Deep Learning Techniques

DL 2. Recurrent Neural Networks (RNN)

1

Outline

- Standard RNN (not Gated)
- Gated RNN

2

Consider a Vanilla Neural Network

e.g., for hand-writing digit recognition

- **Input layer**

  $x \in \mathbb{R}^M$ is the feature vector of an image.

- **Hidden layer** $h \in \mathbb{R}^D$

  $h = g\big(\underbrace{W_{xh}x}_{v}\big) = \tanh(v) = (g_1 . g_2, \dots, g_D),\ \ W_{xh} \in \mathbb{R}^{D\times M}$

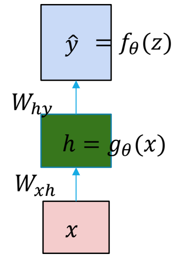
  $g_i \stackrel{\text{def}}{=} \frac{e^{v_i} - e^{-v_i}}{e^{v_i} + e^{-v_i}}$ (rescaled logistic sigmoid)

- **Output layer** $\hat{y} \in \mathbb{R}^K$

  $\hat{y} = f\big(\underbrace{W_{hy}h}_{z}\big) = softmax(z) = (f_1, f_2 \dots, f_K),\ \ W_{hy} \in \mathbb{R}^{D\times K}$

  $f_j = \frac{\exp(z_j)}{\sum_{j'} \exp(z_{j'})}$, estimated probability of j given z

- **Model Parameters** $\boldsymbol{\Theta} = (W_{xh}, W_{hy})$



3

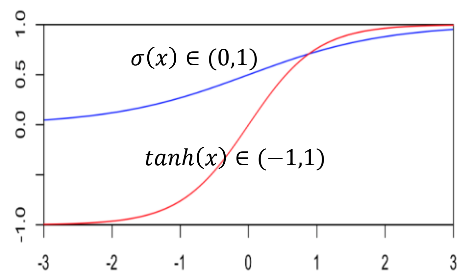
Tanh function is a rescaled sigmoid

$\sigma(x) \stackrel{\text{def}}{=} \frac{e^x}{1+e^x}$ $\qquad for\ x \in (-\infty, \infty),$

$\phi(x) = 2\sigma(2x) - 1$

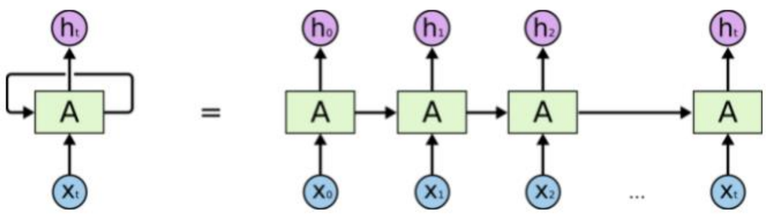
$= 2\frac{e^{2x}}{1+e^{2x}} - 1 = \frac{e^{2x}}{1+e^{2x}} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

$\stackrel{\text{def}}{=} \tanh(\mathrm{x})$



4

RNN for sequential modeling
(https://colah.github.io/posts/2015-08-Understanding-LSTMs/)



An unrolled recurrent neural network.

5

[Adapted from F.F. Li et al. Stanford CS231n]

Recurrent Neural Network (RNN)

- Modeling a sequence of (x_t, h_t, y_t) as

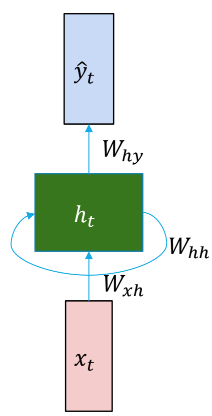
$$\boxed{h_t} := \boxed{g_\theta}(\boxed{h_{t-1},}\boxed{x_t}) = \tanh(\mathrm{v_t}), \quad v_t = W_{hh}h_{t-1} + W_{xh}x_t$$

