CSE 4392 Special Topics
Natural Language Processing

Recurrent Neural Networks

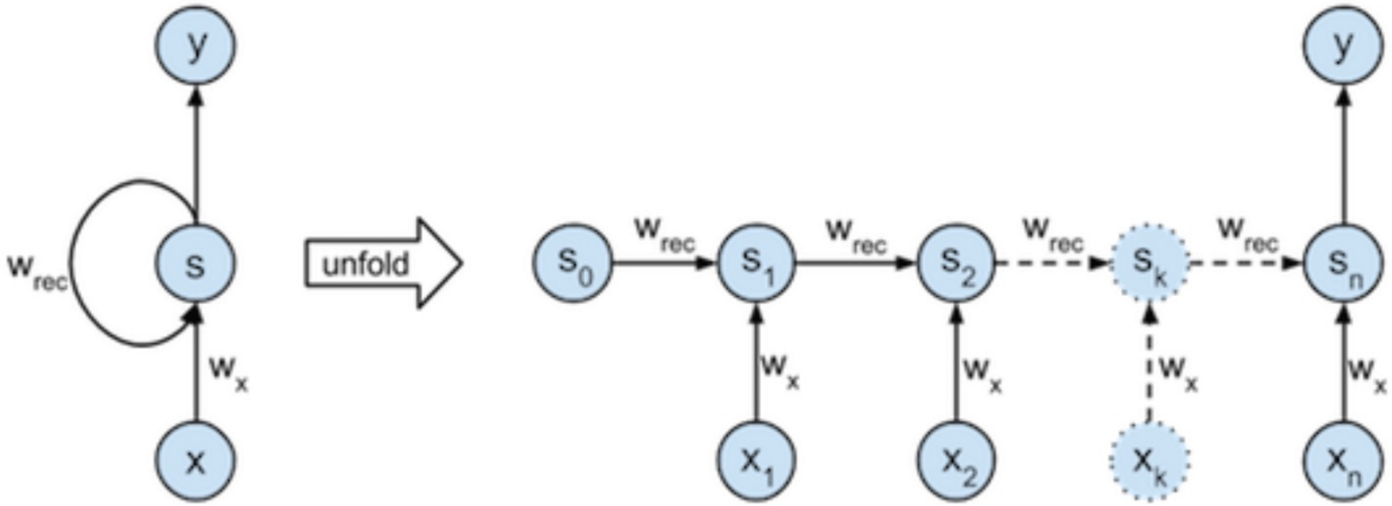
2024 Spring

Overview

- What is a recurrent neural network (RNN)?
- Simple RNNs
- Backpropagation through time
- Long short-term memory networks (LSTMs)
- Applications
- Variants: Stacked RNNs, Bidirectional RNNs

Recurrent Neural Networks (RNNs)

- A class of neural networks designed to handle **variable length inputs**.

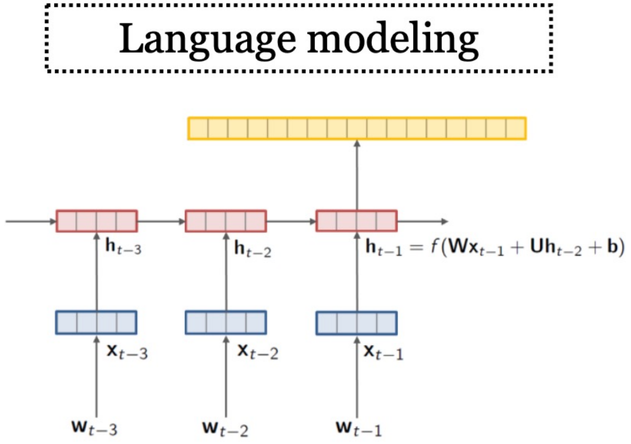


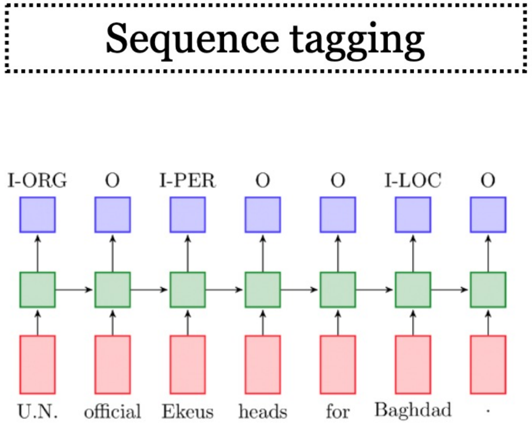
- A function: $y = RNN(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n) \in \mathbb{R}^d$

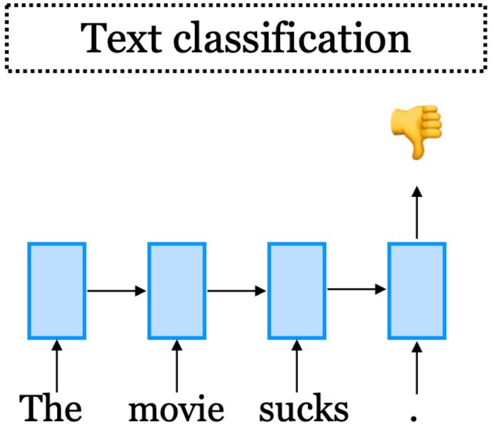
  where $x_i \in \mathbb{R}^{d_{in}}$

Recurrent Neural Networks (RNNs)

