COS 484/584

(Advanced) Natural Language Processing

L12: Recurrent Neural Networks (II)

Spring 2021

(Some slides adapted from Chris Manning, Abigail See, Stanford CS231N)

Announcements

- The midterm grades were released and we accept regrades until next Tuesday (EOD)
- COS484
  - max: 50, median: 39.75, std: 5.95
- COS584
  - max: 49, median: 37, std: 5.76

Announcements

- The project guideline (484 and 584 separately) has been released!
- The proposal deadline is changed to **next Friday (March 26)**
  - 1 page, not graded
- 484: 3 students per team
  - Two options: (a) reproducing a recent NLP paper (encouraged) (2) research project
- 584: 1-2 students per team, research project
- If you can't find a partner yet, use the thread in Ed and we will try to help you.

Announcements

- Assignment 2 due today
- Assignment 3 out today and due on **March 31**
- COS484 precept this Friday
  - Our TAs will give a tutorial on PyTorch.
  - If you don't have enough experience with PyTorch before, you are highly encouraged to attend the 484 precept—the programming problems of assignment 3 & 4 require the use fo PyTorch.

Recap: Recurrent Neural Networks

$\mathbf{h}_0 \in \mathbb{R}^h$ is an initial state

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t) \in \mathbb{R}^h$$

$\mathbf{h}_t$: hidden states which store information from $\mathbf{x}_1$ to $\mathbf{x}_t$

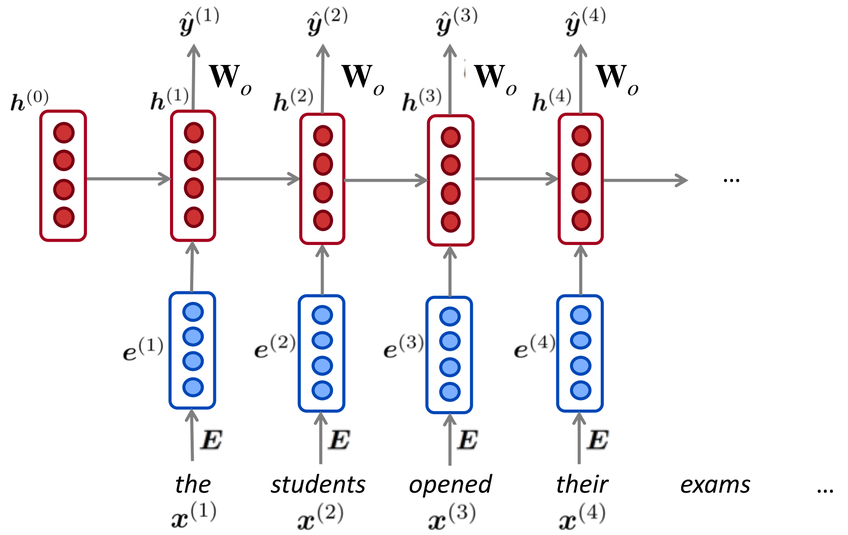
Simple RNNs:

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^h$$

g: nonlinearity (e.g. tanh),

$\mathbf{W} \in \mathbb{R}^{h \times h}, \mathbf{U} \in \mathbb{R}^{h \times d}, \mathbf{b} \in \mathbb{R}^h$

RNNLMs:



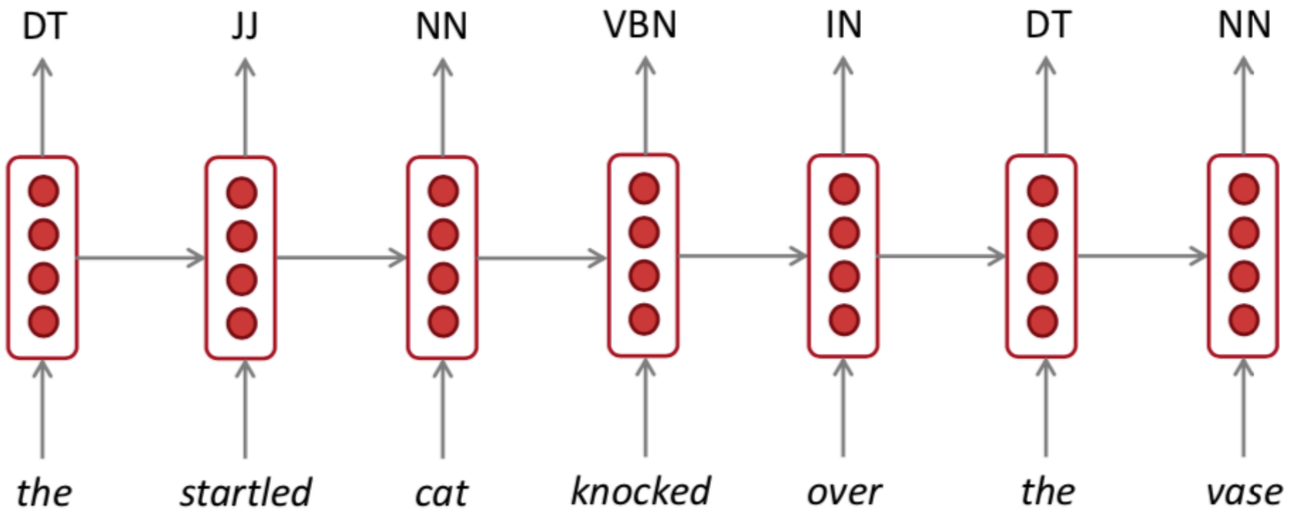
This lecture

- More applications: sequence tagging, text classification
- More variants: multi-layer RNNs, bidirectional RNNs
- More advanced types of RNNs: LSTMs, GRUs

Application: Sequence Tagging

Input: a sentence of n words: $x_1, \ldots, x_n$

Output: $y_1, \ldots, y_n, y_i \in \{1,\ldots C\}$



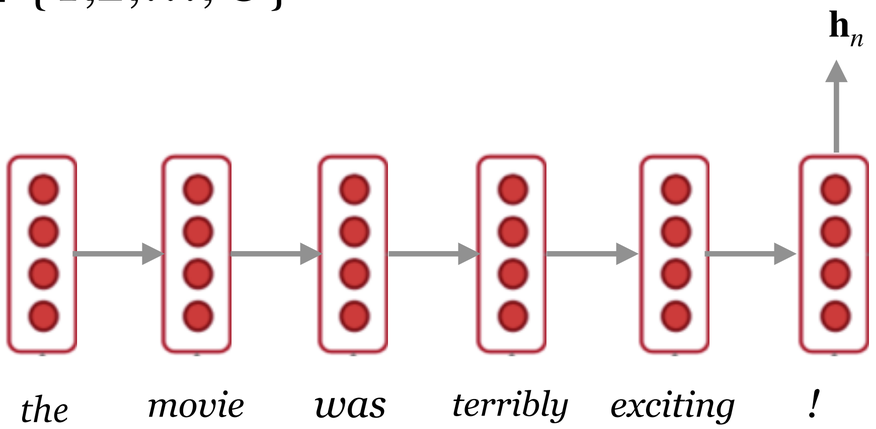
$$P(y_i = k) = softmax_k(\mathbf{W}_o \mathbf{h}_i) \qquad \mathbf{W}_o \in \mathbb{R}^{C \times h} \qquad L = -\frac{1}{n}\sum_{i=1}^{n} \log P(y_i = k)$$

A better solution is to use RNNs + conditional random field (CRF), see Lample et al., 2016 for more details

