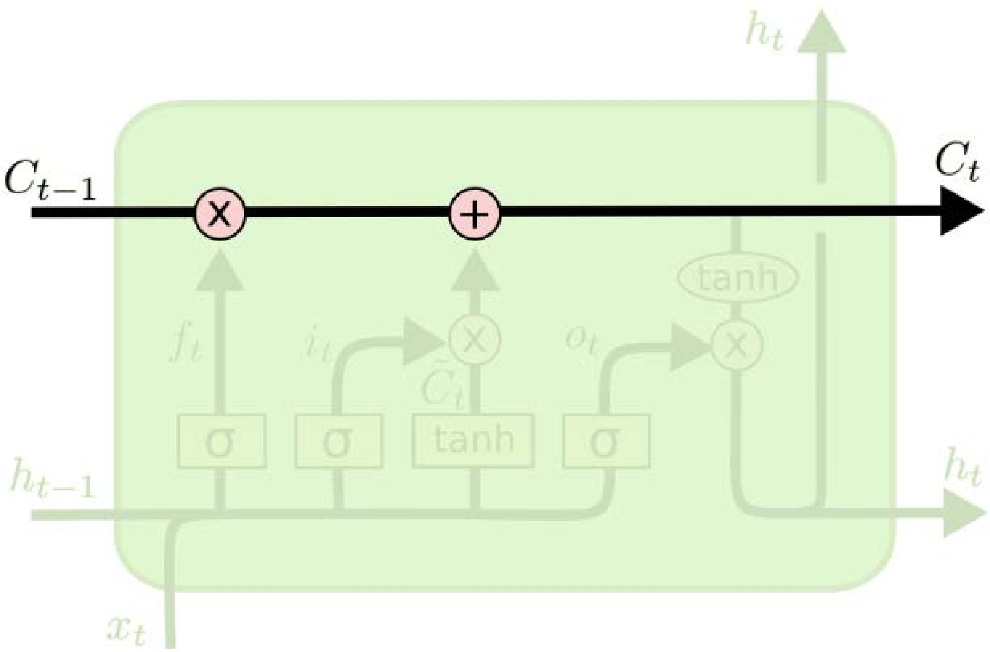
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

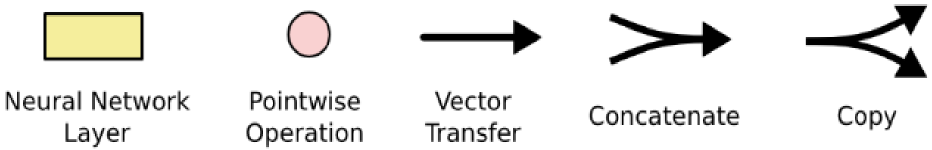
Long Short Term Memory (LSTM): Step by Step

Cell: Transports information through the units (key idea)

→ the horizontal line running through the top

LSTM removes or adds information to the cell state using gates.





Long Short Term Memory (LSTM): Step by Step

Forget Gate:

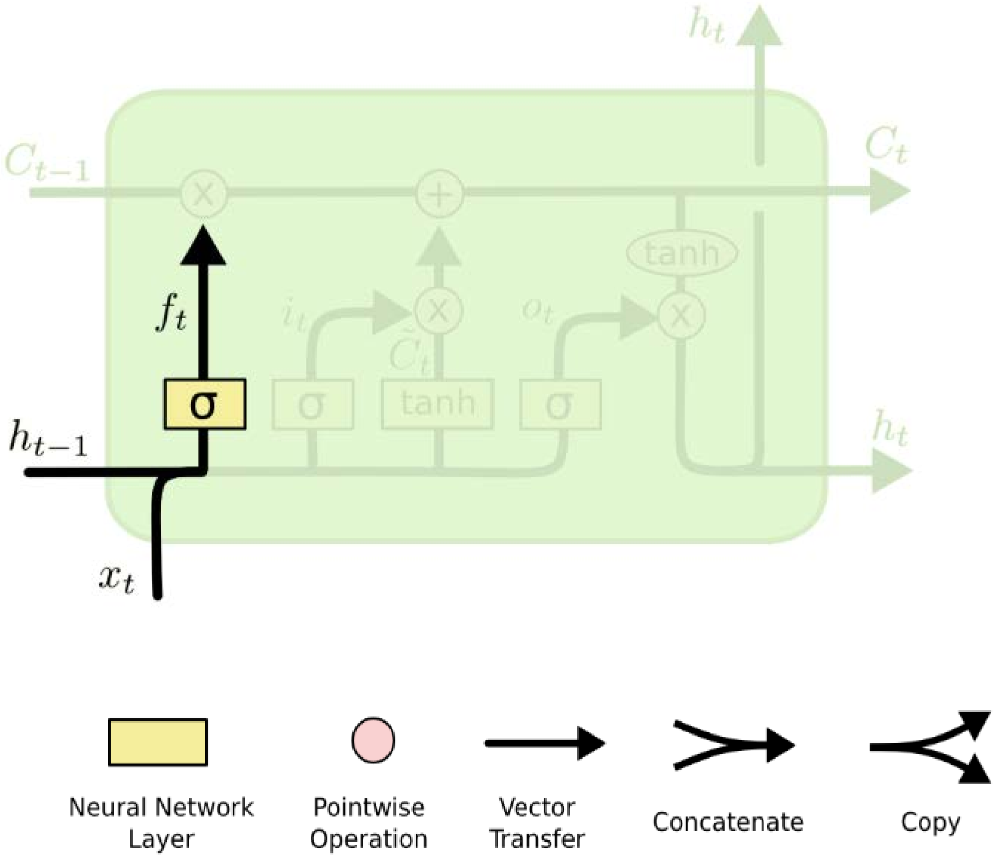
→ decides what information to throw away or remember from the previous cell state

→ decision maker: sigmoid layer (*forget gate layer*)

The output of the sigmoid lies between 0 to 1,

→ 0 being forget, 1 being keep.

$$\boxed{\boldsymbol{f_t = sigmoid(\theta_{xf} x_t + \theta_{hf} h_{t-1} + b_f)}}$$

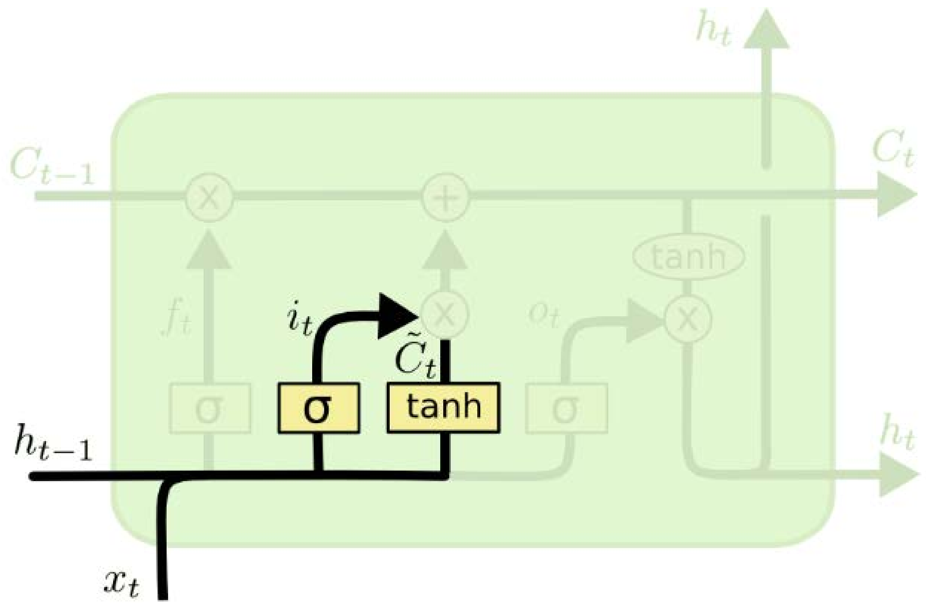


→ looks at h_{t-1} and x_t, and outputs a number between 0 and 1

for each number in the cell state C_{t-1}

Input Gate: Selectively updates the cell state based on the new input.

A multiplicative input gate unit to protect the memory contents stored in j from perturbation by irrelevant inputs



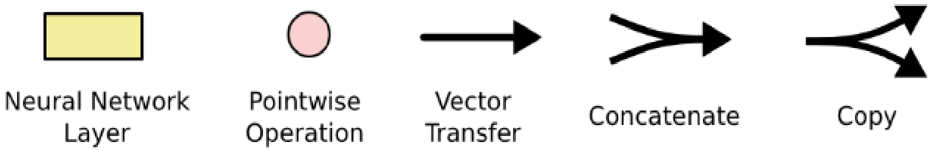
$$i_t = sigmoid(\theta_{xi}x_t + \theta_{hi}h_{t-1} + b_i)$$
$$\tilde{C}_t = Tanh(\theta_{xg}x_t + \theta_{hg}h_{t-1} + b_g)$$

The next step is to decide what new information we're going to store in the cell state. This has two parts:

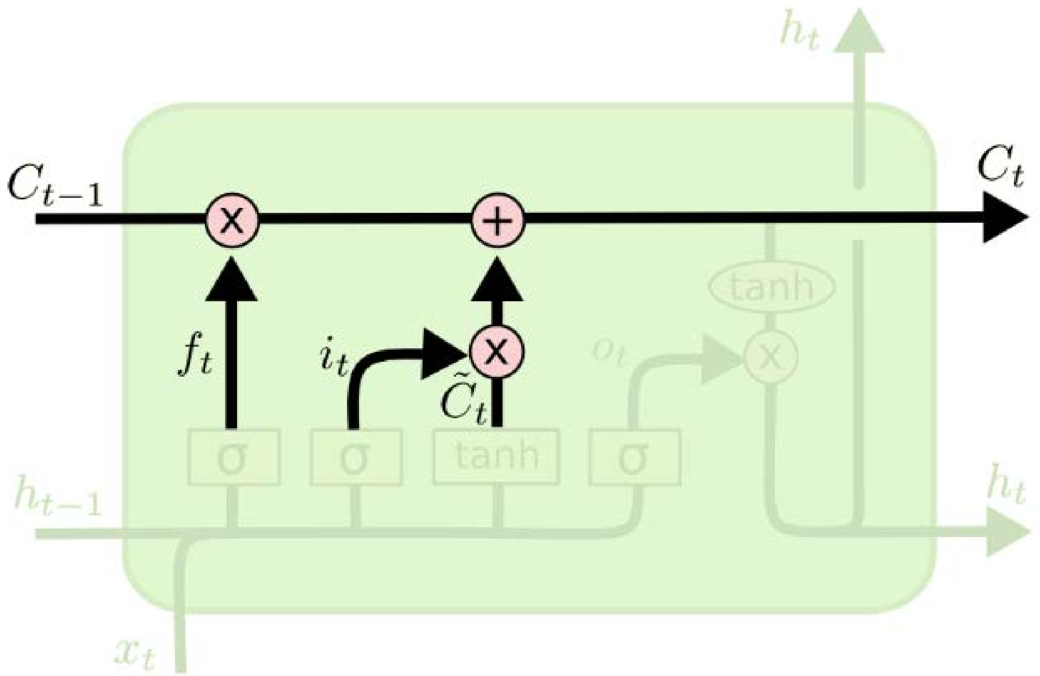
1. A sigmoid layer called the "input gate layer" decides **which values we'll update**.

2. A tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that **could be added to the state**.

In the next step, we'll combine these two to **create an update** to the state.



Cell Update

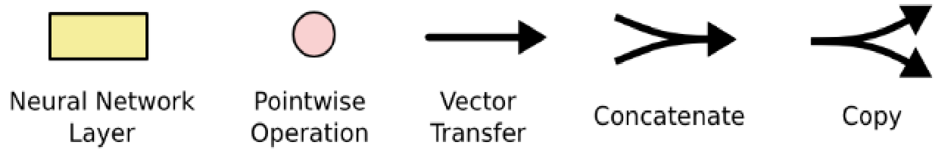


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- update the old cell state, $\mathbf{C_{t-1}}$, into the new cell state $\mathbf{C_t}$

- multiply the old state by $\mathbf{f_t}$, forgetting the things we decided to forget earlier

- add $\mathbf{i_t} * \tilde{C}_t$ to get the new candidate values, scaled by how much we decided to update each state value.