- Shown to be a highly effective approach to language model, sequence tagging and classification tasks:

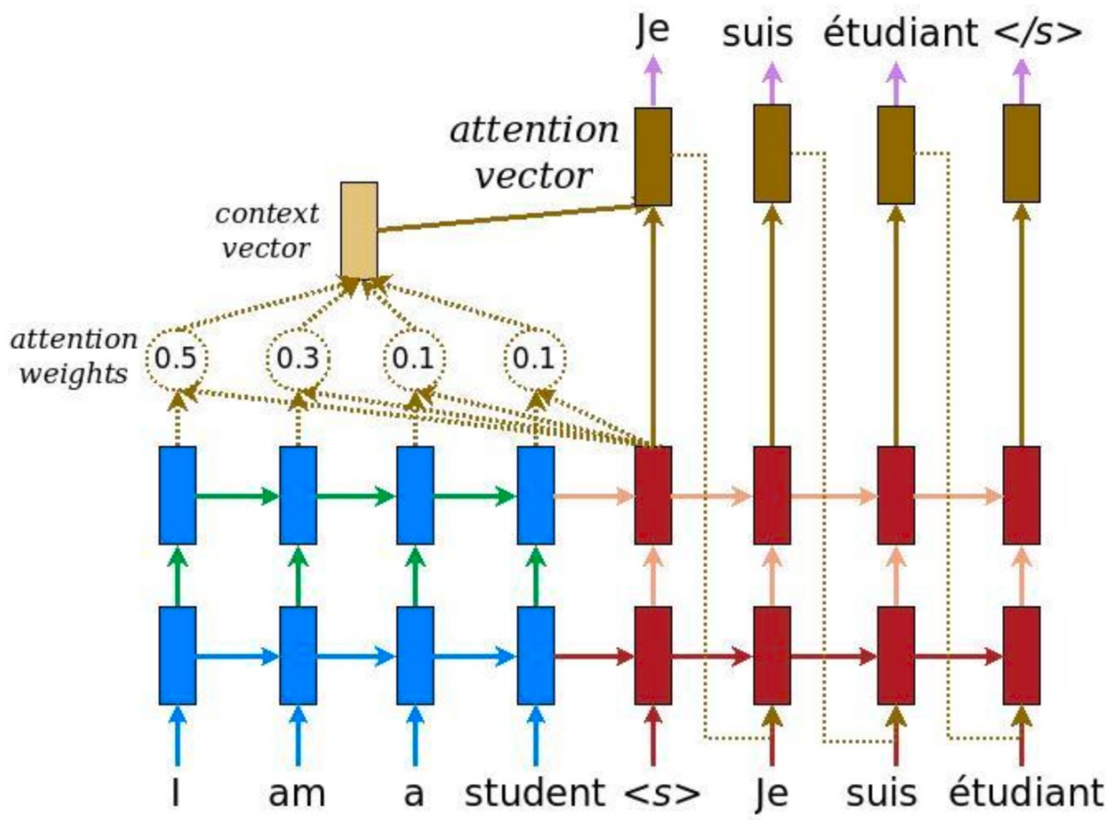






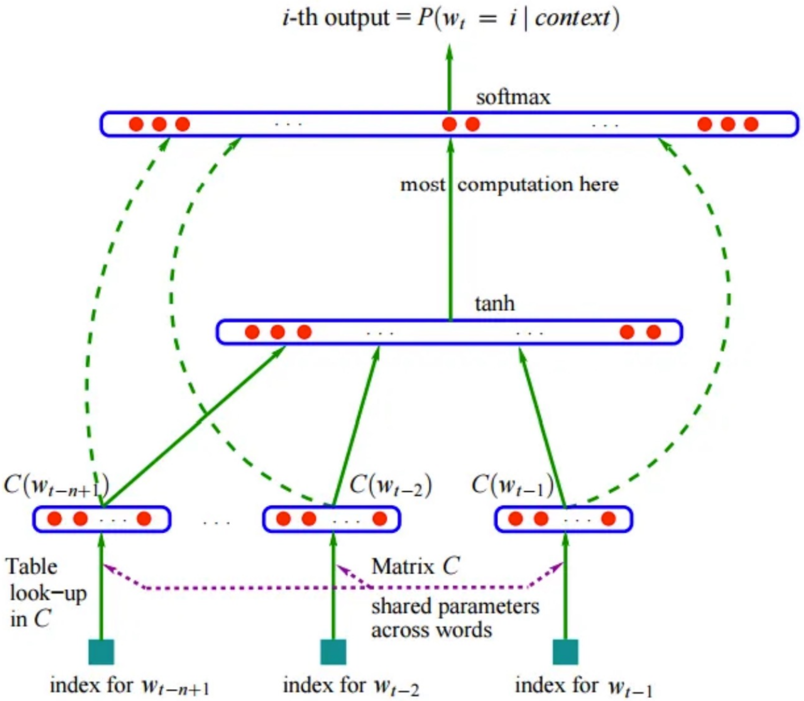
Recurrent Neural Networks

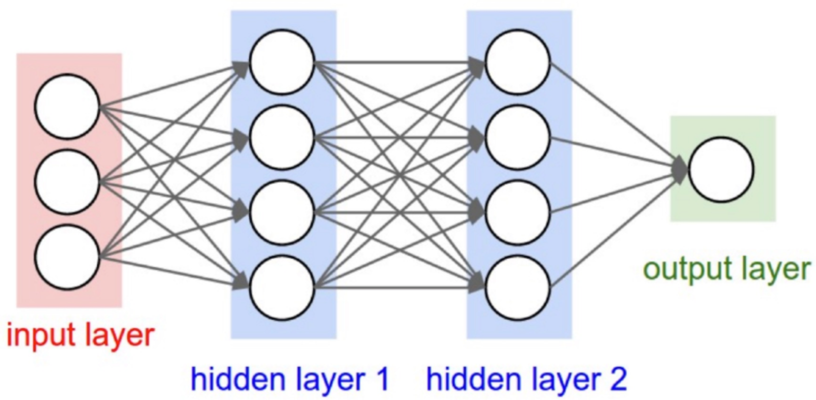
- Form the basis for the modern approaches to machine translation, question answering and dialogue:



Why Variable Length?

- Recall the feed-forward neural LMs we learned:





$$\mathbf{x} = [\mathbf{e}_{\text{the}}, \mathbf{e}_{\text{dogs}}, \mathbf{e}_{\text{are}}] \in \mathbb{R}^{3d}$$

(fixed-window size = 3)

The dogs are barking