Application: Text Classification

Input: a sentence of n words

Output: $y \in \{1,2,\ldots,C\}$



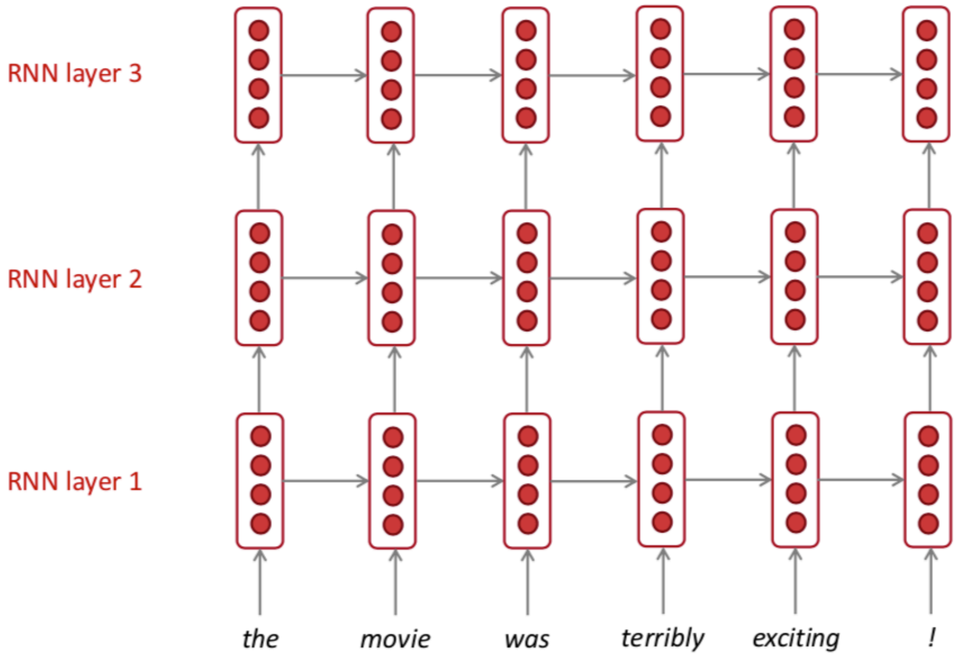
$$P(y = k) = softmax_k(\mathbf{W}_o \mathbf{h}_n) \qquad \mathbf{W}_o \in \mathbb{R}^{C \times h}$$

$$L = -\log P(y = c)$$

Multi-layer RNNs

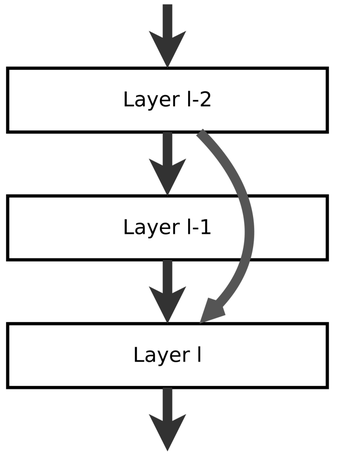
- RNNs are already “deep” on one dimension (unroll over time steps)
- We can also make them “deep” in another dimension by applying multiple RNNs
- Multi-layer RNNs are also called **stacked RNNs**.

Multi-layer RNNs



The hidden states from RNN layer i are the inputs to RNN layer $i + 1$

skip-connections

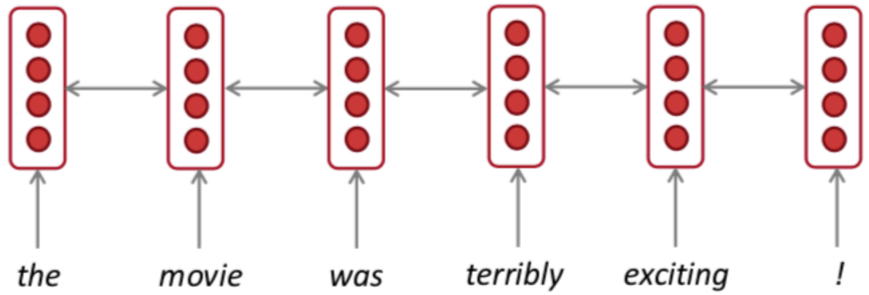


- In practice, using 2 to 4 layers is common (usually better than 1 layer)
- Transformer networks can be up to 24 layers with lots of skip-connections.

Bidirectional RNNs

$\mathbf{h}_t$: hidden states which store information from $\mathbf{x}_1$ to $\mathbf{x}_t$

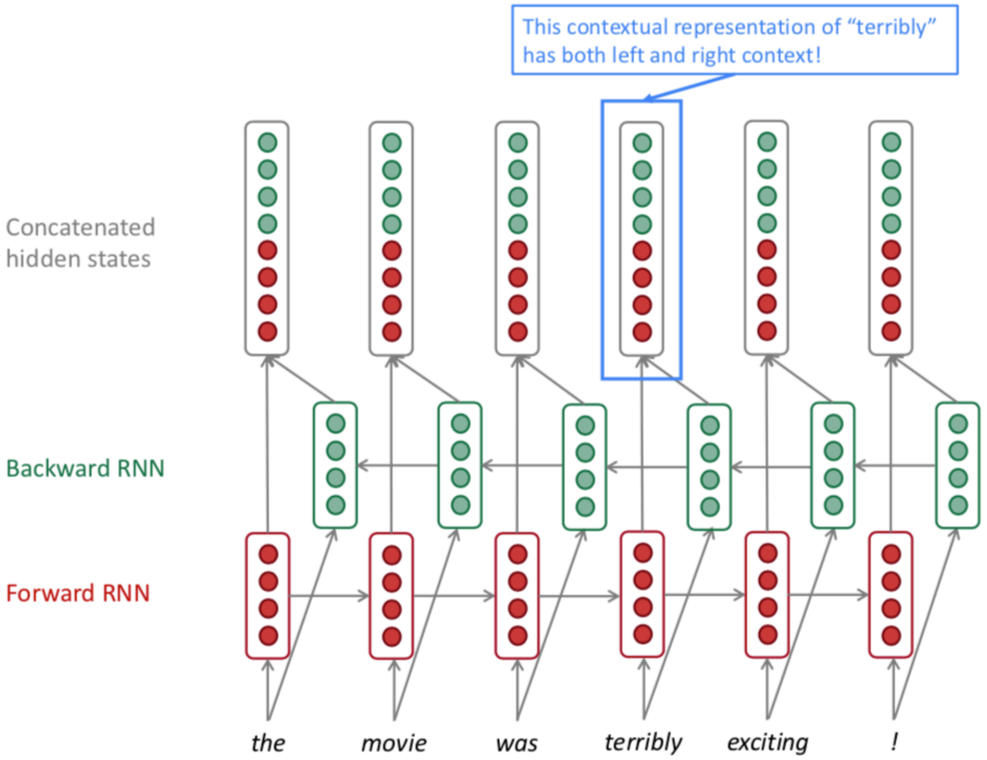
- Bidirectionality is important in language representations:



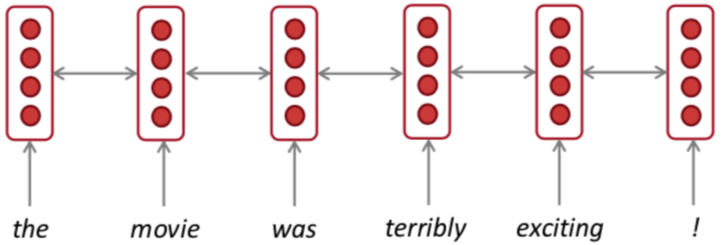
terribly:

- left context “the movie was”
- right context “exciting !”

Bidirectional RNNs



=



$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t) \in \mathbb{R}^h$$

$$\overrightarrow{\mathbf{h}}_t = f_1(\overrightarrow{\mathbf{h}}_{t-1}, \mathbf{x}_t), t = 1,2,\ldots n$$

$$\overleftarrow{\mathbf{h}}_t = f_2(\overleftarrow{\mathbf{h}}_{t+1}, \mathbf{x}_t), t = n, n-1,\ldots 1$$

$$\mathbf{h}_t = [\overleftarrow{\mathbf{h}}_t, \overrightarrow{\mathbf{h}}_t] \in \mathbb{R}^{2h}$$

Zoom poll

Can we use bidirectional RNNs in the following tasks?