New state

Old state

Current input

Some function



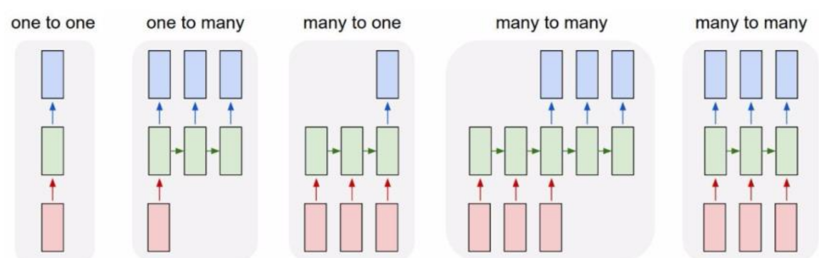
- $\theta = \{W_{xh}, W_{hy}, W_{hh}\}$, the same θ used at each time step

6

[Adapted from F.F. Li et al. Stanford CS231n]

Unrolled RNNs

- Vanila Neural Network (e.g., for image classification)

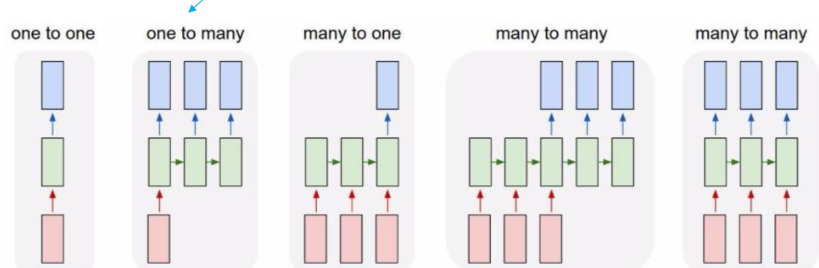


7

[Adapted from F.F. Li et al. Stanford CS231n]

Unrolled RNNs

E.g., Image Captioning: Image → word sequence

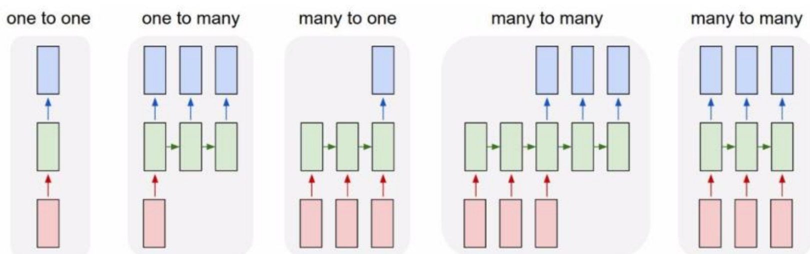


8

[Adapted from F.F. Li et al. Stanford CS231n]

Unrolled RNNs

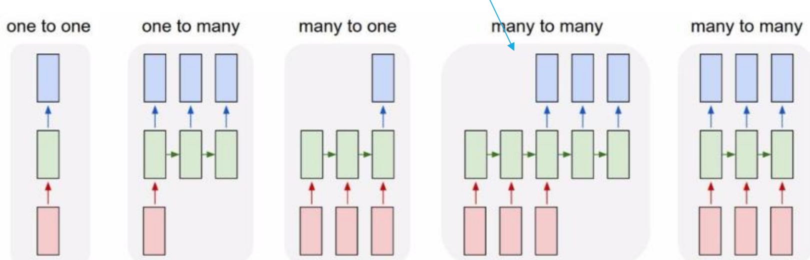
E.g., Text classification:
Word sequence → Topic label



[Adapted from F.F. Li et al. Stanford CS231n]

Unrolled RNNs

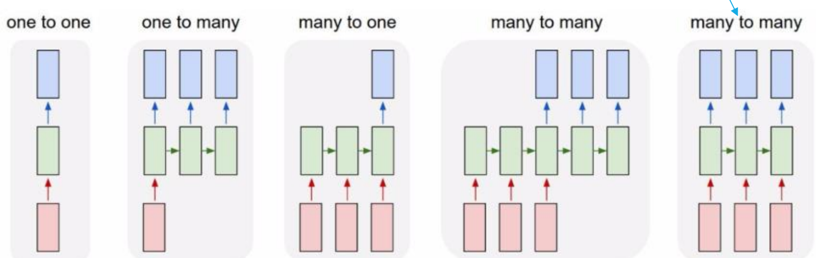
E.g., Machine Translation
Word seq. → Word seq.



[Adapted from F.F. Li et al. Stanford CS231n]

Unrolled RNNs

E.g., POS tagging
Word seq. → Tag seq.