the dogs in the neighborhood are ____

Simple RNNs

- $h_0 \in \mathbb{R}^d$ is an initial state

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

- h_t: hidden states which store information from $\boldsymbol{x}_1$ to $\boldsymbol{x}_t$

- **Simple RNNs**:

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

g: nonlinearity (e.g. tanh),

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

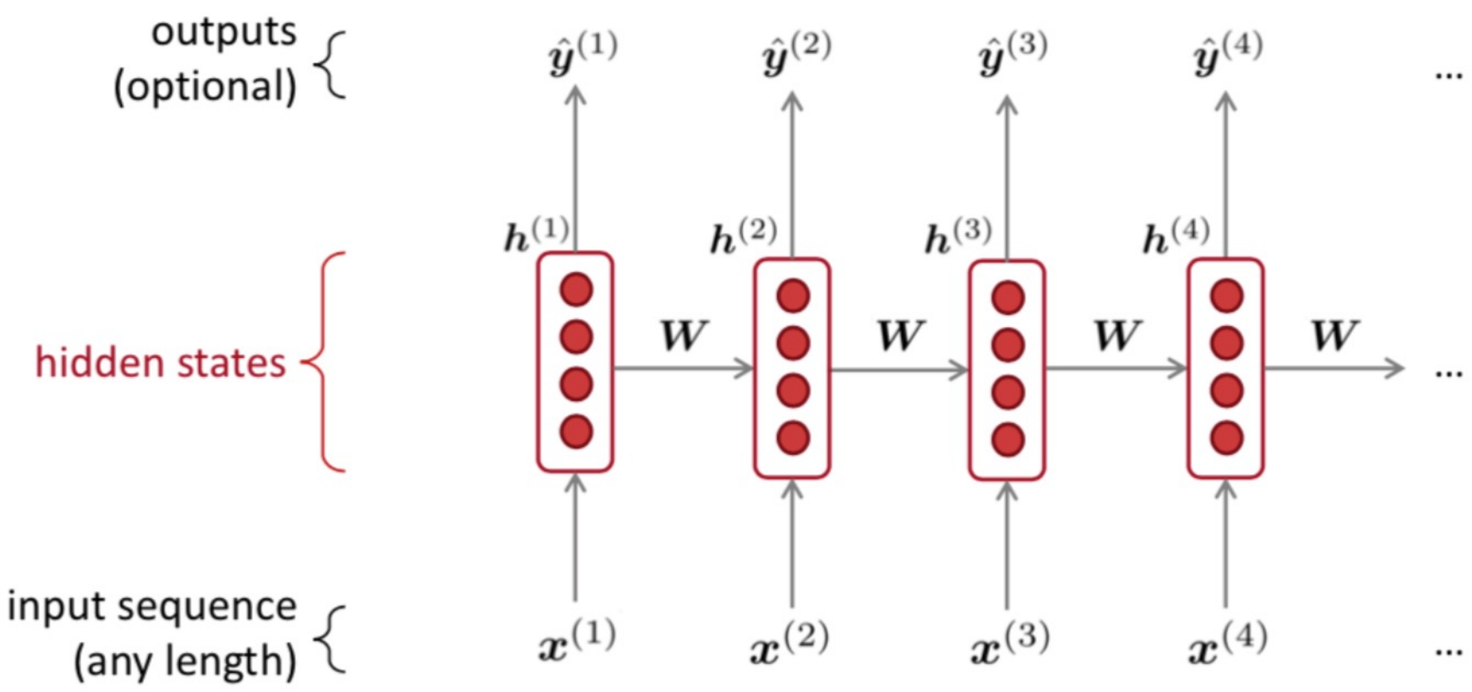
Quiz: Activation Functions

- What's the main difference between sigmoid and tangent hyperbolic (tanh) functions as activation functions?