(1) text classification, (2) sequence tagging, (3) text generation

(a) Yes, Yes, Yes

(b) Yes, No, Yes

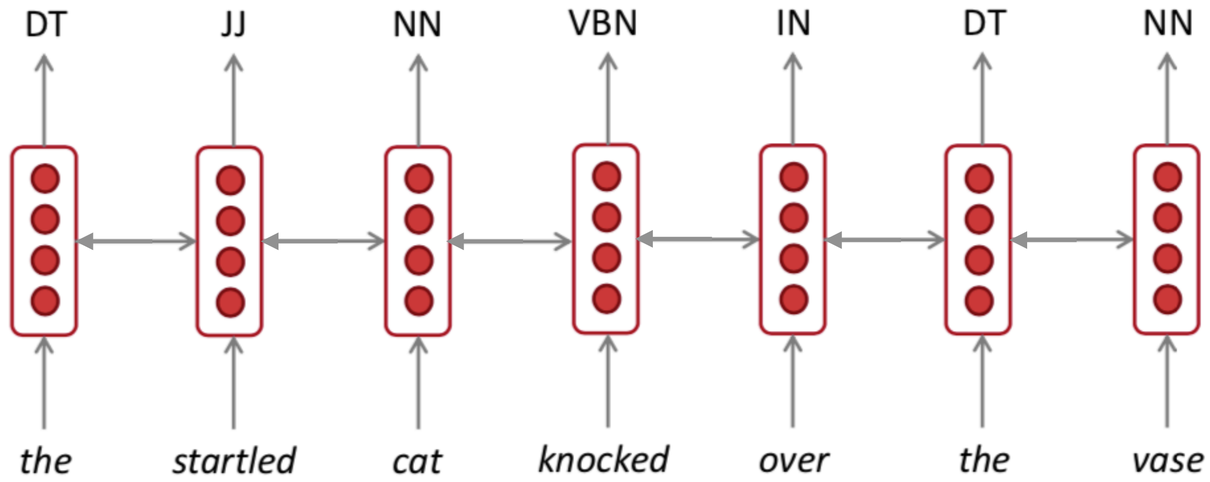
(c) Yes, Yes, No

(d) No, Yes, No

The answer is (c).

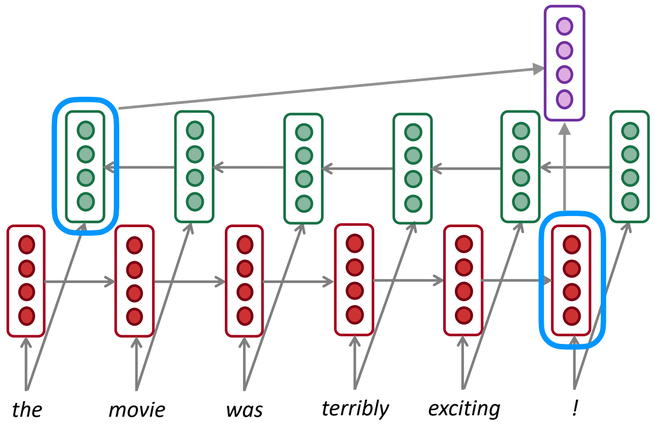
Bidirectional RNNs

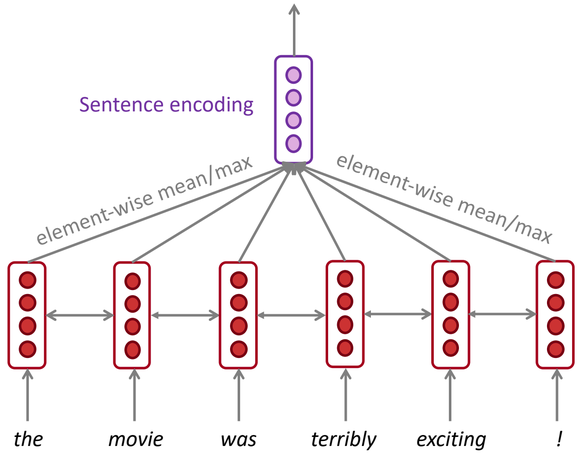
- Sequence tagging: Yes! (esp. important)



Bidirectional RNNs

- Sequence tagging: Yes!
- Text classification: Yes!
  - Common practice: concatenate the last hidden vectors in two directions or take the mean/ max over all the hidden vectors





- Text generation: No. Because we can't see the future to predict the next word.

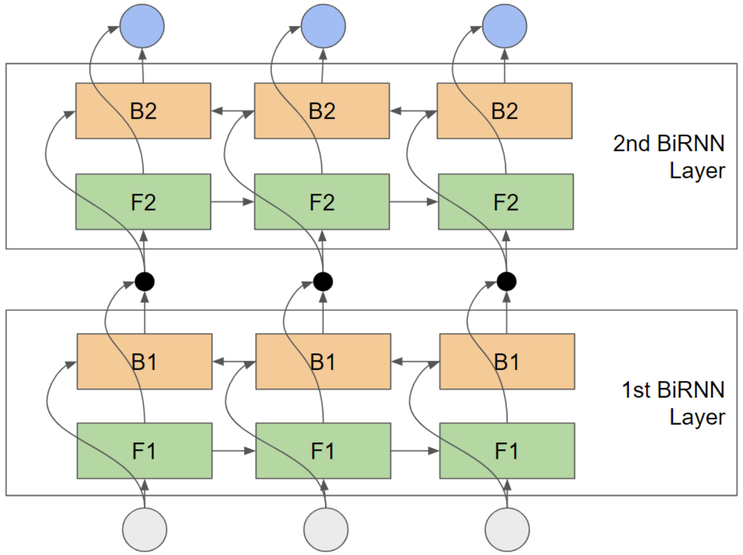
Bidirectional RNNs

- Bidirectional RNNs are only applicable if you have access to the **entire input sequence**

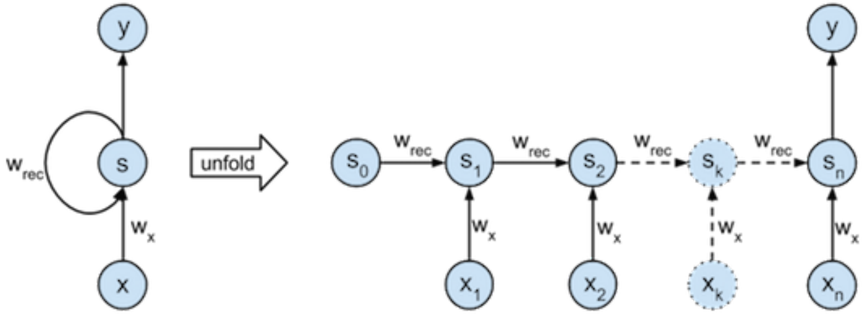
- If you do have entire input sequence, bidirectionality is powerful (and you should use it by default)

- Modeling the bidirectionality is the key idea behind BERT (BERT = **Bidirectional** Encoder Representations from Transformers)
  - We will learn Transformers and BERT in a few weeks!

- A very common choice for sentence/document modeling: multi-layer bidirectional RNNs



Advanced RNN variants



$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t) \in \mathbb{R}^h$$

$$\mathbf{h}_t = \tanh(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^h$$

LSTMs

$$\mathbf{i}_t = \sigma(\mathbf{W}^i\mathbf{h}_{t-1} + \mathbf{U}^i\mathbf{x}_t + \mathbf{b}^i)$$

$$\mathbf{f}_t = \sigma(\mathbf{W}^f\mathbf{h}_{t-1} + \mathbf{U}^f\mathbf{x}_t + \mathbf{b}^f)$$

$$\mathbf{o}_t = \sigma(\mathbf{W}^o\mathbf{h}_{t-1} + \mathbf{U}^o\mathbf{x}_t + \mathbf{b}^o)$$

$$\mathbf{g}_t = \tanh(\mathbf{W}^g\mathbf{h}_{t-1} + \mathbf{U}^g\mathbf{x}_t + \mathbf{b}^g)$$

$$\mathbf{c}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t + \mathbf{g}_t \odot \mathbf{i}_t$$

$$\mathbf{h}_t = \tanh(\mathbf{c}_t) \odot \mathbf{o}_t$$

GRUs

$$\mathbf{r}_t = \sigma(\mathbf{W}^r\mathbf{h}_{t-1} + \mathbf{U}^r\mathbf{x}_t + \mathbf{b}^r)$$

$$\mathbf{z}_t = \sigma(\mathbf{W}^z\mathbf{h}_{t-1} + \mathbf{U}^z\mathbf{x}_t + \mathbf{b}^z)$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{U}\mathbf{x}_t + \mathbf{b})$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t$$

Long Short-Term Memory RNNs (LSTMs)

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the **vanishing gradients problem**.
  - Everyone cites that paper but really a crucial part of the modern LSTM is from Gers et al. (2000)