11

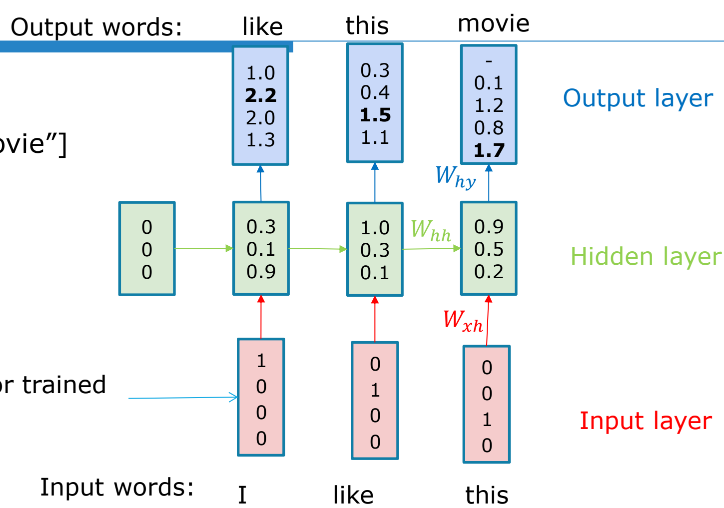
Example: Language Modeling

[Adapted from F.F. Li et al. Stanford CS23

Vocabulary:
["I", "like", "this", "movie"]

Training sequence:
"I like this movie"

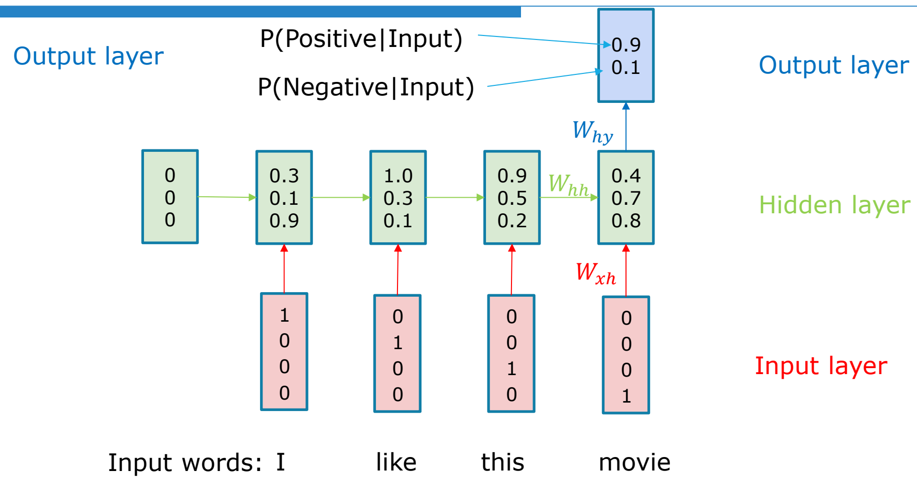
It can also be a dense vector trained by word2vec or GloVe



12

[Adapted from F.F. Li et al. Stanford CS231n]

Example: Sentiment Classification

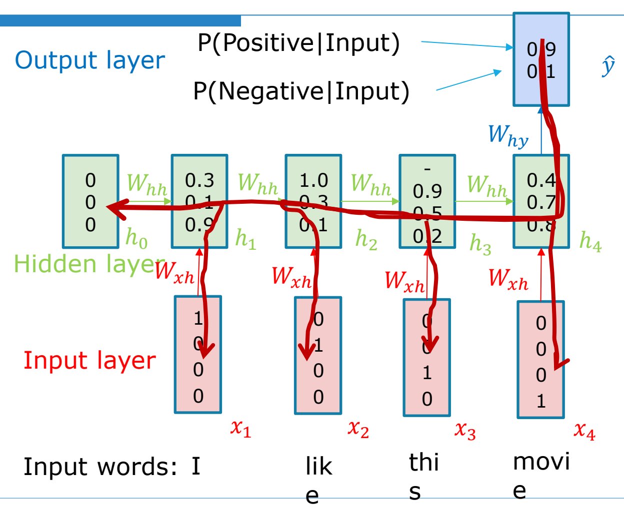


13

[Adapted from F.F. Li et al. Stanford CS231n]

Gradient vanishing problem in training RNN

- Multiplying the same matrices multiple times during forward/backwards propagation
- Causing the gradient to become very small or very large quickly, which is called the gradient **vanishing or exploding problem**



14

Why do gradients vanish (or explode) in neural nets?