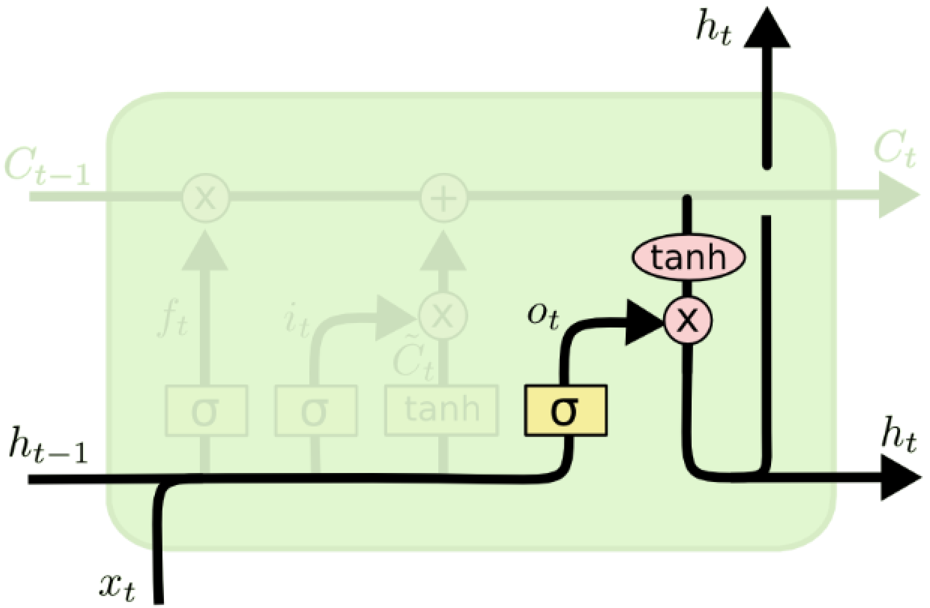
Long Short Term Memory (LSTM): Step by Step

Output Gate: Output is the filtered version of the cell state

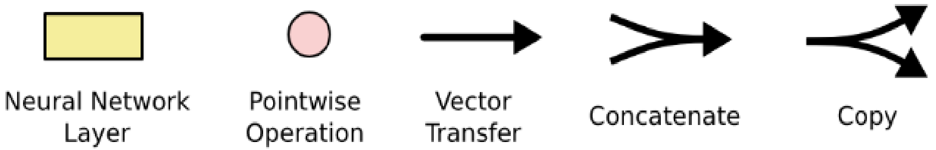
- Decides the part of the cell we want as our output in the form of new hidden state

- multiplicative output gate to protect other units from perturbation by currently irrelevant memory contents

- a sigmoid layer decides what parts of the cell state goes to output. Apply tanh to the cell state and multiply it by the output of the sigmoid gate → only output the parts decided



$$o_t = sigmoid(\theta_{xo} x_t + \theta_{ho} h_{t-1} + b_o)$$
$$h_t = o_t * tanh(C_t)$$



Dealing with Vanishing Gradients in LSTM

As seen, the gradient vanishes due to the recurrent part of the RNN equations

$$h_t = W_{hh} \mathrm{h}_{\mathrm{t}-1} + \text{some other terms}$$

How LSTM tackled vanishing gradient?

***Answer*: forget gate**

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

- The forget gate parameters takes care of the vanishing gradient problem
- Activation function becomes identity and therefore, the problem of vanishing gradient is addressed.
- The derivative of the identity function is, conveniently, always one. **So if f = 1**, information from the previous cell state can pass through this step unchanged

LSTM code snippet

Code snippet for LSTM unit:

```
n_in = 2 # for embedded reber grammar
n_hidden = n_i = n_c = n_o = n_f = 10
n_y = 2 # for embedded reber grammar

W_xi = sample_weights(n_in, n_i)
W_hi = sample_weights(n_hidden, n_i)
W_ci = sample_weights(n_c, n_i)
b_i =  np.random.uniform(-0.5,.5,size = n_i)
W_xf = sample_weights(n_in, n_f)
W_hf = sample_weights(n_hidden, n_f)
W_cf = sample_weights(n_c, n_f)
b_f =  np.random.uniform(0, 1.,size = n_f)
W_xc = sample_weights(n_in, n_c)
W_hc = sample_weights(n_hidden, n_c)
b_c =  np.zeros(n_c)
W_xo = sample_weights(n_in, n_o)
W_ho = sample_weights(n_hidden, n_o)
W_co = sample_weights(n_c, n_o)
b_o =  np.random.uniform(-0.5,.5,size = n_o)
W_hy = sample_weights(n_hidden, n_y)
b_y =  np.zeros(n_y)

c0 = np.zeros(n_hidden)
h0 = np.tanh(c0)
```

Offline

Parameter Dimension

LSTM code snippet

Offline

Code snippet for LSTM unit: LSTM equations forward pass and shape of gates

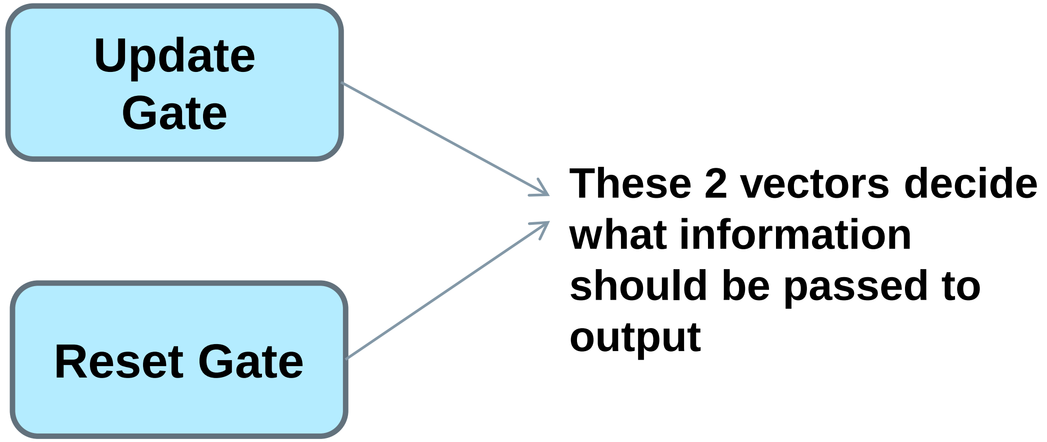
```
i_t = sigma(np.dot(X, W_xi) + np.dot(h0, W_hi) + np.dot(c0, W_ci) + b_i)
f_t = sigma(np.dot(X, W_xf) + np.dot(h0, W_hf) + np.dot(c0, W_cf) + b_f)
c_t = f_t * c0 + i_t * np.tanh(np.dot(X, W_xc) + np.dot(h0, W_hc) + b_c)
o_t = sigma(np.dot(X, W_xo)+ np.dot(h0, W_ho) + np.dot(c_t, W_co)  + b_o)
h_t = o_t * np.tanh(c_t)
y_t = sigma(np.dot(h_t, W_hy) + b_y)


print(np.shape(i_t))
print(np.shape(f_t))
print(np.shape(c_t))
print(np.shape(o_t))
print(np.shape(h_t))

print(np.shape(y_t))
```

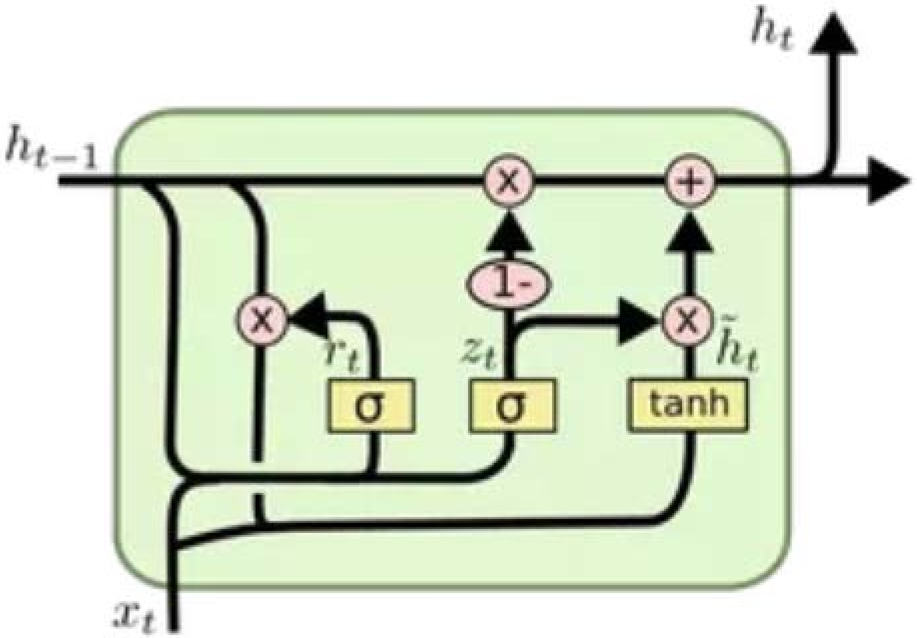
```
(300L, 10L)
(300L, 10L)
(300L, 10L)
(300L, 10L)
(300L, 10L)
(300L, 2L)
```

Gated Recurrent Unit (GRU)

- GRU like LSTMs, attempts to solve the Vanishing gradient problem in RNN

Gates:





- Units with short-term dependencies will have active reset gates r
- Units with long term dependencies have active update gates z

Gated Recurrent Unit (GRU)

Update Gate:

- to determine how much of the past information (from previous time steps) needs to be passed along to the future.
- to learn to copy information from the past such that gradient is not vanished.

Here, x_t is the input and h_{t-1} holds the information from the previous gate.

$$z_t = sigmoid(W^z x_t + U^z h_{t-1})$$

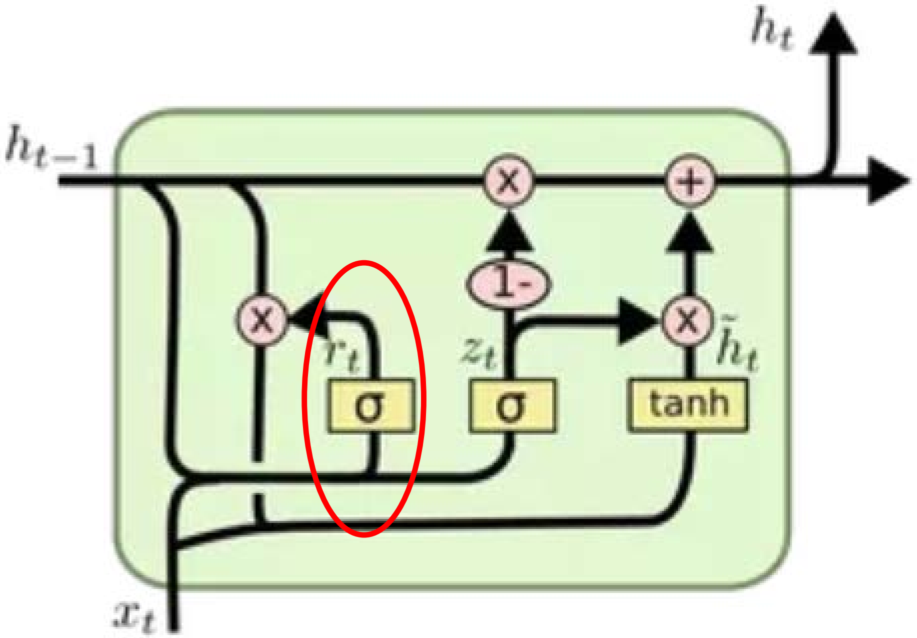
Gated Recurrent Unit (GRU)

Reset Gate

- model how much of information to forget by the unit

Here, x_t is the input and h_{t-1} holds the information from the previous gate.