LONG SHORT-TERM MEMORY

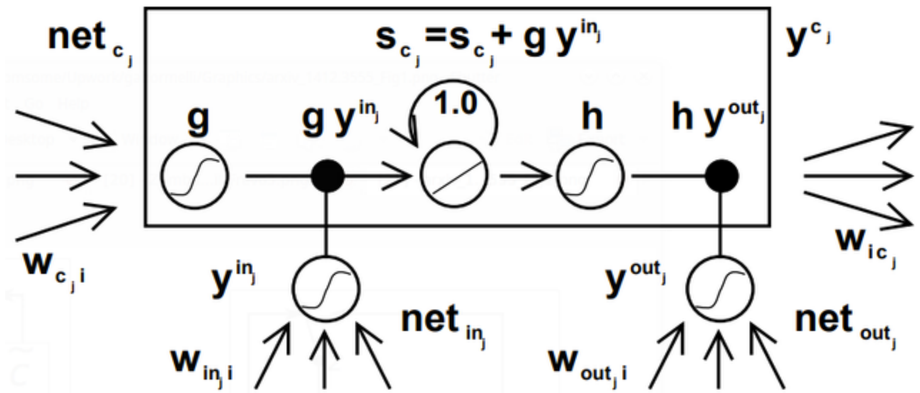
Neural Computation 9(8):1735–1780, 1997

Sepp Hochreiter
Fakultät für Informatik
Technische Universität München
80290 München, Germany
hochreit@informatik.tu-muenchen.de
http://www7.informatik.tu-muenchen.de/~hochreit

Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juergen@idsia.ch
http://www.idsia.ch/~juergen

Learning to Forget: Continual Prediction with LSTM

Felix A. Gers
Jürgen Schmidhuber
Fred Cummins
IDSIA, 6900 Lugano, Switzerland



[advanced]

Recap: Vanishing Gradient Problem

$$\mathbf{h}_2 = g(\mathbf{W}\mathbf{h}_1 + \mathbf{U}\mathbf{x}_2 + \mathbf{b})$$
$$\mathbf{h}_3 = g(\mathbf{W}\mathbf{h}_2 + \mathbf{U}\mathbf{x}_3 + \mathbf{b})$$
$$L_3 = -\log \hat{\mathbf{y}}_3(w_4)$$

$$\frac{\partial L_3}{\partial \mathbf{W}} = \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{W}} + \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_2}{\partial \mathbf{W}} + \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1}\frac{\partial \mathbf{h}_1}{\partial \mathbf{W}}$$

Vanishing gradient problem:
When these are small, the gradient signal gets smaller and smaller as it backpropagates further

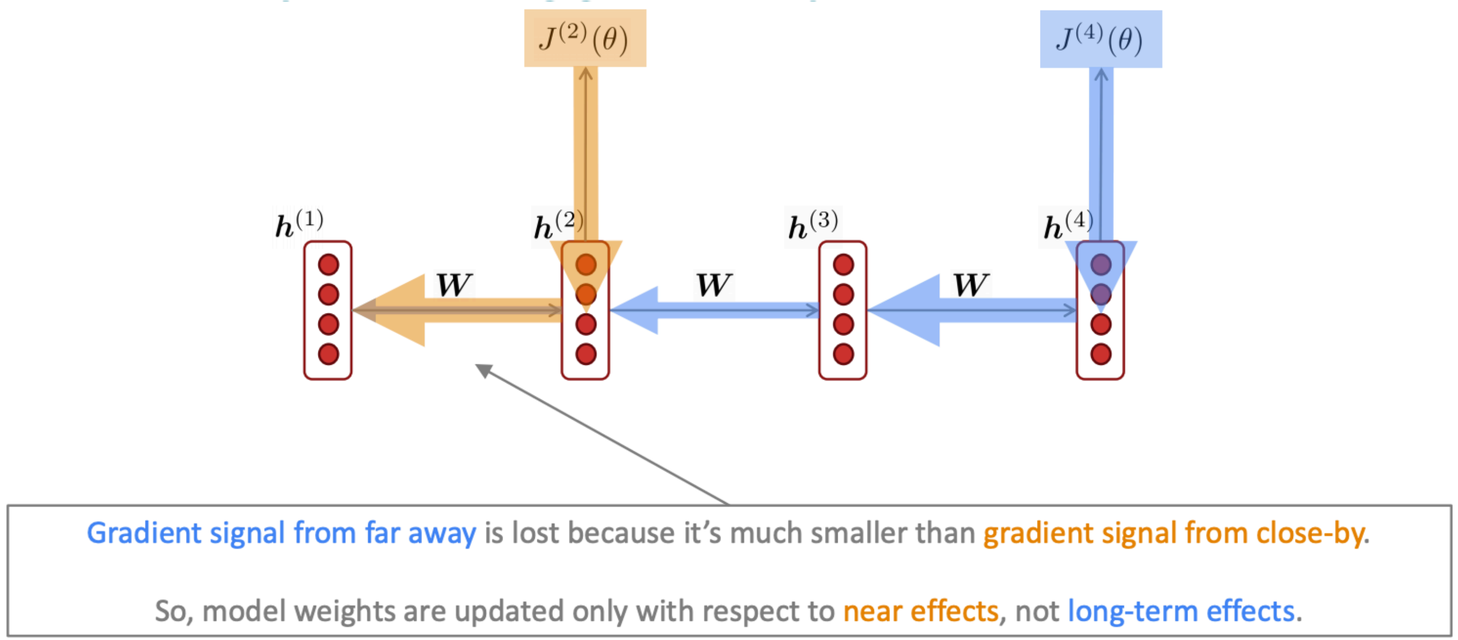
$$\frac{\partial L}{\partial \mathbf{W}} = -\frac{1}{n}\sum_{t=1}^{n}\sum_{k=1}^{t}\frac{\partial L_t}{\partial \mathbf{h}_t}\left(\prod_{j=k+1}^{t}\frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}\right)\frac{\partial \mathbf{h}_k}{\partial \mathbf{W}}$$

If k and t are far away, the gradients are very easy to grow/shrink exponentially

(called the gradient exploding or gradient vanishing problem)

[advanced]

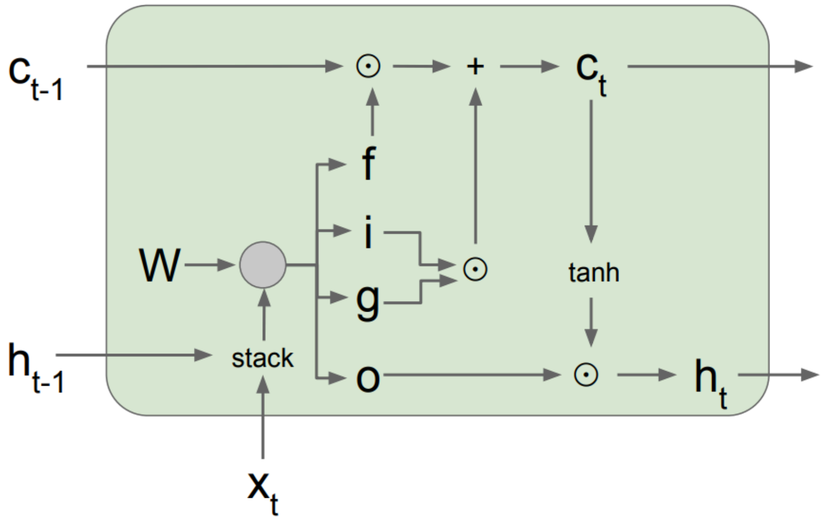
Recap: Vanishing Gradient Problem



When she tried to print her **tickets**, she found that the printer was out of toner.
She went to the stationery store to buy more toner. It was very overpriced.
After installing the toner into the printer, she finally printed her _____

LSTMs: The intuition

- Key idea: turning **multiplication** into **addition** and using "**gates**" to control how much information to add/erase
- At each time step, instead of re-writing the hidden state $\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$, there is also a cell state $\mathbf{c}_t \in \mathbb{R}^h$ which stores **long-term information**
  - We write to/erase information from $\mathbf{c}_t$ after each step
  - We read $\mathbf{h}_t$ from $\mathbf{c}_t$