- "Neural Networks and Deep Learning", Chapter 5, by Michael Nielsen (Dec 2017), http://neuralnetworksanddeeplearning.com/chap5.html
- Example: a multi-layer nnet with a sigmoid function at each layer
- Showing how the gradient of the output variable w.r.t. an input-layer variable would vanish or explode when the number of layers increases
- More generally, "neural networks suffer from an unstable gradient problem."

[based on the RNN tutorial by WILDML, 2015]

Why do gradients vanish in RNN?

- We could (incorrectly) compute the gradient of loss function $L(y,\hat{y})$ as

$$\frac{\partial L}{\partial W_{hh}} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial h_4}\frac{\partial h_4}{\partial W_{hh}}$$

- But h_4 also depends on h_3, h_2, h_1, and each h_i depends on W_{hh}.

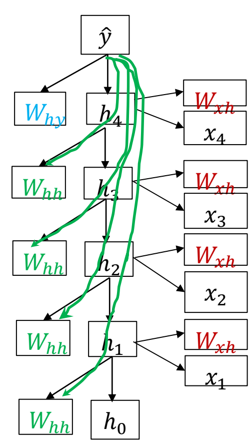
  So, the correct formula should be

$$\frac{\partial L}{\partial W_{hh}} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial h_4}\frac{\partial h_4}{\partial W_{hh}} + \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial W_{hh}}$$

$$+ \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial W_{hh}} + \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial W_{hh}}$$

$$= \sum_{i=1}^{n} \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial h_n}\left(\prod_{j=i+1}^{n}\frac{\partial h_j}{\partial h_{j-1}}\right)\frac{\partial h_i}{\partial W_{hh}} \quad \text{(here } n = 4\text{)}$$

- When n is large (e.g., 500), we have a long chain of $\prod_j \frac{\partial h_j}{\partial h_{j-1}}$.



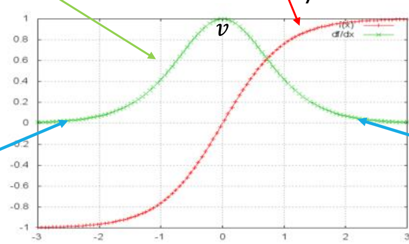
[based on the RNN tutorial by WILDML, 2015]

Why do gradients vanish in RNN (cont'd)

$$\prod_j \boxed{\frac{\partial h_j}{\partial h_{j-1}}} = \prod_j \boxed{\frac{\partial \tanh(v_j)}{\partial v_j}} \frac{\partial v_j}{\partial h_{j-1}} \qquad \boxed{h_t = \tanh(v_t)} \qquad v_t = W_{hh}h_{t-1} + W_{xh}x_t$$

Jacobin Matrix $\in \mathbb{R}^{d \times d}$

Gradient bounded by 1



Vanishing gradient

Vanishing gradient

$\frac{\partial \tanh(v)}{\partial v}$

$\tanh(v)$

Notice that $\frac{\partial \tanh(v_j)}{\partial v_j}$ is between 0 and 1 and could be very near zero -- multiplying it several times would result in the vanishing gradient.

Outline

- Optimizing Neural Network
  - Stochastic Gradient Descent (Recap)
- Recurrent Neural Network (RNN)
  - Vanilla RNN
  - Gated RNN