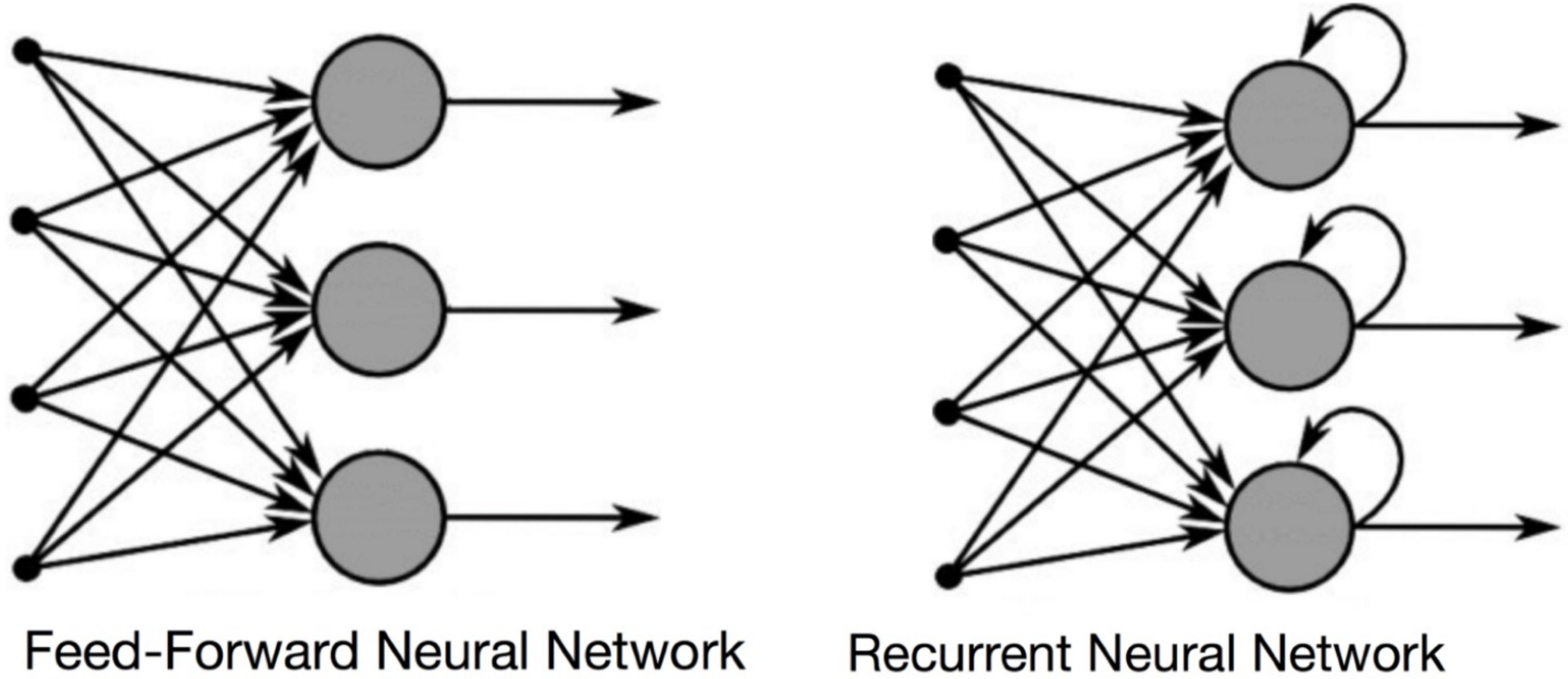
Simple RNNs

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

- Key idea: apply the same weights $\boldsymbol{W}$ repeatedly



RNNs vs. Feedforward NNs



Recurrent Neural Language Models (RNNLMs)

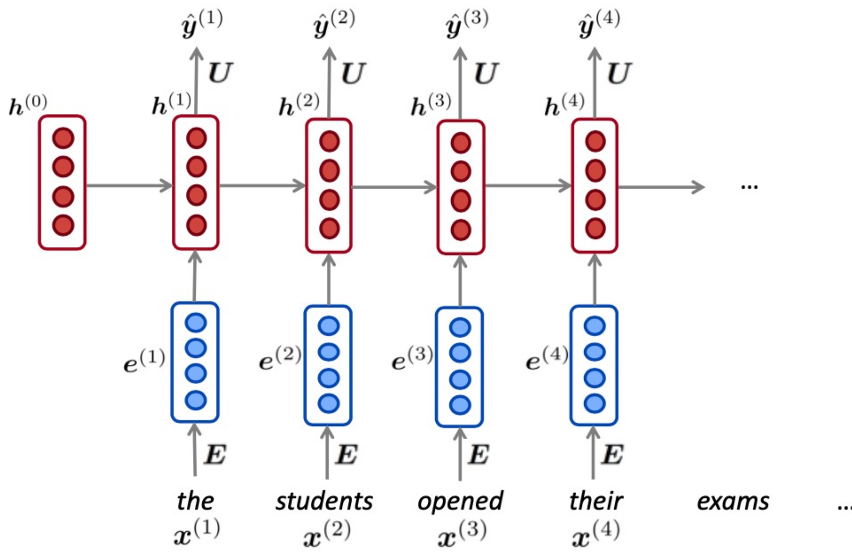
$$P(w_1, w_2, \ldots, w_n) = P(w_1) \times P(w_2 \mid w_1) \times P(w_3 \mid w_1, w_2) \times \ldots \times P(w_n \mid w_1, w_2, \ldots, w_{n-1})$$