$$r_t = sigmoid(W^{(r)}x_t + U^{(r)}h_{t-1})$$



Memory Content:

$$h'_t = tanh(Wx_t + r\_t \odot Uh_{t-1})$$

Final Memory at current time step

$$h_t = z_t \odot h_{(t-1)} + (1 - z_t) \odot h'_t$$

Dealing with Vanishing Gradient s in Gated Recurrent Unit (GRU)

Offline

We had a product of Jacobian:

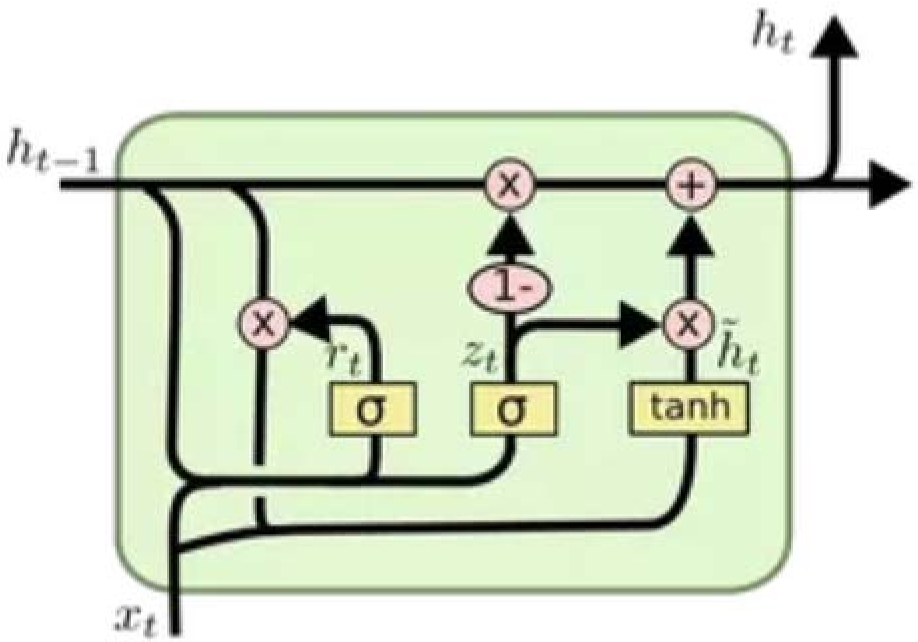
$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \leq \alpha^{t-j-1}$$

Where, alpha depends upon weight matrix and derivative of the activation function

Now,

$$\frac{\partial h_j}{\partial h_{j-1}} = z_j + (1 - z_j)\frac{\partial h'_j}{\partial h_{j-1}}$$

And, $$\frac{\partial h'_j}{\partial h_{j-1}} = 1 \; for \; z_j = 1$$



Code snippet of GRU unit

Code snippet of GRU unit:

Offline

```
def GRU(x_t, s_t1_prev):

        # Get the input as a word vector
        x_e = E[:,x_t]

        # GRU Layer
        z_t1 = T.nnet.hard_sigmoid(U[0].dot(x_e) + W[0].dot(s_t1_prev) + b[0])
        r_t1 = T.nnet.hard_sigmoid(U[1].dot(x_e) + W[1].dot(s_t1_prev) + b[1])
        c_t1 = T.tanh(U[2].dot(x_e) + W[2].dot(s_t1_prev * r_t1) + b[2])
        s_t1 = (T.ones_like(z_t1) - z_t1) * c_t1 + z_t1 * s_t1_prev

        # Final output calculation
        # Theano's softmax returns a matrix with one row, we only need the row
        o_t = T.nnet.softmax(V.dot(s_t1) + c)[0]

        return [o_t, s_t1]

```

Comparing LSTM and GRU

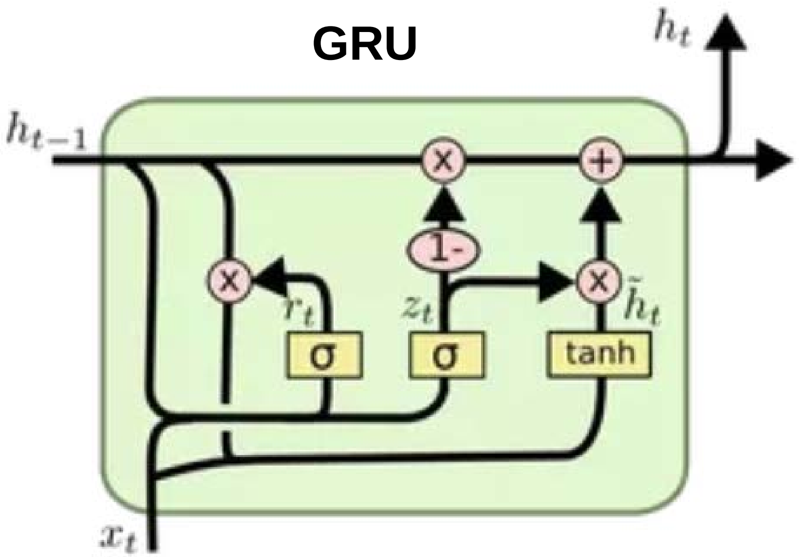
LSTM over GRU

One feature of the LSTM has: **controlled exposure of the memory content**, not in GRU.

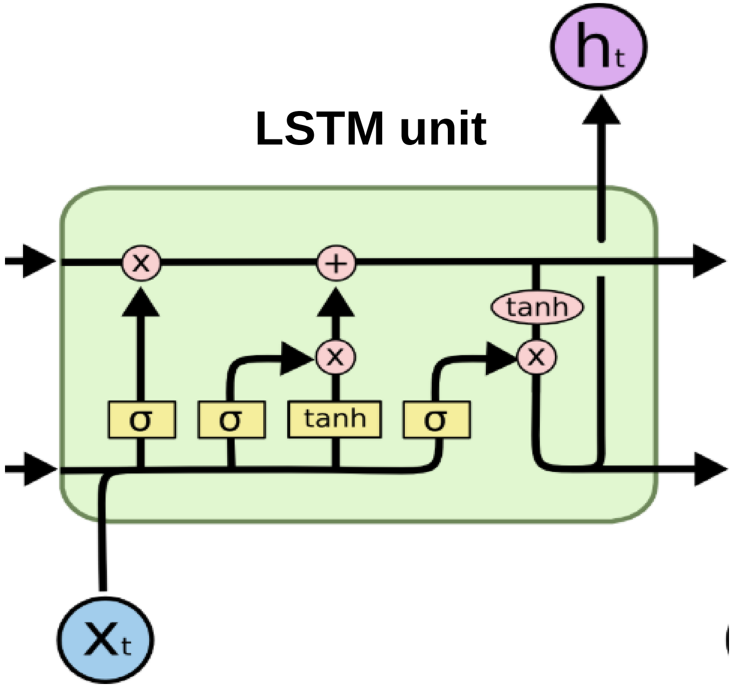
In the LSTM unit, the amount of the memory content that is seen, or used by other units in the network is controlled by the output gate. On the other hand **the GRU exposes its full content without any control**.

- GRU performs comparably to LSTM

GRU



LSTM unit



Chung et al, 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling

Break (10 minutes)

Bi-directional RNNs

Bidirectional Recurrent Neural Networks (BRNN)

- connects two hidden layers of opposite directions to the same output
- output layer can get information from past (backwards) and future (forward) states simultaneously
- learn representations from future time steps to better understand the context and eliminate ambiguity

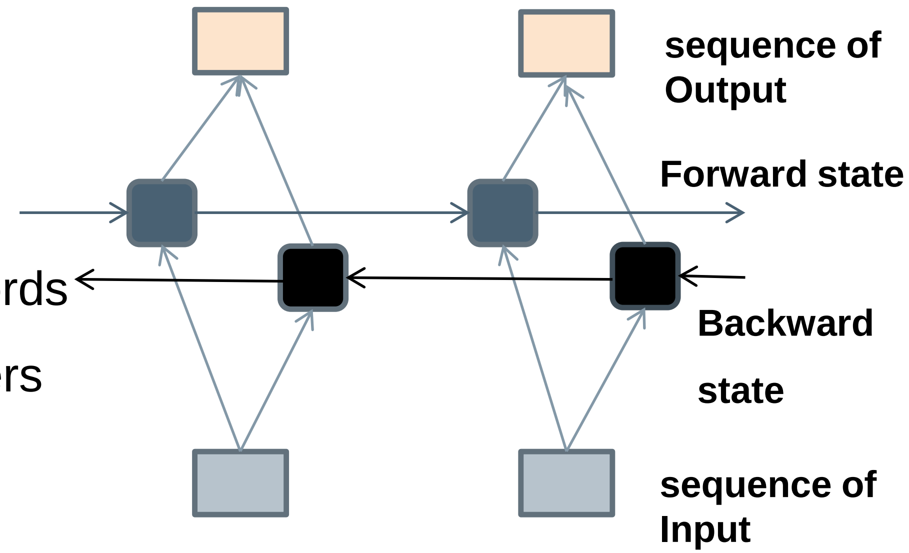
Example sentences:

Sentence1: “He said, ***Teddy*** bears are on sale”

Sentnce2: “He said, ***Teddy*** Roosevelt was a great President”.

when we are looking at the word “Teddy” and the previous two words “He said”, we might not be able to understand if the sentence refers to the President or Teddy bears.

Therefore, to resolve this ambiguity, we need to look ahead.



https://towardsdatascience.com/introduction-to-sequence-models-rnn-bidirectional-rnn-lstm-gru-73927ec9df15

Bi-directional RNNs

Bidirectional Recurrent Neural Networks (BRNN)

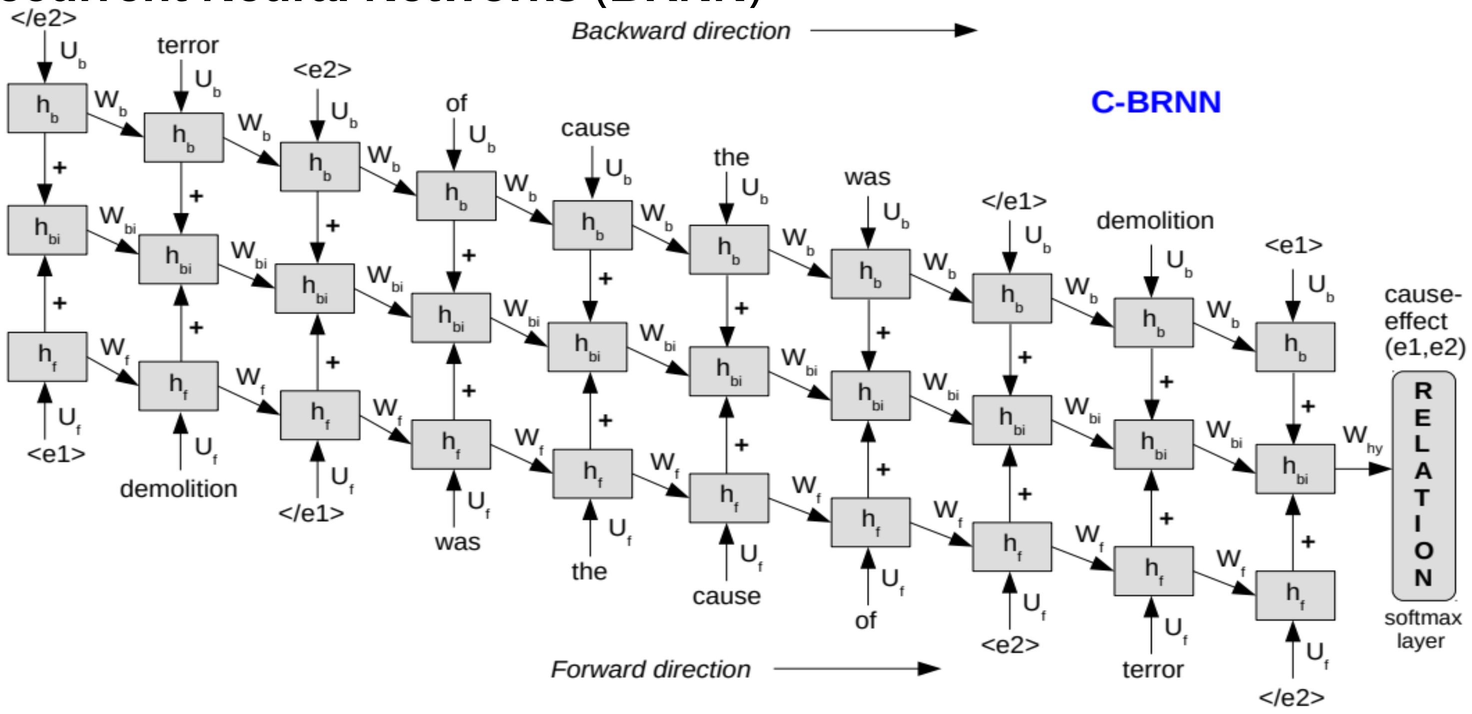


Figure 1: Connectionist Bi-directional Recurrent Neural Network (C-BRNN) (Vu et al., 2016a)