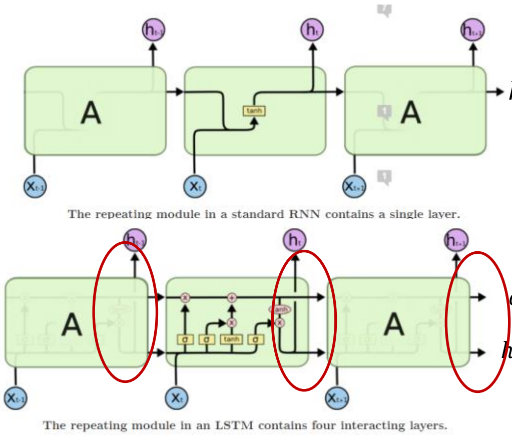
Outline

- ✓ Standard RNN (not Gated)
- Gated RNN

Gated Recurrent Neural Networks

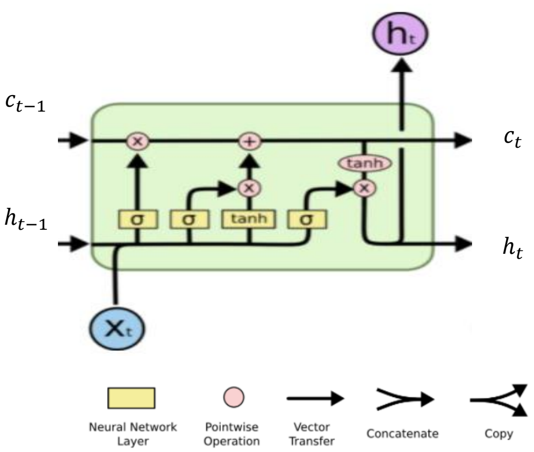
- **LSTM** (Long Short Term Memory) as a representative model
- Introducing gates to plain RNN, to shorten the information flow and to avoid non-linear operations (like tanh) as needed
- Meditating the gradient vanishing issue in RNN effectively

Plain RNN vs. LSTM



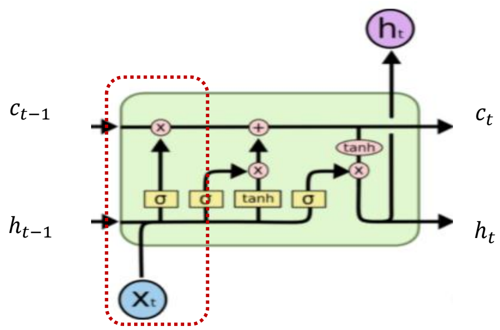
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left(\underbrace{(W_{hh} \quad W_{xh})}_{W_o}\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM Module



- Cell state c (a vector) allows the information to flow through a linear path (instead of non-linear activations) in many steps, and hence is less prone to vanishing gradient.
- Gates ($\sigma's$) allow the system to skip some steps of the process, or to block the long chain of steps.

LSTM Module



Forget gate $f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f)$

Input gate $i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i)$

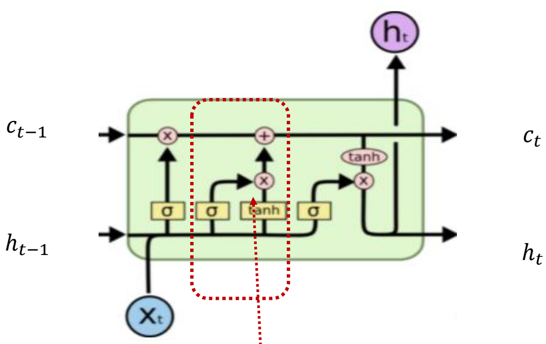
Output gate $o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o)$

The behavior of the gates are controlled by model parameters of $W's$ and $b's$.

- $f_t = \sigma(.) \in (0,1)$ mimic a soft gate.
- $f_t \to 0$ means the gate is closed, forcing c_{t-1} (old memory) to be forgotten.
- $f_t \to 1$ means the gate is open, allowing c_{t-1} (old memory) to be kept.

23

LSTM Module



Input Gate $i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i)$

- Linear path

  $$c_t = f_t * c_{t-1} + i_t * g_t$$

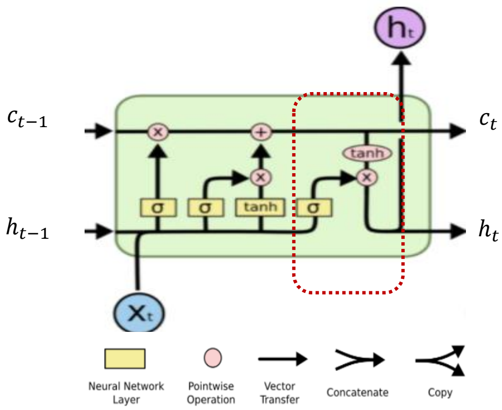
- Non-linear function

  $$g_t = \tanh(W_c * [h_{t-1}, x_t] + b_c)$$

- If i_t is closed, then input x_t is skipped in the updating of c_t .
- If i_t is open but f_t is closed, then we forget c_{t-1} and renew c_t with g_t