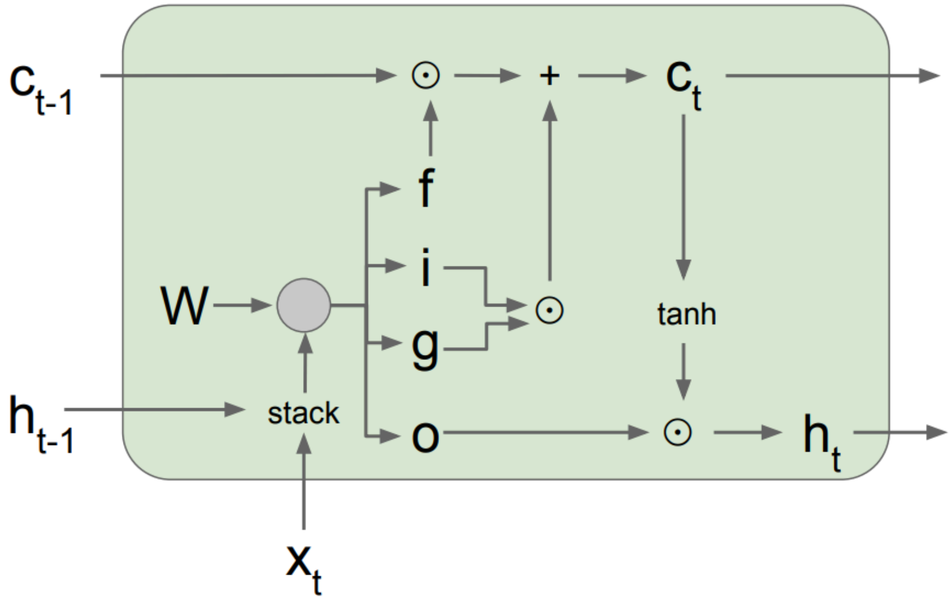


LSTMs: the formulation

- Input gate **(how much to write)**:
  $\mathbf{i}_t = \sigma(\mathbf{W}^i\mathbf{h}_{t-1} + \mathbf{U}^i\mathbf{x}_t + \mathbf{b}^i) \in \mathbb{R}^h$
- Forget gate **(how much to erase)**:
  $\mathbf{f}_t = \sigma(\mathbf{W}^f\mathbf{h}_{t-1} + \mathbf{U}^f\mathbf{x}_t + \mathbf{b}^f) \in \mathbb{R}^h$
- Output gate **(how much to reveal)**:
  $\mathbf{o}_t = \sigma(\mathbf{W}^o\mathbf{h}_{t-1} + \mathbf{U}^o\mathbf{x}_t + \mathbf{b}^{(o)}) \in \mathbb{R}^h$
- New memory cell **(what to write)**:
  $\mathbf{g}_t = \tanh(\mathbf{W}^g\mathbf{h}_{t-1} + \mathbf{U}^g\mathbf{x}_t + \mathbf{b}^g) \in \mathbb{R}^h$
- Final memory cell: $\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$

element-wise product

- Final hidden cell: $\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$



$\mathbf{h}_0, \mathbf{c}_0 \in \mathbb{R}^h$ are initial states (usually set to $\mathbf{0}$)

LSTMs: the formulation

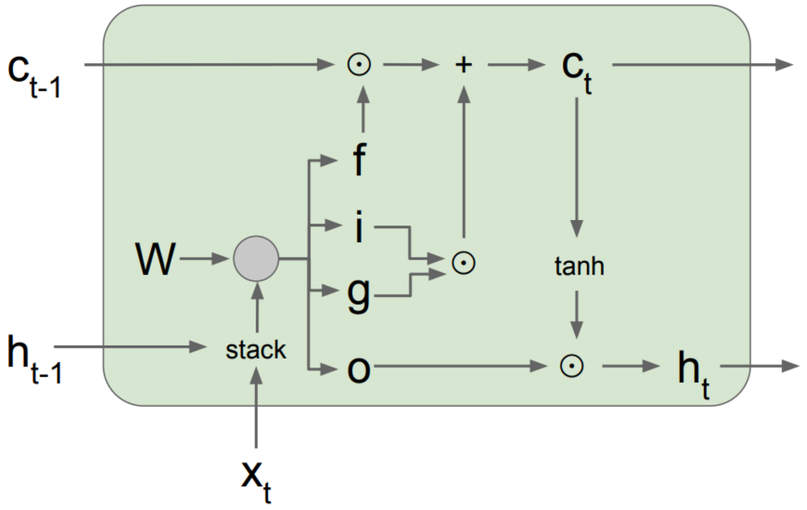
- LSTMs has 4x parameters compared to simple RNNs:

Input dimension: d, hidden size: h

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^h$$

$$\mathbf{W} \in \mathbb{R}^{h\times h}, \mathbf{U} \in \mathbb{R}^{h\times d}, \mathbf{b} \in \mathbb{R}^h$$

$$\mathbf{W}^i, \mathbf{W}^f, \mathbf{W}^g, \mathbf{W}^o \in \mathbb{R}^{h\times h}$$
$$\mathbf{U}^i, \mathbf{U}^f, \mathbf{U}^g, \mathbf{U}^o \in \mathbb{R}^{h\times d}$$
$$\mathbf{b}^i, \mathbf{b}^f, \mathbf{b}^g, \mathbf{b}^o \in \mathbb{R}^h$$

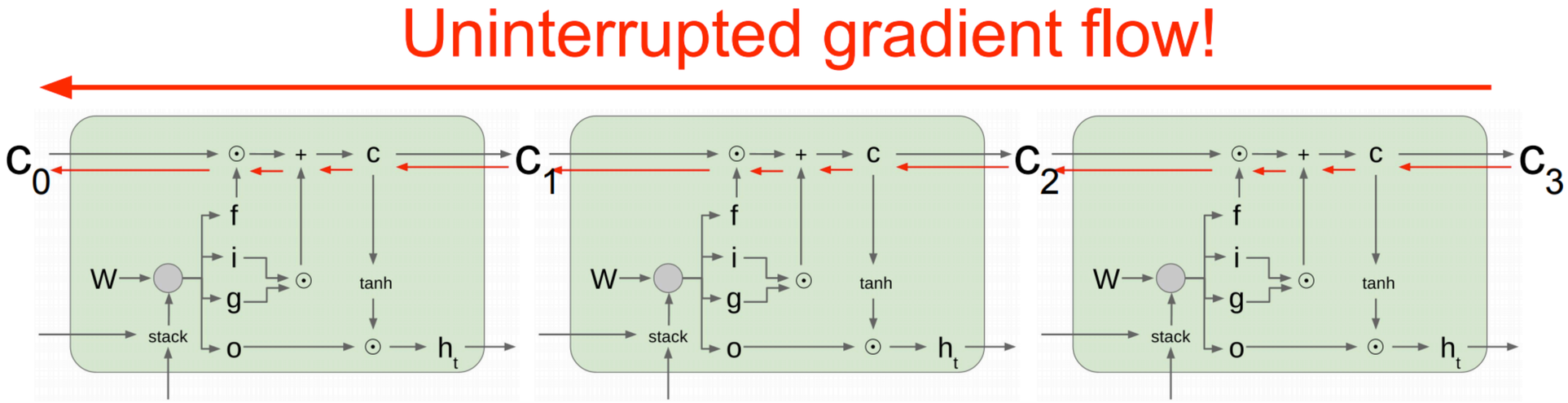


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

Q: What is the range of the hidden representations $\mathbf{h}_t$?

LSTMs: the formulation



- LSTM doesn't guarantee that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

- LSTMs were invented in 1997 but finally got working from 2013-2015.

Gated Recurrent Units (GRUs)

- Introduced by Kyunghyun Cho in 2014:

Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation

Kyunghyun Cho
Bart van Merriënboer **Caglar Gulcehre** Université de Montréal
firstname.lastname@umontreal.ca

Dzmitry Bahdanau Jacobs University, Germany
d.bahdanau@jacobs-university.de

Fethi Bougares **Holger Schwenk** Université du Maine, France
firstname.lastname@lium.univ-lemans.fr

Yoshua Bengio Université de Montréal, CIFAR Senior Fellow
find.me@on.the.web

- Simplified 3 gates to 2 gates: **reset** gate and **update** gate, without an explicit cell state

Gated Recurrent Units (GRUs)

- Reset gate:

$$\mathbf{r}_t = \sigma(\mathbf{W}^r \mathbf{h}_{t-1} + \mathbf{U}^r \mathbf{x}_t + \mathbf{b}^r)$$

- Update gate:

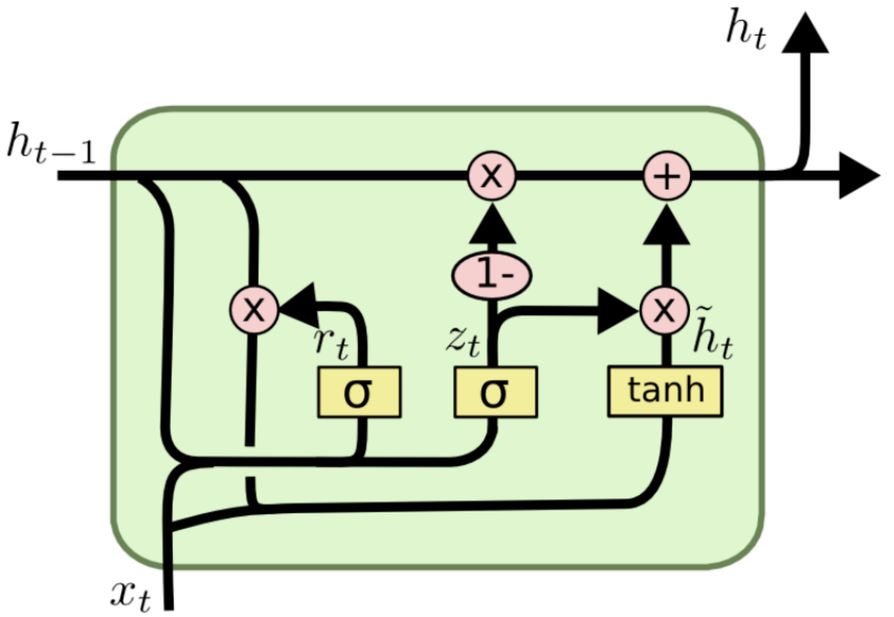
$$\mathbf{z}_t = \sigma(\mathbf{W}^z \mathbf{h}_{t-1} + \mathbf{U}^z \mathbf{x}_t + \mathbf{b}^z)$$

- New hidden state:

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{U}\mathbf{x}_t + \mathbf{b})$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t$$

merge input and forget gate!



Q: What is the range of the hidden representations $\mathbf{h}_t$?

Q: How many parameters are there compared to simple RNNs?

Zoom poll

Let’s compare LSTMs and GRUs. Which of the following statements is correct?

(a) GRUs can be trained faster

(b) In theory LSTMs can capture long-term dependencies better

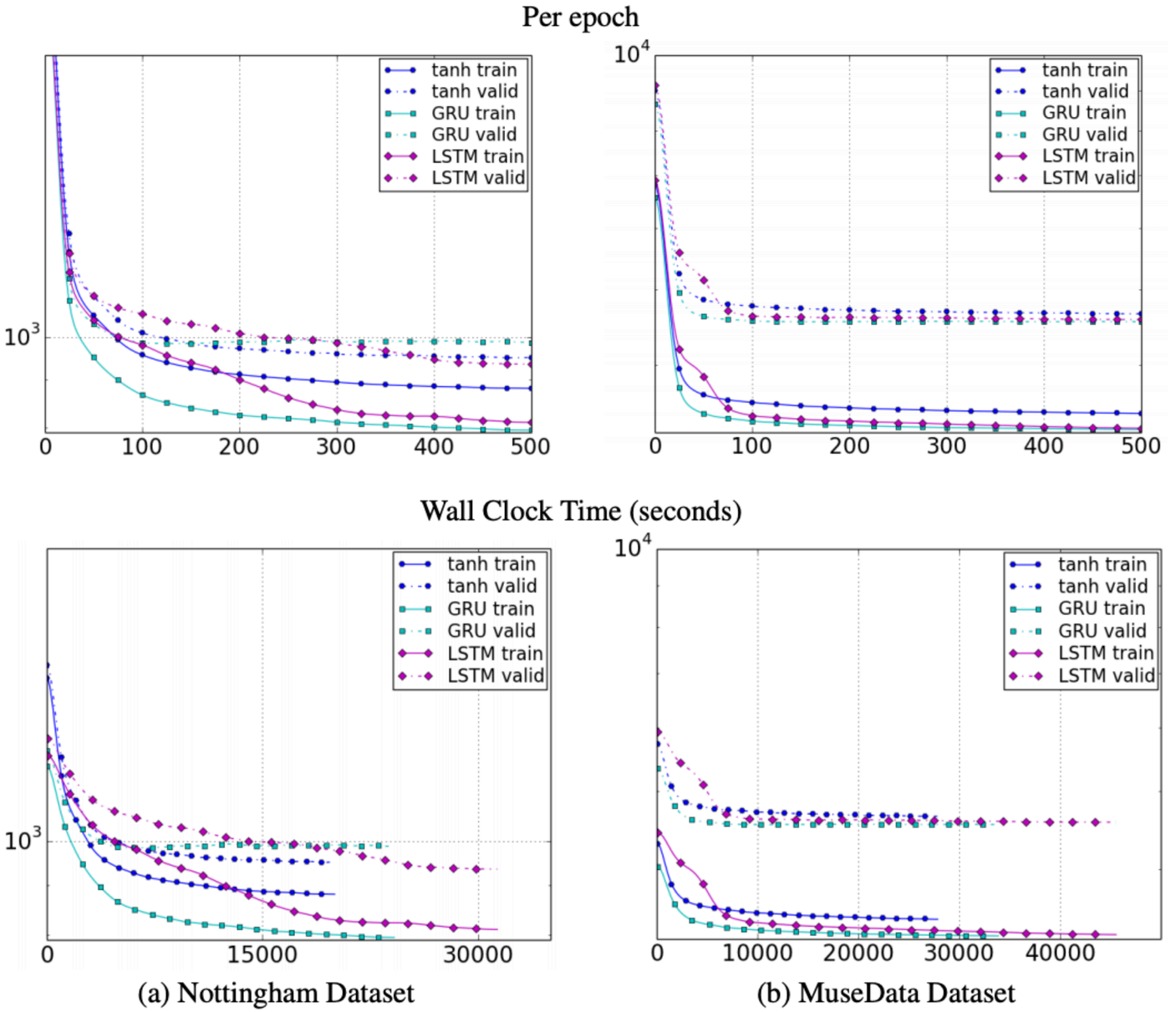
(c) LSTMs have a controlled exposure of memory content while GRUs don’t