Gupta 2015. (Master Thesis). *Deep Learning Methods for the Extraction of Relations in Natural Language Text*

Gupta and Schütze. 2018. *LISA: Explaining Recurrent Neural Network Judgments via Layer-wIse Semantic Accumulation and Example to Pattern Transformation*

Vu et al., 2016. *Combining recurrent and convolutional neural networks for relation classification*

Recursive Neural Networks (RecNNs): TreeRNN or TreeLSTM

- applying the same set of weights recursively over a structured input, by traversing a given structure in topological order,

e.g., parse tree

- Use principle of compositionality

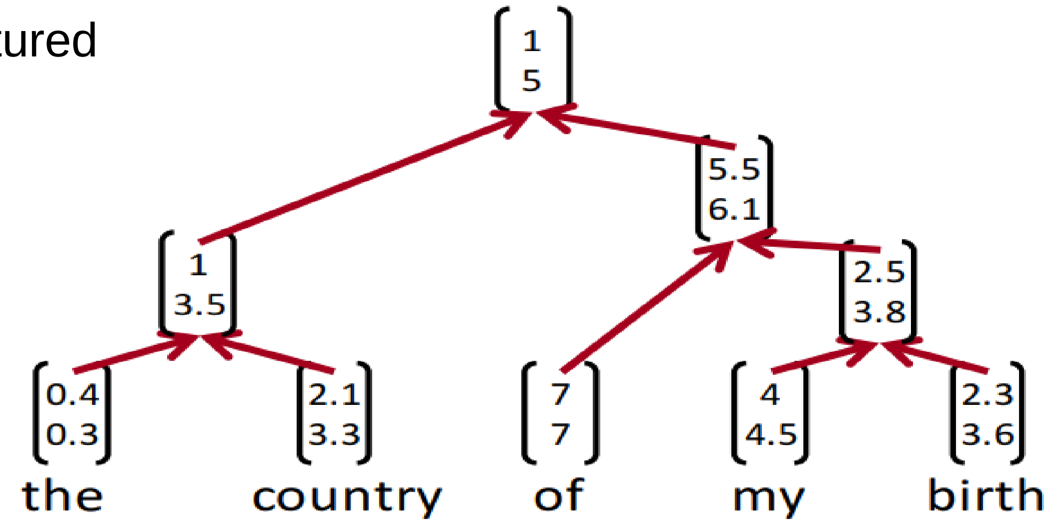
- Recursive Neural Nets can jointly learn compositional

vector representations and parse trees

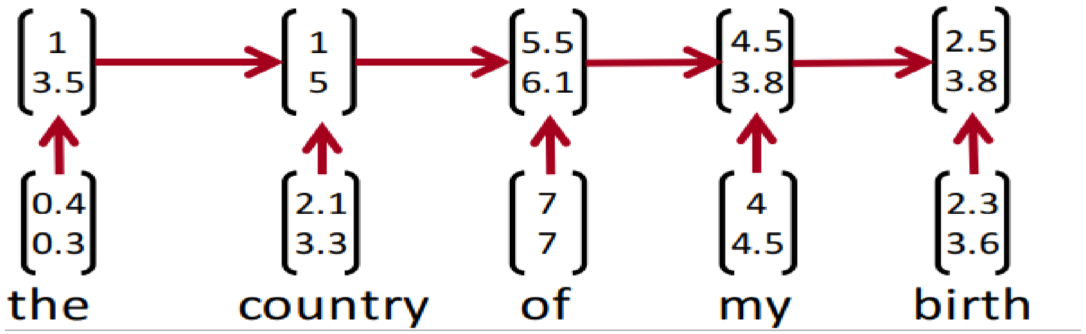
- The meaning (vector) of a sentence is determined by

(1) the meanings of its words and

(2) the rules that combine them.



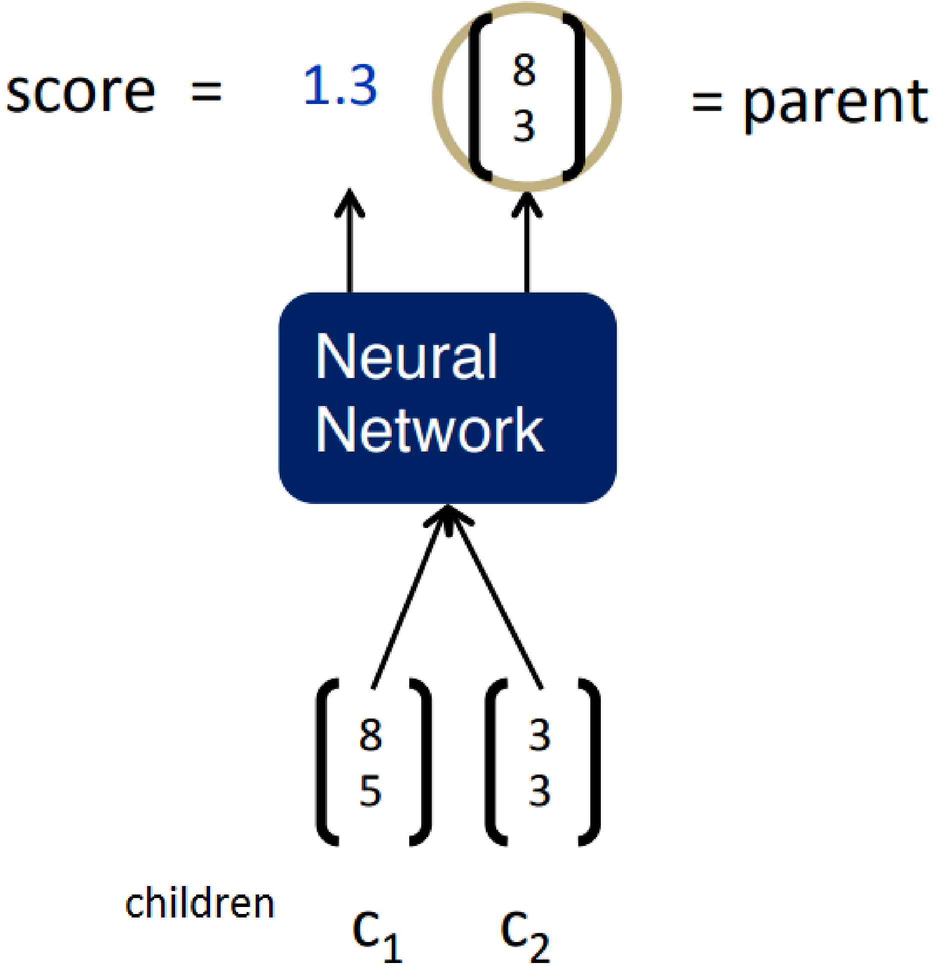
RecNN



RNN

http://www.iro.umontreal.ca/~bengioy/talks/gss2012-YB6-NLP-recursive.pdf

Recursive Neural Networks (RecNNs): TreeRNN or TreeLSTM



score = $U^T p$

$$\boxed{p = \tanh\left(W\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b\right)}$$

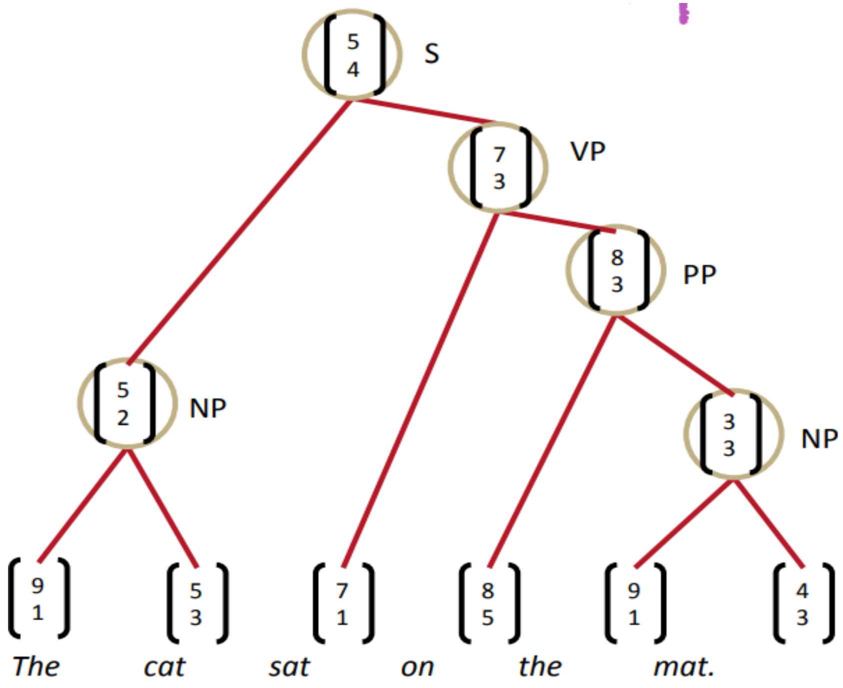
Same W parameters at all nodes of the tree

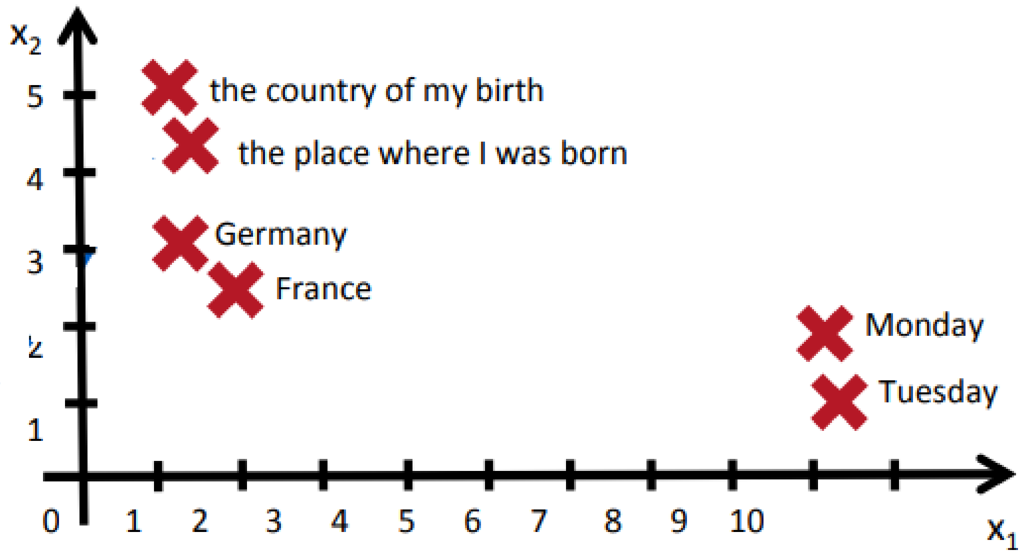
http://www.iro.umontreal.ca/~bengioy/talks/gss2012-YB6-NLP-recursive.pdf

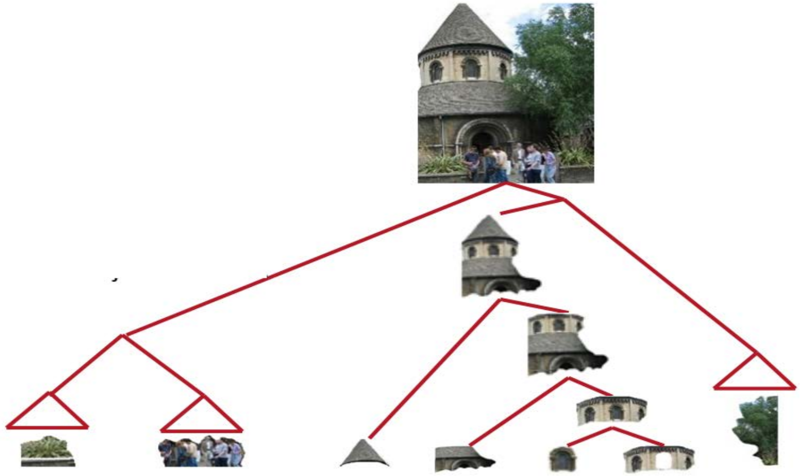
Recursive Neural Networks (RecNNs): TreeRNN or TreeLSTM

Applications

- represent the meaning of longer phrases
- Map phrases into a vector space
- Sentence parsing
- Scene parsing







Recursive Neural Networks (RecNNs): TreeRNN or TreeLSTM

Application: *Relation Extraction Within and Cross Sentence Boundaries, i.e., document-level relation extraction*

Paul Allen has started a company and named [*Vern Raburn*]$_{e1}$ its President. The company, to be called [*Paul Allen Group*]$_{e2}$ will be based in Bellevue, Washington.