$$= P(w_1 \mid \mathbf{h}_0) \times P(w_2 \mid \mathbf{h}_1) \times P(w_3 \mid \mathbf{h}_2) \times \ldots \times P(w_n \mid \mathbf{h}_{n-1})$$

- Denote $\hat{\boldsymbol{y}}_t = softmax(\boldsymbol{W}_o \boldsymbol{h}_t), W_o \in \mathbb{R}^{|V| \times d}$
- Cross-entropy loss:

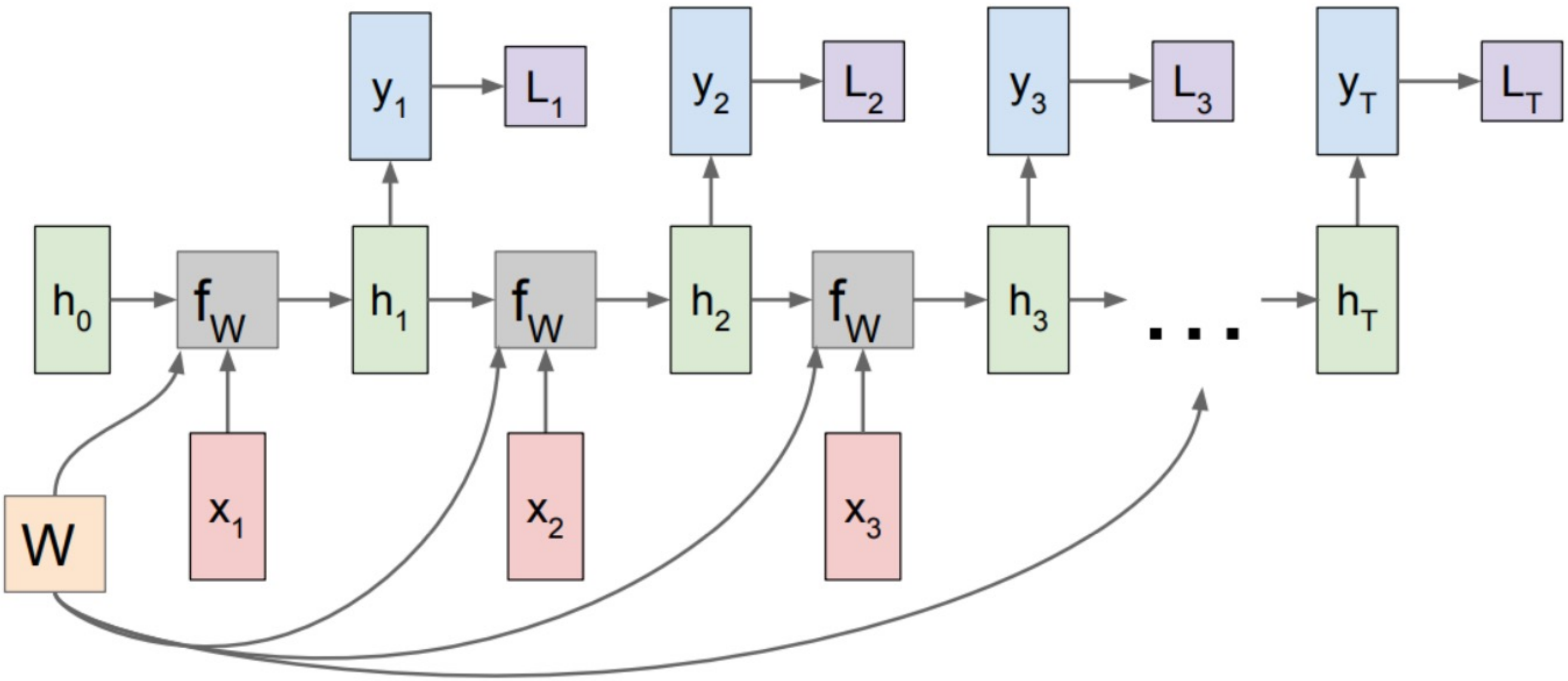
$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]$$

(the negative log probability the model assigns to the next word in the training sequence)



Training RNNLMs

- Back-propagation? Yes, but not so simple!