24

LSTM Module

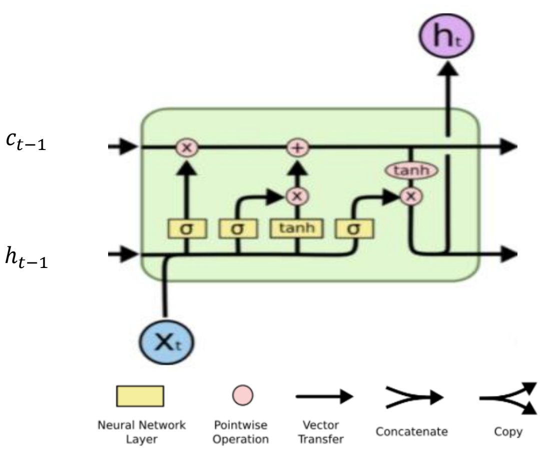


Output gate $o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o)$

$$h_t = o_t * tanh(c_t)$$

- If o_t is closed, then we block the hidden state h_t from going forward.
- Otherwise, hidden state $h_t = tanh(c_t)$ is propaged forward.

LSTM Module



- Who control the gates?
  - The model parameters
    $W_f, W_i,, W_o, W_c$
  - Not human designers of LSTM!

Performance on language modeling

[Yoon Kim et al. AAAI 2016]

	PPL	Size
LSTM-Word-Small	97.6	5 m
LSTM-Char-Small	92.3	5 m
LSTM-Word-Large	85.4	20 m
LSTM-Char-Large	78.9	19 m
KN-5 (Mikolov et al. 2012)	141.2	2 m
RNN† (Mikolov et al. 2012)	124.7	6 m
RNN-LDA† (Mikolov et al. 2012)	113.7	7 m
genCNN† (Wang et al. 2015)	116.4	8 m
FOFE-FNNLM† (Zhang et al. 2015)	108.0	6 m
Deep RNN (Pascanu et al. 2013)	107.5	6 m
Sum-Prod Net† (Cheng et al. 2014)	100.0	5 m
LSTM-1† (Zaremba et al. 2014)	82.7	20 m
LSTM-2† (Zaremba et al. 2014)	78.4	52 m

5-gram LM (not neural network)

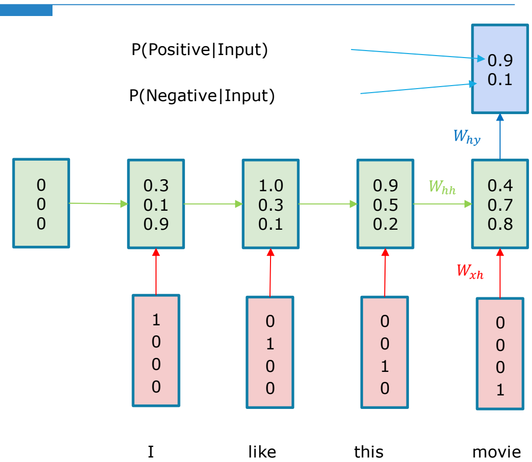
Plain RNN for LM

Word-level LSTM

Table 3: Performance of our model versus other neural language models on the English Penn Treebank test set. *PPL* refers to perplexity (lower is better) and size refers to the approximate number of parameters in the model. KN-5 is a Kneser-Ney 5-gram language model which serves as a non-neural baseline. †For these models the authors did not explicitly state the number of parameters, and hence sizes shown here are estimates based on our understanding of their papers or private correspondence with the respective authors.

Concluding Remarks

- RNN produces dynamic word embeddings at the inference time in its hidden layer (overcoming a fundamental limitation of CBOW, GloVe, etc.)
- Word embeddings by RNN are sensitive to the prediction tasks.
- RNN uses the shared model parameters (W_{xh}, W_{hh}, W_{hy}) in each of its steps.
- Gradient vanishing is a main issue with RNN when the sequence length is large (e.g., with hundreds or thousands tokens).



Reference

- Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.
- Louis-Philippe Morency, Tadas Baltrusaitis, CMU 11-777: Advanced Multimodal Machine Learning
- Fei-Fei Li, Andrej Karpathy, Justin Johnson Stanford CS231n: Convolutional Neural Networks for Visual Recognition
- RNN tutorial by WILDML http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/
- Christopher Olah's blog: Understanding LSTM Networks
- Denny Britz: Recurrent Neural Networks tutorial