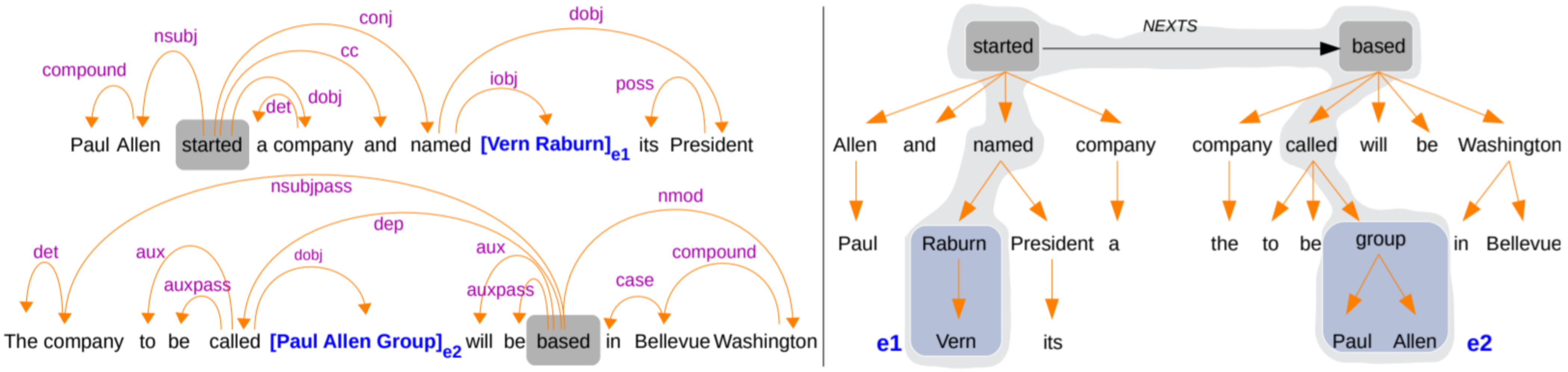
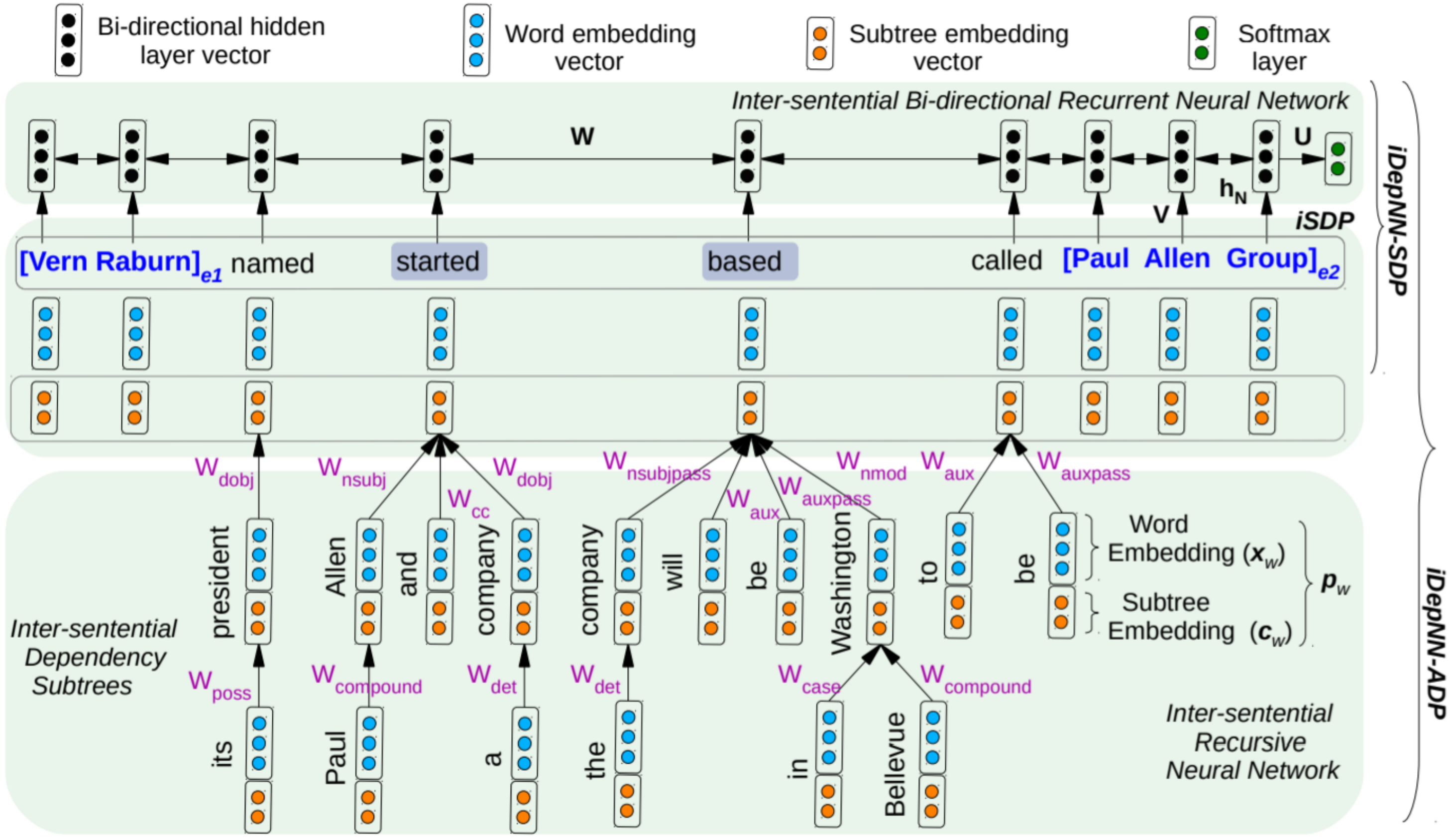


Figure 1: Left: Sentences and their dependency graphs. Right: Inter-sentential Shortest Dependency Path (iSDP) across sentence boundary. Connection between the roots of adjacent sentences by *NEXTS*.

Gupta et al., 2019. *Neural Relation Extraction Within and Across Sentence Boundaries.*

Recursive Neural Networks (RecNNs): TreeRNN or TreeLSTM

Relation Extraction Within and Cross Sentence Boundaries, i.e., document-level relation extraction



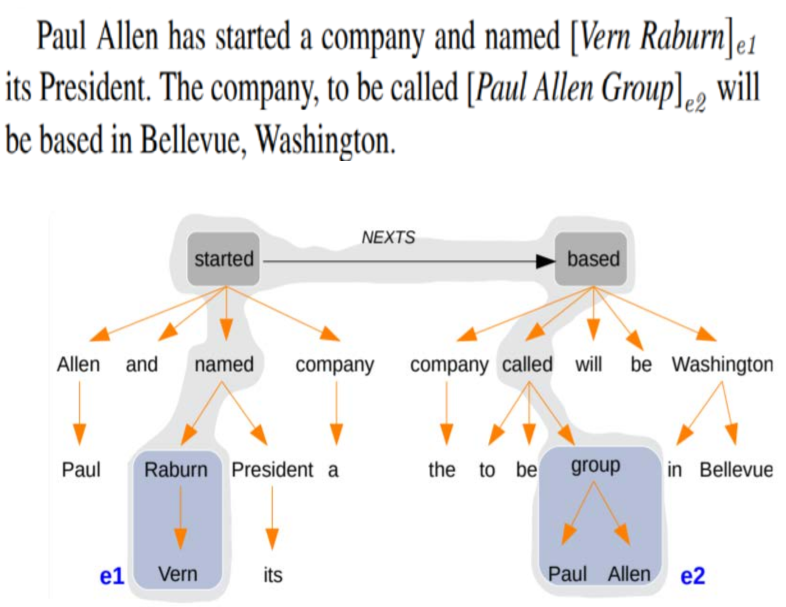
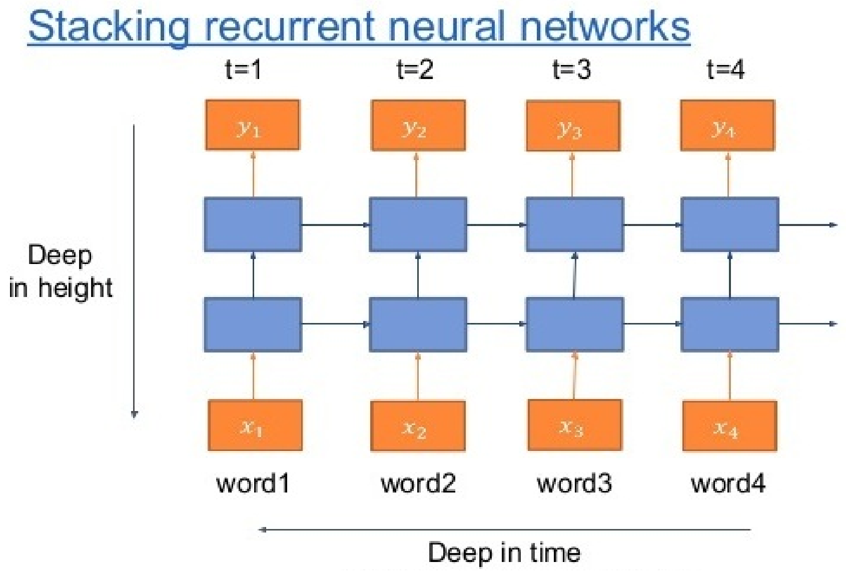


Figure 2: Inter-sentential Dependency-based Neural Network variants: iDepNN-SDP and iDepNN-ADP

Gupta et al., 2019. *Neural Relation Extraction Within and Across Sentence Boundaries.*

Deep and Multi-tasking RNNs



…

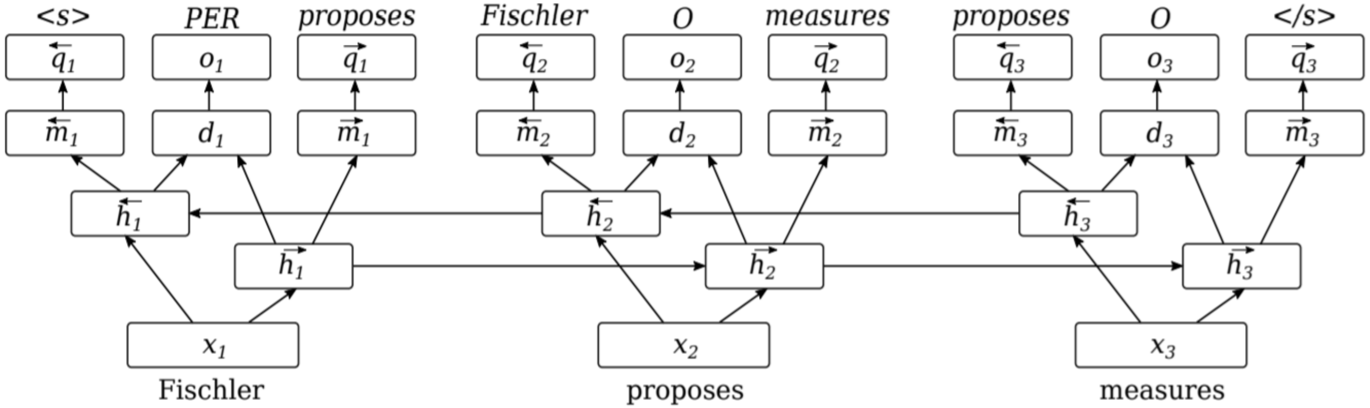


Figure 1: The unfolded network structure for a sequence labeling model with an additional language modeling objective, performing NER on the sentence *"Fischler proposes measures"*. The input tokens are shown at the bottom, the expected output labels are at the top. Arrows above variables indicate the directionality of the component (forward or backward).