- The algorithm is called *Backpropagation Through Time* (BPTT)

Backpropagation through time

$$\mathbf{h}_1 = g(\mathbf{W}\mathbf{h}_0 + \mathbf{U}\mathbf{x}_1 + \mathbf{b})$$

$$\mathbf{h}_2 = g(\mathbf{W}\mathbf{h}_1 + \mathbf{U}\mathbf{x}_2 + \mathbf{b})$$

$$\mathbf{h}_3 = g(\mathbf{W}\mathbf{h}_2 + \mathbf{U}\mathbf{x}_3 + \mathbf{b})$$

$$L_3 = -\log \hat{\mathbf{y}}_3(w_4)$$

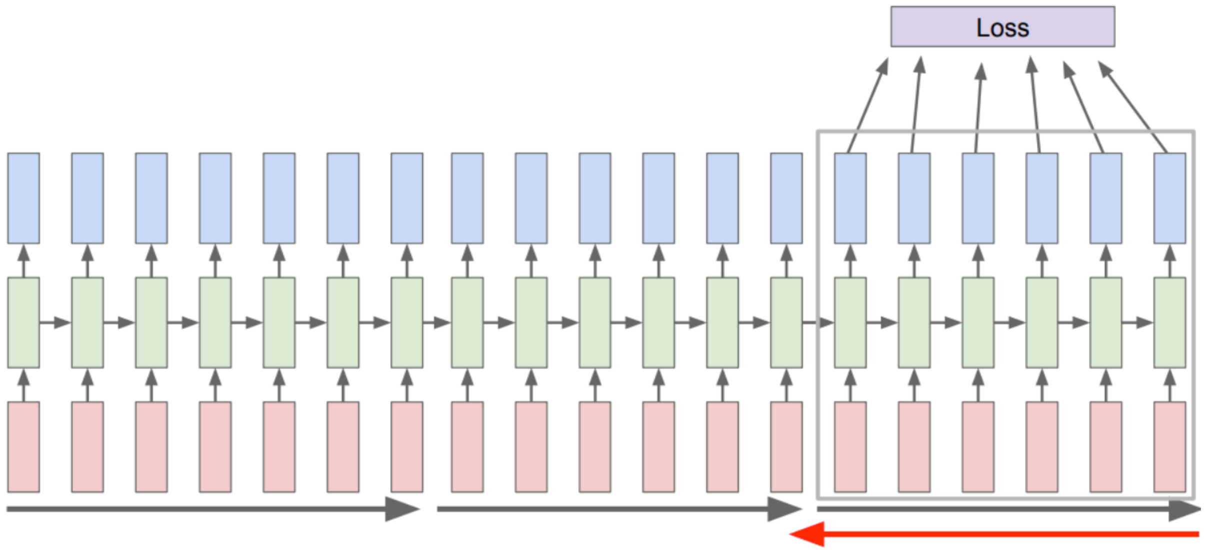
- You should know how to compute: $\dfrac{\partial L_3}{\partial \mathbf{h}_3}$

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

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

Truncated backpropagation though time

- Backpropagation is very expensive if the input sequence is long.

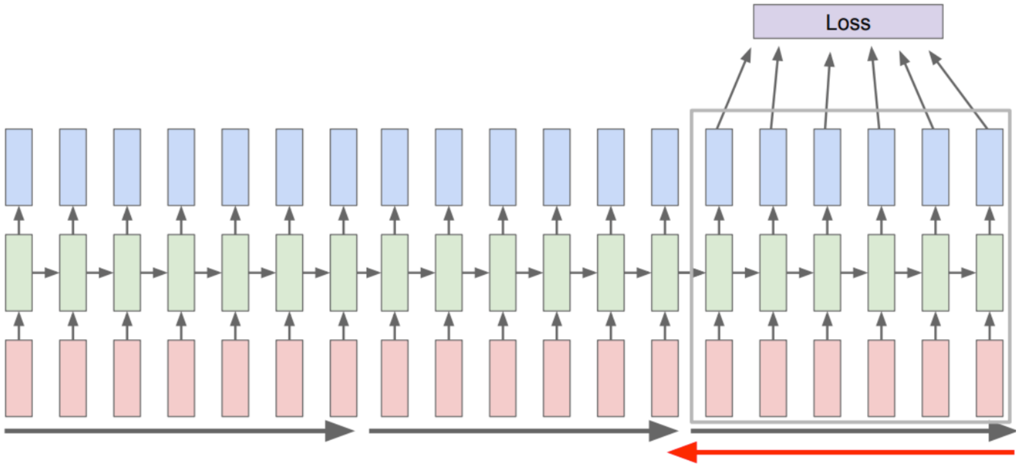


- Run forward and backward through chunks of sequence instead of the whole sequence
- Carry hidden state forward forever, but only backpropagate for some smaller number of steps

Quiz: Backpropagation Though Time

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

- Suppose n is the input length, the backpropagation length is m, please re-derive the above formula for $\frac{\partial L}{\partial \mathbf{W}}$ while backpropagating only m steps.