(d) None of the above

All of these are correct (a) (b) (c) ☺

LSTMs vs GRUs

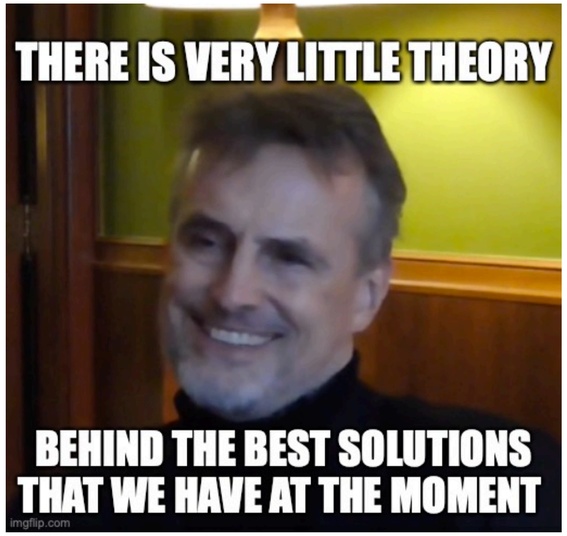
Music modeling



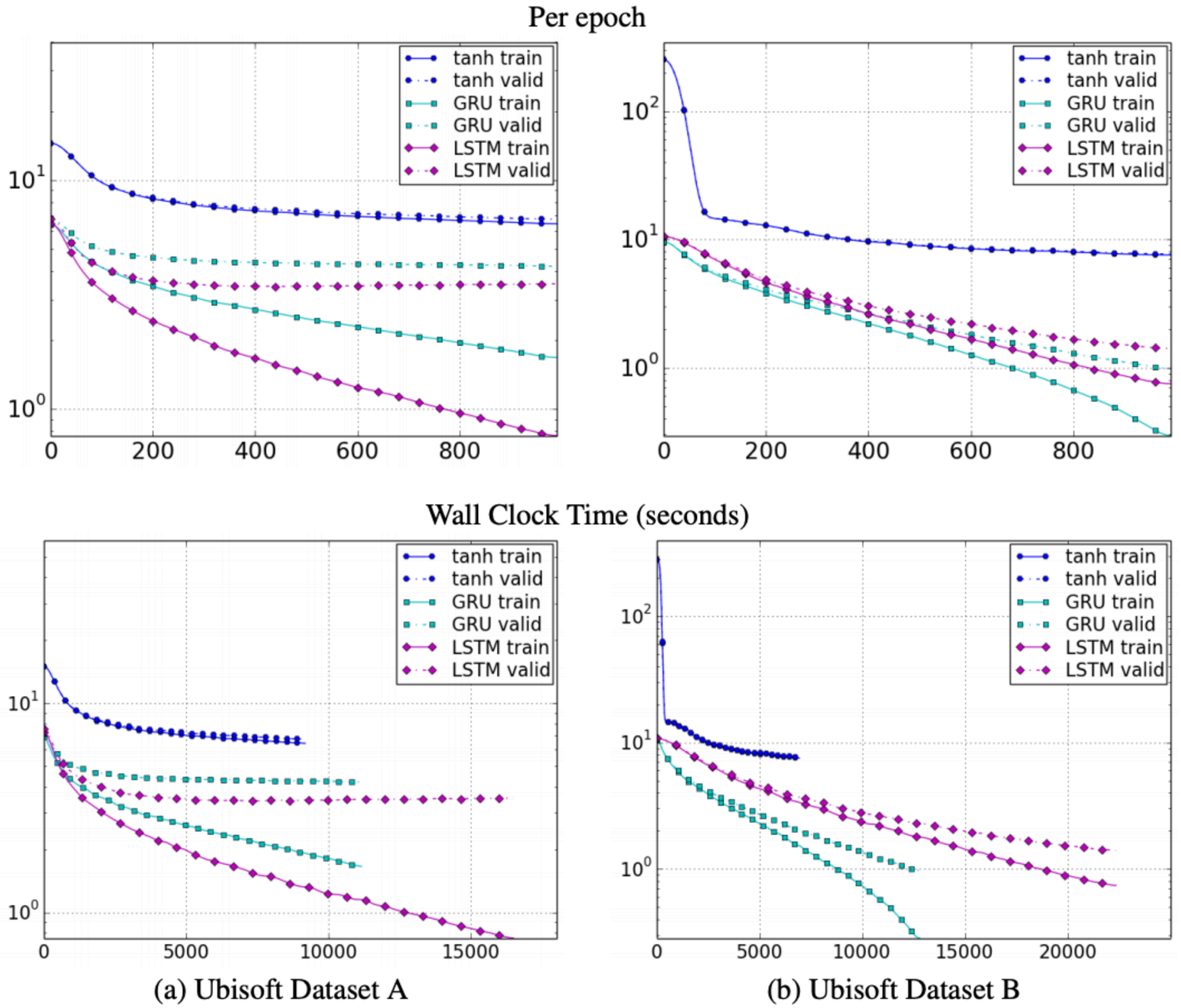
(Chung et al, 2014): Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling

LSTMs vs GRUs

Speech signal modeling



https://imgflip.com/i/495iim
(only for fun!!!)



(Chung et al, 2014): Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling

Recap: progress on language models

On the Penn Treebank (PTB) dataset

Metric: perplexity

Simple RNNs + LDA + Kneser-Ney 5-gram + cache

Model	#Param	Validation	Test
Mikolov & Zweig (2012) – RNN-LDA + KN-5 + cache	9M‡	-	92.0
Zaremba et al. (2014) – LSTM	20M	86.2	82.7
Gal & Ghahramani (2016) – Variational LSTM (MC)	20M	-	78.6
Kim et al. (2016) – CharCNN	19M	-	78.9
Merity et al. (2016) – Pointer Sentinel-LSTM	21M	72.4	70.9
Grave et al. (2016) – LSTM + continuous cache pointer†	-	-	72.1
Inan et al. (2016) – Tied Variational LSTM + augmented loss	24M	75.7	73.2
Zilly et al. (2016) – Variational RHN	23M	67.9	65.4
Zoph & Le (2016) – NAS Cell	25M	-	64.0
Melis et al. (2017) – 2-layer skip connection LSTM	24M	60.9	58.3
Merity et al. (2017) – AWD-LSTM w/o finetune	24M	60.7	58.8
Merity et al. (2017) – AWD-LSTM	24M	60.0	57.3
Ours – AWD-LSTM-MoS w/o finetune	22M	58.08	55.97
Ours – AWD-LSTM-MoS	22M	**56.54**	**54.44**
Merity et al. (2017) – AWD-LSTM + continuous cache pointer†	24M	53.9	52.8
Krause et al. (2017) – AWD-LSTM + dynamic evaluation†	24M	51.6	51.1
Ours – AWD-LSTM-MoS + dynamic evaluation†	22M	**48.33**	**47.69**

(Yang et al, 2018): Breaking the Softmax Bottleneck: A High-Rank RNN Language Model

Are LSTMs and GRUs optimal?

MUT1:

$$
\begin{aligned}
z &= \mathrm{sigm}(W_{\mathrm{xz}} x_t + b_{\mathrm{z}}) \\
r &= \mathrm{sigm}(W_{\mathrm{xr}} x_t + W_{\mathrm{hr}} h_t + b_{\mathrm{r}}) \\
h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + \tanh(x_t) + b_{\mathrm{h}}) \odot z \\
&+ h_t \odot (1 - z)
\end{aligned}
$$

MUT2:

$$
\begin{aligned}
z &= \mathrm{sigm}(W_{\mathrm{xz}} x_t + W_{\mathrm{hz}} h_t + b_{\mathrm{z}}) \\
r &= \mathrm{sigm}(x_t + W_{\mathrm{hr}} h_t + b_{\mathrm{r}}) \\
h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh} x_t + b_{\mathrm{h}}) \odot z \\
&+ h_t \odot (1 - z)
\end{aligned}
$$

MUT3:

$$
\begin{aligned}
z &= \mathrm{sigm}(W_{\mathrm{xz}} x_t + W_{\mathrm{hz}} \tanh(h_t) + b_{\mathrm{z}}) \\
r &= \mathrm{sigm}(W_{\mathrm{xr}} x_t + W_{\mathrm{hr}} h_t + b_{\mathrm{r}}) \\
h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh} x_t + b_{\mathrm{h}}) \odot z \\
&+ h_t \odot (1 - z)
\end{aligned}
$$

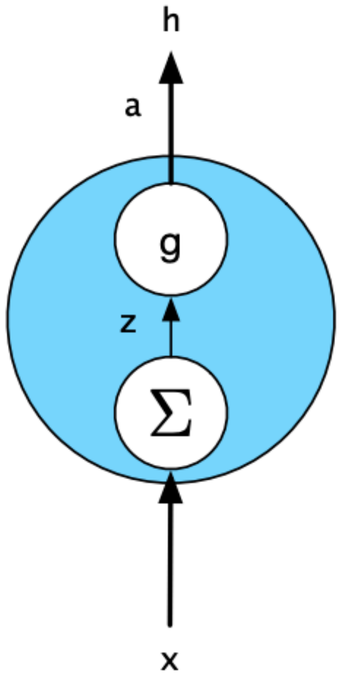
Arch.	Arith.	XML	PTB
Tanh	0.29493	0.32050	0.08782
LSTM	0.89228	0.42470	0.08912
LSTM-f	0.29292	0.23356	0.08808
LSTM-i	0.75109	0.41371	0.08662
LSTM-o	0.86747	0.42117	0.08933
LSTM-b	0.90163	0.44434	0.08952
GRU	0.89565	0.45963	0.09069
MUT1	**0.92135**	**0.47483**	0.08968
MUT2	0.89735	**0.47324**	0.09036
MUT3	0.90728	0.46478	**0.09161**

Arch.	5M-tst	10M-v	20M-v	20M-tst
Tanh	4.811	4.729	4.635	4.582 (97.7)
LSTM	4.699	4.511	4.437	4.399 (81.4)
LSTM-f	4.785	4.752	4.658	4.606 (100.8)
LSTM-i	4.755	4.558	4.480	4.444 (85.1)
LSTM-o	4.708	4.496	4.447	4.411 (82.3)
LSTM-b	4.698	4.437	4.423	**4.380 (79.83)**
GRU	4.684	4.554	4.559	4.519 (91.7)
MUT1	4.699	4.605	4.594	4.550 (94.6)
MUT2	4.707	4.539	4.538	4.503 (90.2)
MUT3	4.692	4.523	4.530	4.494 (89.47)

(Jozefowicz et al, 2015): An Empirical Exploration of Recurrent Network Architectures