Deep RNN architecture

Multi-task RNN architecture

Marek Rei . 2017. Semi-supervised Multitask Learning for Sequence Labeling

RNN in Practice: Training Tips

Weight Initialization Methods

➢ *Identity weight initialization with ReLU activation*

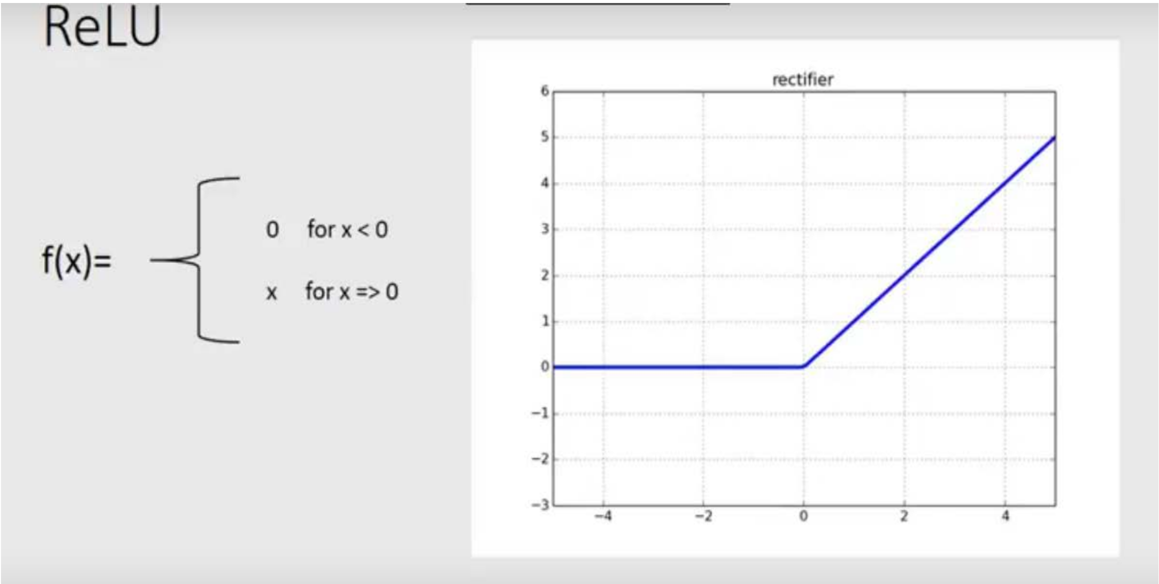
$$\frac{\partial h_t}{\partial \mathrm{h}_k} = \prod_{t \geq i > k} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{t \geq i > k} \mathrm{W}_{hh}{}^{T} diag[g'(h_{i-1})]$$

Activation Function: **ReLU**

i.e., ReLU(x) = *max*{0,x}

And it's gradient = 0 for x < 0 and 1 for x > 0

Therefore,



RNN in Practice: Training Tips

Weight Initialization Methods **(in Vanilla RNNs)**

- Random $\mathbf{W}_{hh}$ initialization of RNN → no constraint on eigenvalues

➡ ***vanishing or exploding gradients in the initial epoch***

- Careful initialization of $\mathbf{W}_{hh}$ with suitable eigenvalues
  - → $\mathbf{W}_{hh}$ initialized to *Identity* matrix
  - → Activation function: *ReLU*

- allows the RNN to learn in the initial epochs
- can generalize well for further iterations

What else?

→ *Batch Normalization:* faster convergence

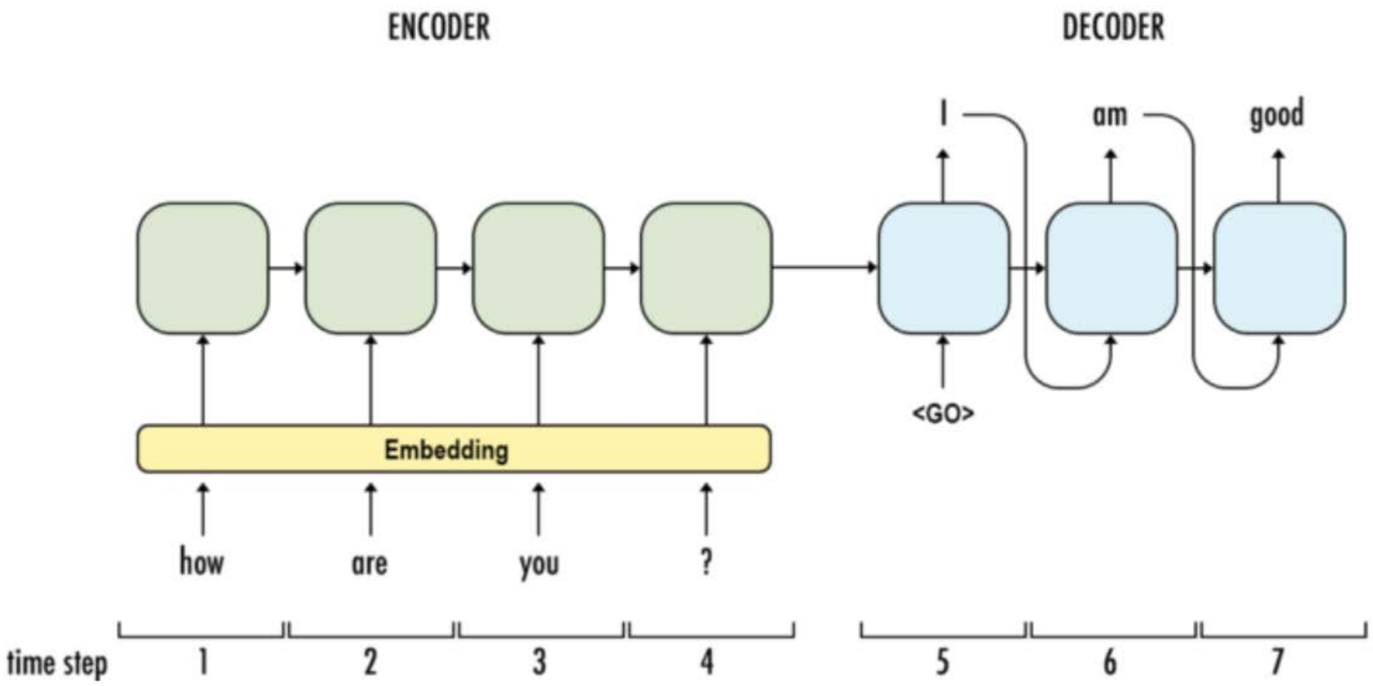
→ *Dropout:* better generalization

Geoffrey et al, "A Simple Way to Initialize Recurrent Networks of Rectified Linear Units"

Attention Mechanism: Attentive RNNs

→Translation often requires arbitrary input length and output length

→ Encode-decoder can be applied to N-to-M sequence, but is one hidden state really enough?



https://medium.com/syncedreview/a-brief-overview-of-attention-mechanism-13c578ba9129

Attention Mechanism: Attentive RNNs

Attention to improve the performance of the Encoder-Decoder RNN on machine translation.

→ allows to focus on local or global features

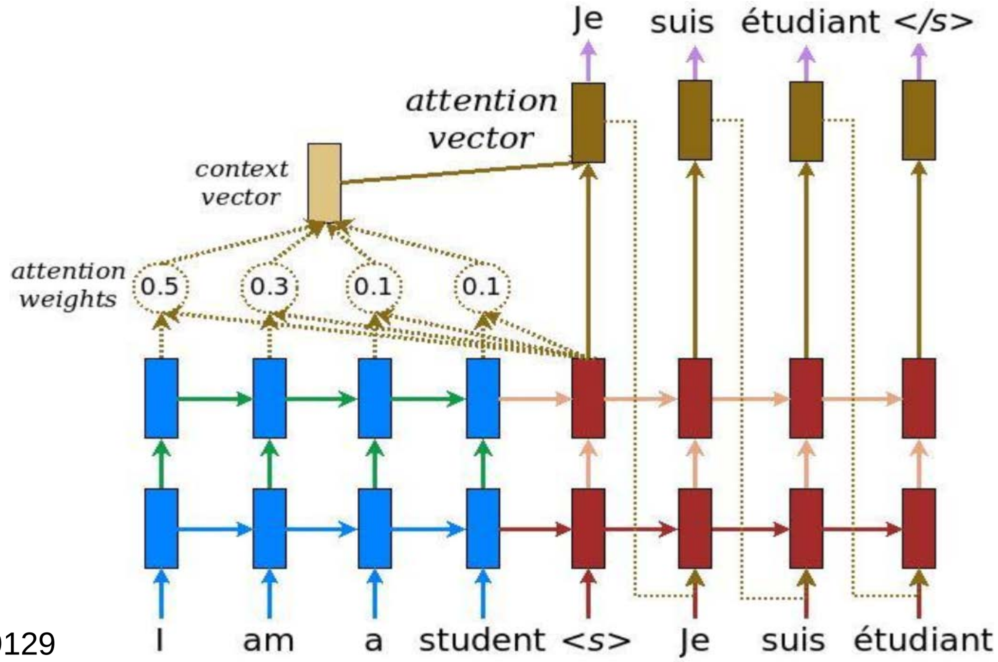
→ is a vector, often the outputs of dense layer using softmax function

→ generates a context vector into the gap between encoder and decoder

Context vector

→ takes all cells' outputs as input

→ compute the probability distribution of source language words for each word in decoder (e.g., 'Je')



https://medium.com/syncedreview/a-brief-overview-of-attention-mechanism-13c578ba9129

How does it Work?