Progress on Language Models

- On the Penn Treebank (PTB) dataset:
- Metric: Perplexity

KN5: Kneser-Ney 5-gram

Model	Individual
KN5	141.2
KN5 + cache	125.7
Feedforward NNLM	140.2
Log-bilinear NNLM	144.5
Syntactical NNLM	131.3
Recurrent NNLM	124.7
RNN-LDA LM	113.7

(Mikolov and Zweig, 2012): Context dependent recurrent neural network language model

Progress on language models

- On the Penn Treebank (PTB) dataset:
- Metric: Perplexity

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

Vanishing/Exploding Gradients

- Consider the gradient of L_t at step t, with respect to the hidden state $\mathbf{h}_k$ at some previous step k ($k < t$):

$$\frac{\partial L_t}{\partial \mathbf{h}_k} = \frac{\partial L_t}{\partial \mathbf{h}_t}\left(\prod_{t \geq j > k} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}\right)$$

$$= \frac{\partial L_t}{\partial \mathbf{h}_t} \times \prod_{t \geq j > k}\left(diag\left(g'(\mathbf{W}\mathbf{h}_{j-1} + \mathbf{U}\mathbf{x}_j + \mathbf{b})\right)\mathbf{W}\right)$$

- (Pascanu et al, 2013) showed that if the largest eigenvalue of $\mathbf{W}$ is less than 1 for $g = tanh$, then the gradient will shrink exponentially. This problem is called **vanishing gradients**.
- In contrast, if the gradients are getting too large, it is called **exploding gradients**.

Why is exploding gradient a problem?

- When gradients are too large, we take very big steps in SGD, making the algorithm difficult to converge.
- Solution: Gradient clipping – if the norm of the gradient is beyond a threshold, scale it down before applying SGD update.

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

WHY IS VANISHING GRADIENT A PROBLEM?

- If the gradients becomes vanishingly small over *long distances* (step k to step t), then we can't tell whether:
  - We don't need long-term dependencies, or
  - We have wrong parameters to capture the true dependency

    the dogs in the neighborhood are ___

    Still difficult to predict "barking"

- How to fix vanishing gradient problem?
  - **LSTMs: Long short-term memory networks**
  - GRUs: Gated recurrent units

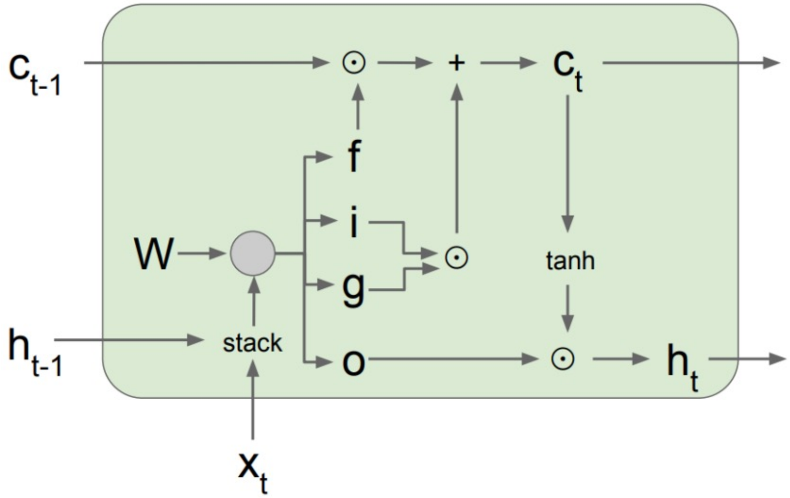
Long Short-term Memory (LSTM)

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the vanishing gradients problem
- Work extremely well in practice
- **Basic idea**: turning multiplication into addition
- Use “gates” to control how much information to add/erase

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

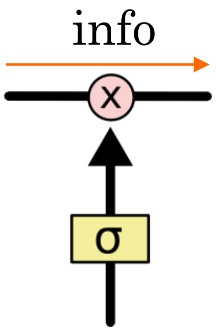
At each timestep, there is a hidden state $\boldsymbol{h}_t \in \mathbb{R}^d$ and also a cell state $\boldsymbol{c}_t \in \mathbb{R}^d$

- $\boldsymbol{c}_t$ stores **long-term information**
- We write/erase $\boldsymbol{c}_t$ after each step
- We read $\boldsymbol{h}_t$ from $\boldsymbol{c}_t$

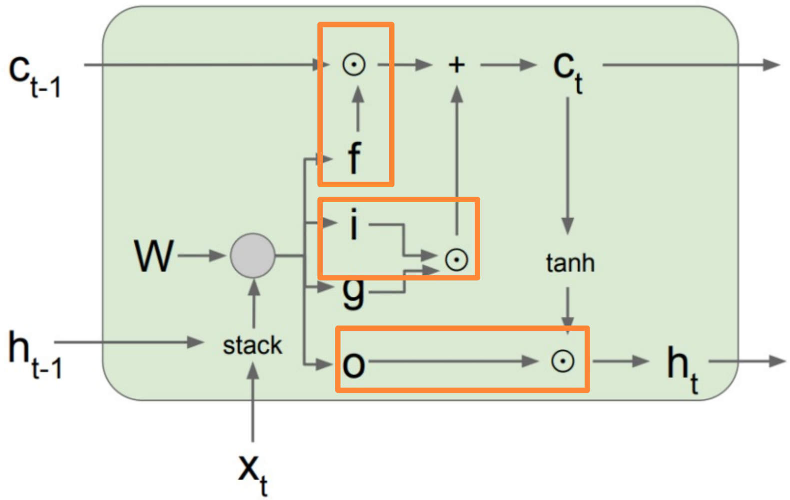


Long Short-term Memory (LSTM)