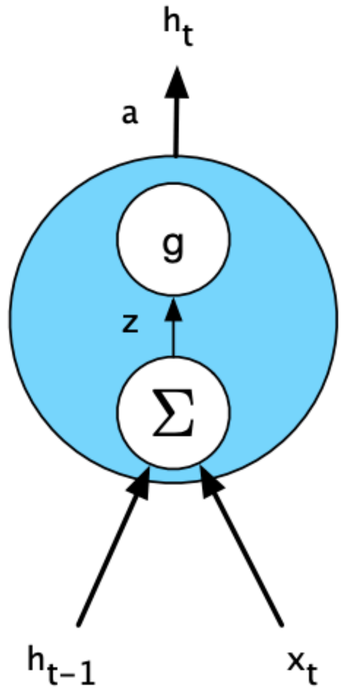
Comparison: FFNNs vs simple RNNs vs LSTMs vs GRUs

Feedforward NNs



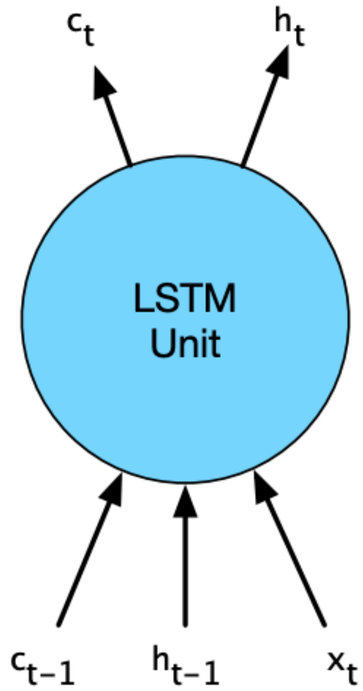
(a)

Simple RNNs



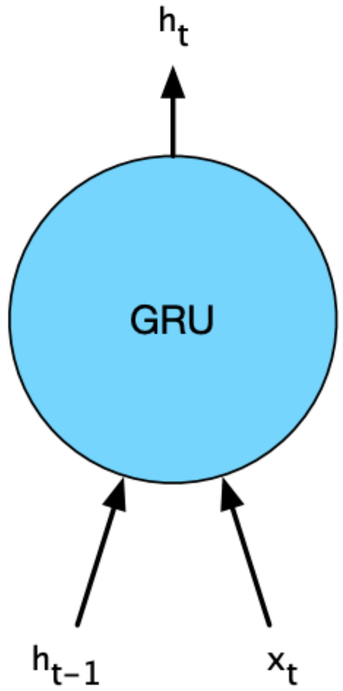
(b)

LSTMs



(c)

GRUs



(d)

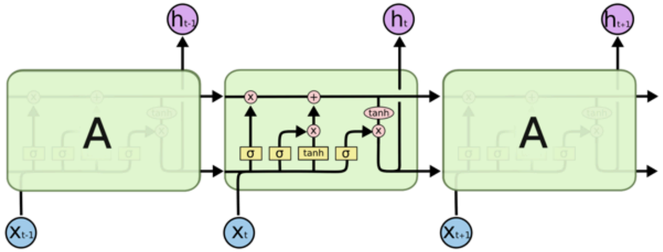
A note on terminology

- Simple RNNs are also called vanilla RNNs 
- Sometimes vanilla RNNs don't work that well, so we need to use some advanced RNN variants such as LSTMs  or GRUs 
- In practice, we always use multi-layer RNNs 

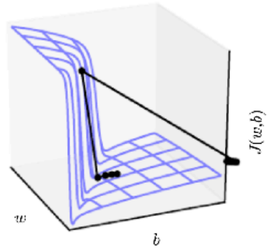
… together with fancy ingredients such as skip-connections with self-attention, variational dropout..

Slide credit: Abigail See (with modifications)

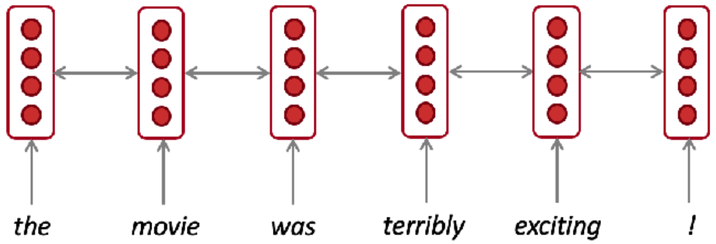
Practical takeaways



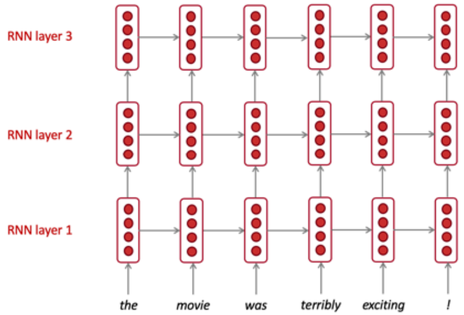
1. LSTMs are powerful



2. Clip your gradients



3. Use bidirectionality when possible



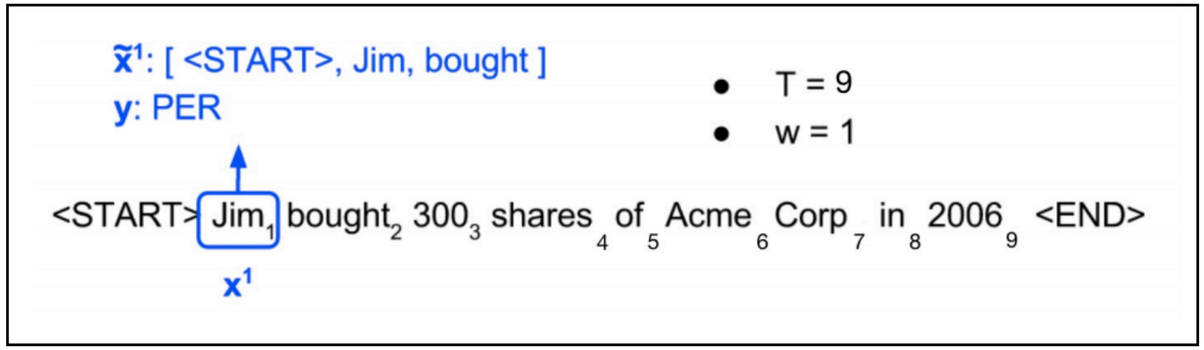
4. Multi-layer RNNs are more powerful, but you might need skip connections if it's deep

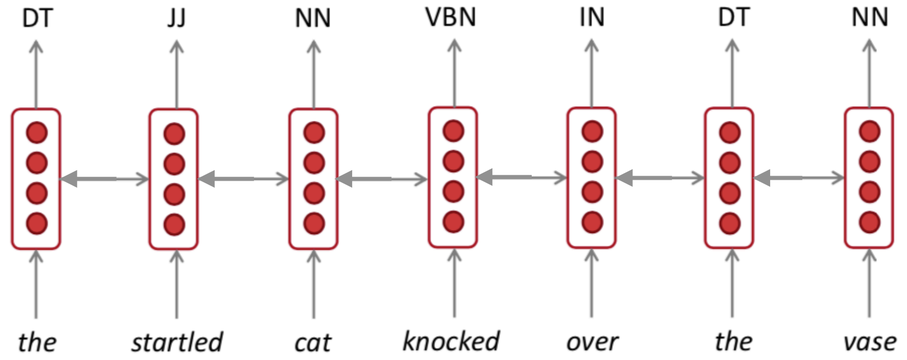
A preview of assignment 3

- You will need to learn how to use PyTorch to train neural networks (on **GPUs**) for the named entity recognition (NER) task
- We will ask you to implement FFNNs and LSTMs.

Ousted WeWork founder Adam Neumann lists his Manhattan penthouse for $37.5 million

[organization] [person] [location] [monetary value]





A preview of assignment 3

Docs > torch.nn > Linear

LINEAR

CLASS `torch.nn.Linear(in_features, out_features, bias=True)` [SOURCE]

Applies a linear transformation to the incoming data: $y = xA^T + b$

This module supports TensorFloat32.

LSTM

CLASS `torch.nn.LSTM(*args, **kwargs)` [SOURCE]

Applies a multi-layer long short-term memory (LSTM) RNN to an input sequence.

For each element in the input sequence, each layer computes the following function:

Parameters

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two LSTMs together to form a *stacked LSTM*, with the second LSTM taking in outputs of the first LSTM and computing the final results. Default: 1
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (batch, seq, feature). Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each LSTM layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional LSTM. Default: `False`
- **proj_size** – If `> 0`, will use LSTM with projections of corresponding size. Default: 0

Make sure that you go to the precept this Friday if you are not familiar with PyTorch!!