Idea: Compute Context vector for every output/target word, t (during decoding)

For each target word, t

1. *generate scores between each encoder state* **h_s** *and the target state* **h_t**
2. *apply softmax to normalize scores* → *attention weights*

(the probability distribution conditioned on the target state)

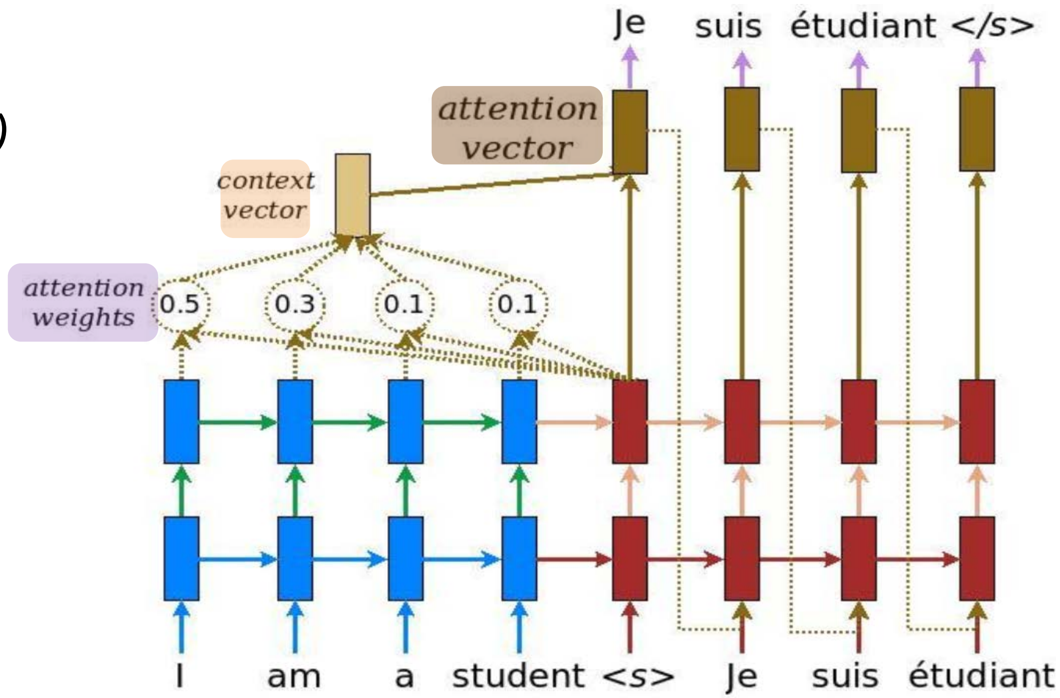
$$\alpha_{ts} = \frac{\exp\left(\operatorname{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\operatorname{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)}$$

3. *compute context vector for the target word, t*

   using attention weights $c_t = \sum_s \alpha_{ts} \bar{\boldsymbol{h}}_s$

4. *compute attention vector for the target word, t*

$$\boldsymbol{a}_t = f(\boldsymbol{c}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{W}_c[\boldsymbol{c}_t; \boldsymbol{h}_t])$$



https://medium.com/syncedreview/a-brief-overview-of-attention-mechanism-13c578ba9129

Explainability/Interpretability of RNNs

Visualization

→Visualize *output predictions:* ***LISA***

→ Visualize *neuron activations: Sensitivity Analysis*

Further Details:

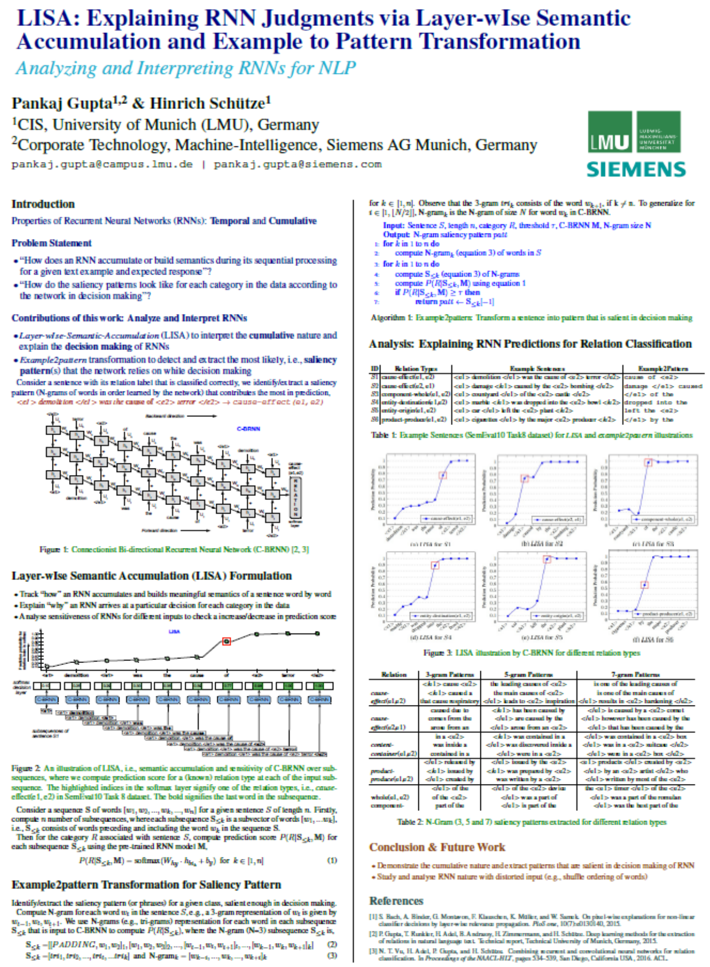
- ***Gupta*** *et al, 2018*. *"LISA: Explaining Recurrent Neural Network Judgments via* ***Layer-wIse Semantic Accumulation*** *and Example to Pattern Transformation". https://arxiv.org/abs/1808.01591*
- *Andrej Karpathy, Blog on "Unreasonable Effectiveness of Recurrent Neural Networks"*
- *Hendrick et al, "Visual Analysis of Hidden State Dynamics in Recurrent Neural Networks"*

Explainability/Interpretability of RNNs

→ Visualize *output predictions:* ***LISA***

Checkout our POSTER about LISA paper (EMNLP2018 conference)

https://www.researchgate.net/publication/328956863_LISA_Explaining_RNN_Judgments_via_Layer-wIse_Semantic_Accumulation_and_Example_to_Pattern_Transformation_Analyzing_and_Interpreting_RNNs_for_NLP



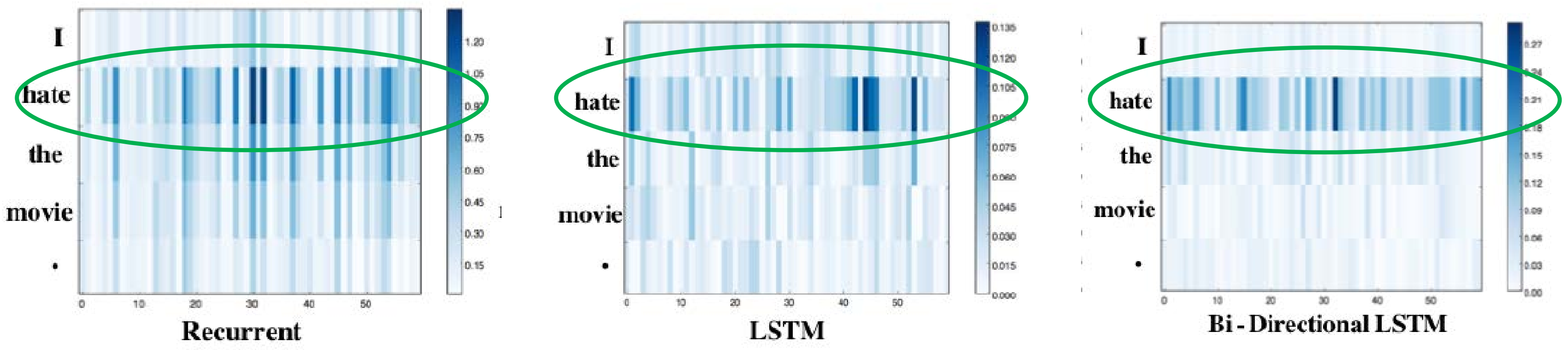
Full paper:

Gupta et al, 2018. "LISA: Explaining Recurrent Neural Network Judgments via ***Layer-wIse Semantic Accumulation*** *and Example to Pattern Transformation".*
https://arxiv.org/abs/1808.01591

Explainability/Interpretability of RNNs

Visualize *neuron activations via Heat maps, i.e.* Sensitivity Analysis

Figure below shows the plot of the sensitivity score .Each row corresponds to saliency score for the correspondent word representation with each grid representing each dimension.



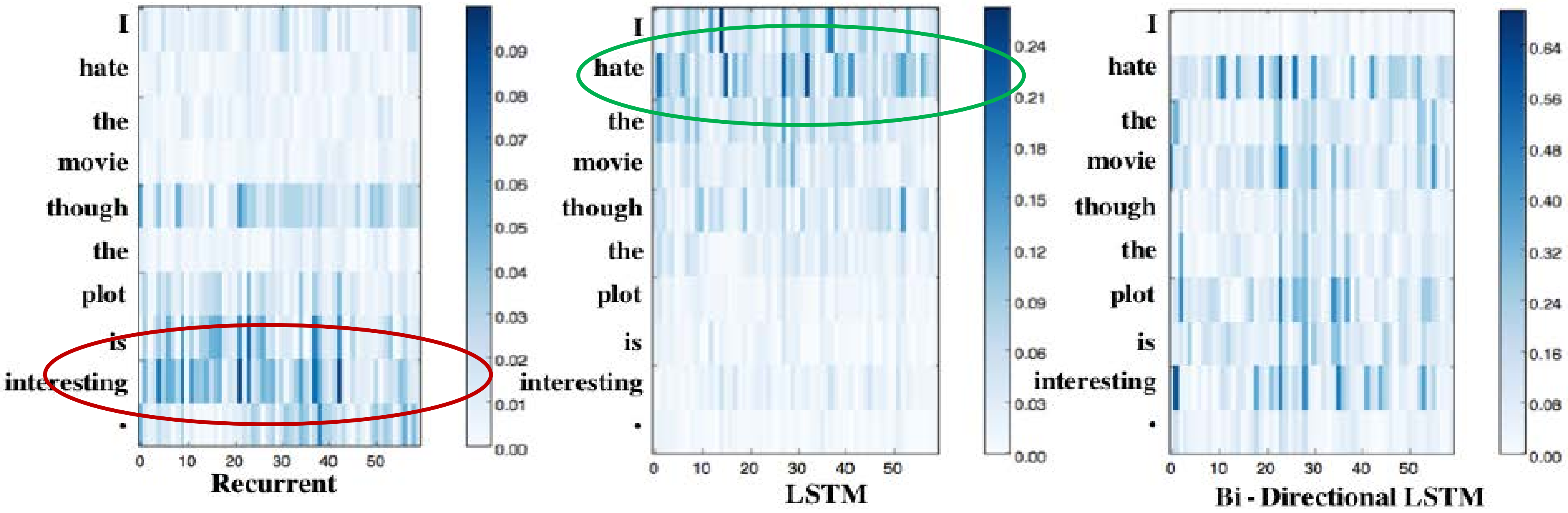
All three models assign high sensitivity to “hate” and dampen the influence of other tokens. LSTM offers a clearer focus on “hate” than the standard recurrent model, but the bi-directional LSTM shows the clearest focus, attaching almost zero emphasis on words other than “hate”. This is presumably due to the gates structures in LSTMs and Bi-LSTMs that controls information flow, making these architectures better at filtering out less relevant information.

LSTM and RNN capture short-term depdendency

Jiwei LI et al, “Visualizing and Understanding Neural Models in NLP”

Explainability/Interpretability of RNNs

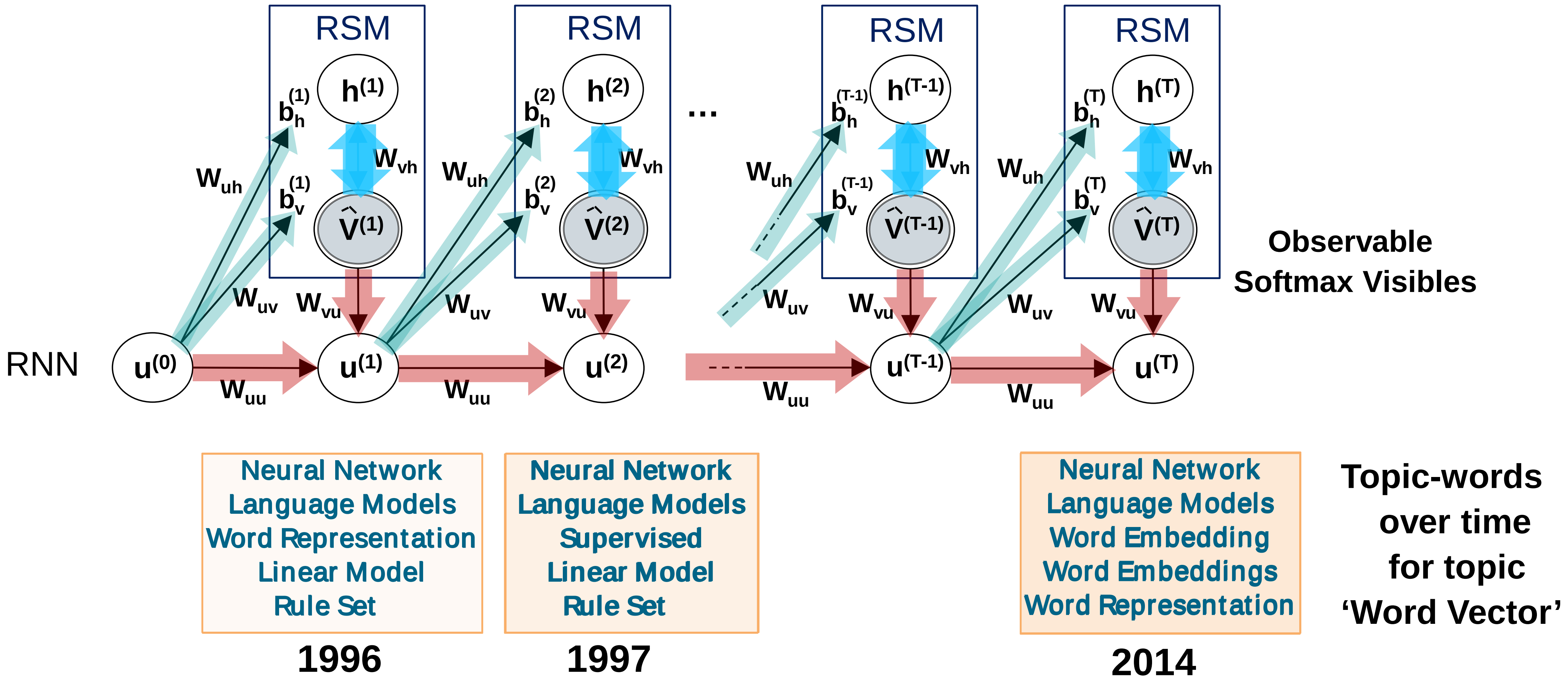
Visualize *neuron activations via Heat maps, i.e.* Sensitivity Analysis