- There are three gates:
  - each is a *feed- forward layer*, followed by a *sigmoid activation function*, followed by an *element-wise multiplication* with the layer being gated
  - Note we use ⊙ and ⊗ interchangeably to denote element-wise multiplication



a typical gate



LONG SHORT-TERM MEMORY (LSTM)

- forget gate (how much to erase)

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

- input gate (how much to write)

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

- output gate (how much to reveal)

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

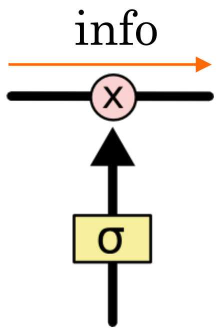
- new cell values

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

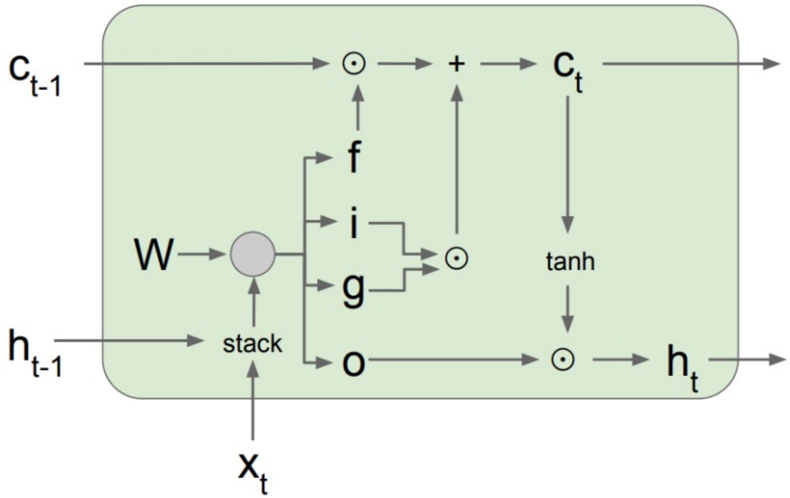
- Final memory cell:

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

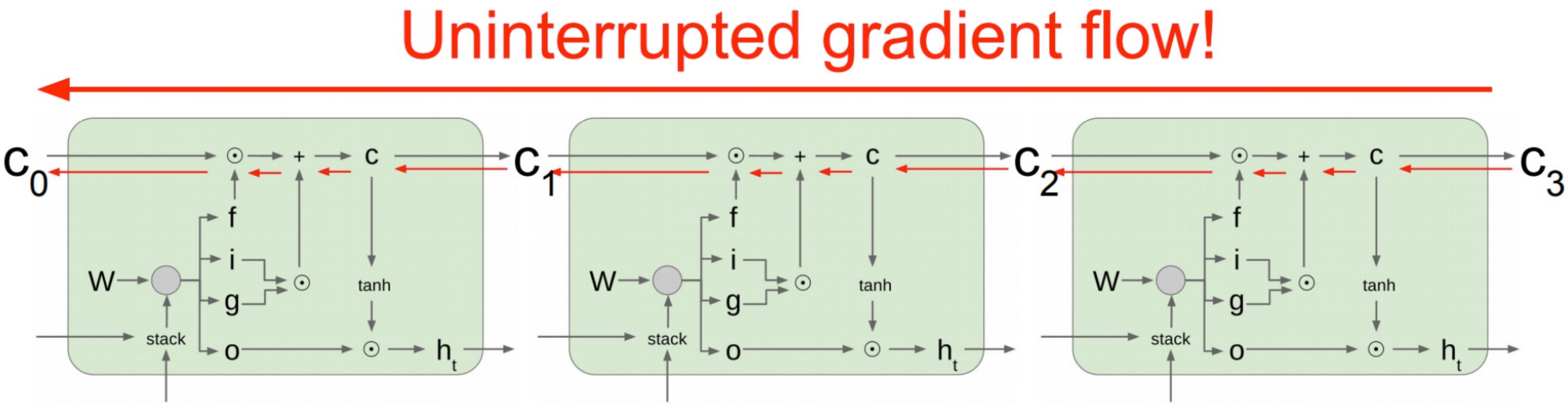
- Final hidden cell: $\mathbf{h}_t = \mathbf{o}_t \odot \mathbf{c}_t$



a typical gate



Long Short-term Memory (LSTM)



- LSTM doesn't guarantee there is no vanishing/exploding gradients.
- It does provide an easier way for models to learn long-distance dependencies.
- LSTM was first invented in 1997, but wasn't working until 2013-2015.

Is LSMT Architecture 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**

Next-step-prediction accuracies

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)

Perplexity on PTB

Reference to LSTM

- Section 9.5 of Jurafsky and Martin
- https://colah.github.io/posts/2015-08-Understanding-LSTMs/