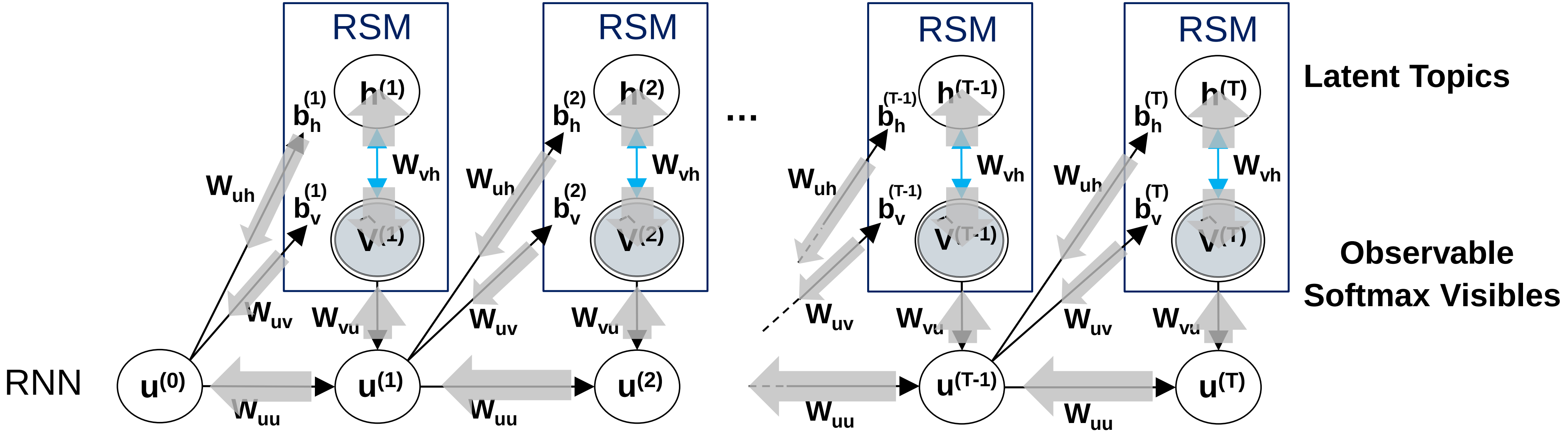
LSTM captures long-term depdendency, (vanilla) RNN not.

Jiwei LI et al, “Visualizing and Understanding Neural Models in NLP”

RNNs in Topic Trend Extraction (Dynamic Topic Evolution): RNN-RSM



Gupta et al. 2018. *Deep Temporal-Recurrent-Replicated-Softmax for Topical Trends over Time*



Cost in RNN-RSM, the negative log-likelihood

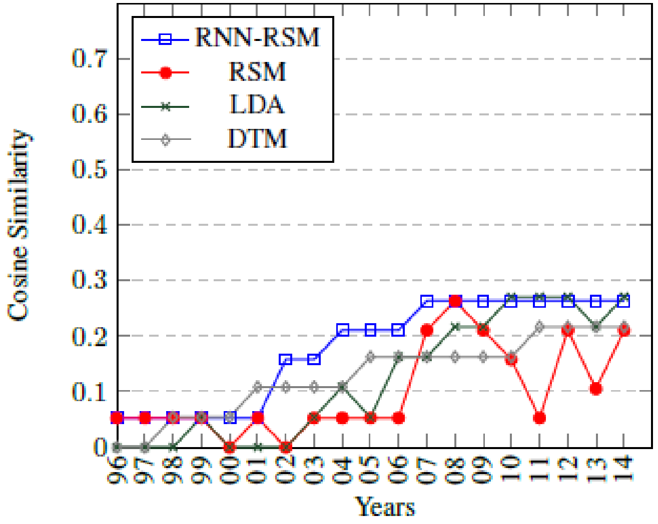
$$C = \sum_{t=1}^{T} C_t \equiv \sum_{t=1}^{T} - \ln P(\widehat{\mathbf{V}}^{(t)})$$

Training via BPTT

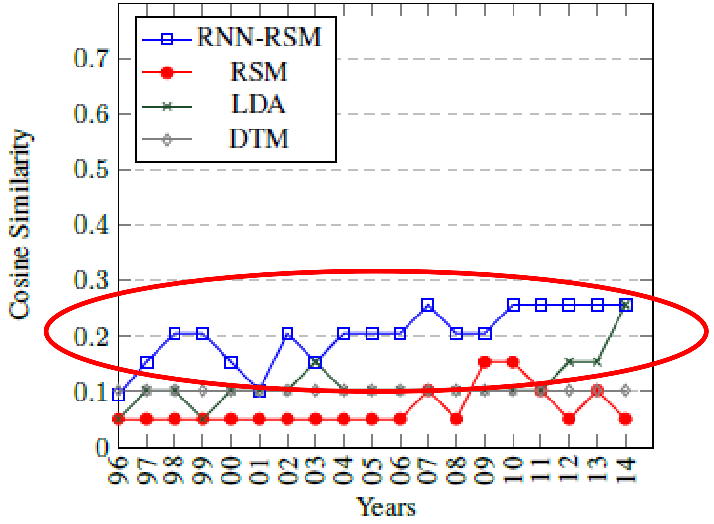
Gupta et al. 2018. *Deep Temporal-Recurrent-Replicated-Softmax for Topical Trends over Time*

RNNs in Topic Trend Extraction (Dynamic Topic Evolution): RNN-RSM

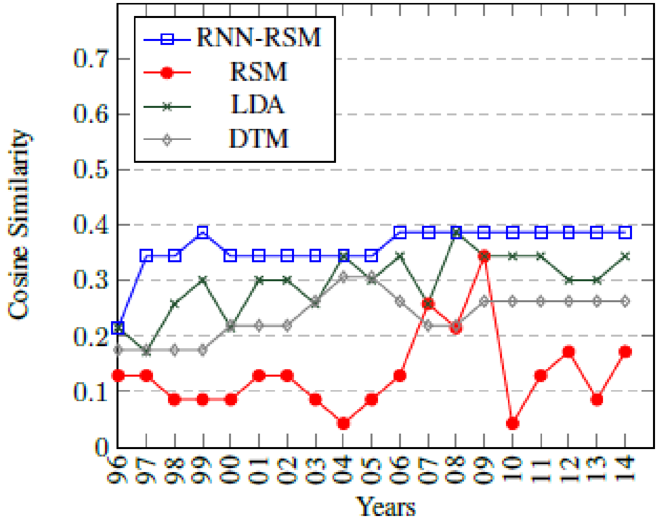
Topic Trend Extraction or Topic Evolution in NLP research over time



Topic: Sentiment Analysis



Topic: Word Vector



Topic: Dependency Parsing

Gupta et al. 2018. *Deep Temporal-Recurrent-Replicated-Softmax for Topical Trends over Time*

Key Takeaways

- RNNs model sequential data
- Long term dependencies are a major problem in RNNs

  Solution:

  → careful weight initialization

  → LSTM/GRUs
- Gradients Explodes

  Solution: → Gradient norm clipping
- Regularization (Batch normalization and Dropout) and attention help
- Interesting direction to visualize and interpret RNN learning

References, Resources and Further Reading

- RNN lecture (Ian Goodfellow): https://www.youtube.com/watch?v=ZVN14xYm7JA
- Andrew Ng lecture on RNN: https://www.coursera.org/lecture/nlp-sequence-models/why-sequence-models-0h7gT
- Recurrent Highway Networks (RHN)
- LSTMs for Language Models (Lecture 07)
- Bengio et al,. "On the difficulty of training recurrent neural networks." (2012)
- Geoffrey et al, “Improving Perfomance of Recurrent Neural Network with ReLU nonlinearity”
- Geoffrey et al, “A Simple Way to Initialize Recurrent Networks of Rectified Linear Units”
- Cooijmans, Tim, et al. "Recurrent batch normalization."(2016).
- Dropout : A Probabilistic Theory of Deep Learning, Ankit B. Patel, Tan Nguyen, Richard G. Baraniuk.
- Barth (2016) : “Semenuita et al. 2016. “Recurrent dropout without memory loss”
- Andrej Karpathy, Blog on “Unreasonable Effectiveness of Recurrent Neural Networks”
- Ilya Sutskever, et al. 2014. “Sequence to Sequence Learning with Neural Networks”
- Bahdanau et al. 2014. “Neural Machine Translation by Jointly Learning to Align and Translate”
- Hierarchical Attention Networks for Document Classification, 2016.
- Attention-Based Bidirectional Long Short-Term Memory Networks for Relation Classification, 2016
- Good Resource: http://slazebni.cs.illinois.edu/spring17/lec20_rnn.pdf

References, Resources and Further Reading

- Lecture from the course Neural Networks for Machine Learning by Greff Hinton
- Lecture by Richard Socher: https://cs224d.stanford.edu/lectures/CS224d-Lecture8.pdf
- Understanding LSTM: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Recursive NN: http://www.iro.umontreal.ca/~bengioy/talks/gss2012-YB6-NLP-recursive.pdf
- Attention: https://medium.com/syncedreview/a-brief-overview-of-attention-mechanism-13c578ba9129
- Gupta, 2015. *Master Thesis on "Deep Learning Methods for the Extraction of Relations in Natural Language Text"*
- Gupta et al., 2016. *Table Filling Multi-Task Recurrent Neural Network for Joint Entity and Relation Extraction.*
- Vu et al., 2016. *Combining recurrent and convolutional neural networks for relation classification.*
- Vu et al., 2016. Bi-directional recurrent neural network with ranking loss for spoken language understanding.
- Gupta et al. 2018. *Deep Temporal-Recurrent-Replicated-Softmax for Topical Trends over Time*
- Gupta et al., 2018. *LISA: Explaining Recurrent Neural Network Judgments via Layer-wIse Semantic Accumulation and Example to Pattern Transformation.*
- Gupta et al., 2018. *Replicated Siamese LSTM in Ticketing System for Similarity Learning and Retrieval in Asymmetric Texts*.
- Gupta et al., 2019. *Neural Relation Extraction Within and Across Sentence Boundaries*
- *Talk/slides*: https://vimeo.com/277669869

Thanks !!!

Write me, if interested in ….

firstname.lastname@siemens.com

@Linkedin: https://www.linkedin.com/in/pankaj-gupta-6b95bb17/

About my research contributions:

https://scholar.google.com/citations?user=_YjIJF0AAAAJ&hl=en