

Sequence-to-sequence models: Part I

Recurrent Neural Networks, Long short-term memories (LSTMs)

Ryan Chan

17 April 2023

Outline

Language modelling

Window-based language models

- n-gram models

- Fixed-window / feedforward neural language models

Recurrent Neural Networks

- Training RNNs

Long short-term memories (LSTMs)

RNN variants: stacked & bidirectional

Language modelling

- **Language models** provide a probability distribution of sequences of *tokens* in a sentence
- Given a *vocabulary* $V = \{w_1, \dots, w_{|V|}\}$ of words, a language model assigns a probability to a sequence of tokens, y_1, \dots, y_t : $p(y_1, \dots, y_t)$
- Common way to write this in terms of conditional probability

$$p(y_1, \dots, y_t) = p(y_1) \cdot p(y_2|y_1) \cdot p(y_3|y_2, y_1) \cdots p(y_t|y_{1:t-1}) \quad (1)$$

$$= \prod_{i=1}^t p(y_i|y_{1:i-1}) \quad (2)$$

- Text generation algorithm:
 - For $i = 1, \dots, t$, sample next word $y_i \sim p(y_i|y_{1:i-1})$
 - Note: Some people introduce a *temperature* parameter T to control the randomness of the language model and sample $y_i \sim p(y_i|y_{1:i-1})^{\frac{1}{T}}$
- **Question:** how do we compute $p(y_i|y_{1:i-1})$?

n -gram models

- Prior to deep learning, n -gram language models [Brown et al., 1992; Brants et al., 2007] were dominant
- **Markov assumption**: prediction of a token y_t *only* depends on the preceding $(n - 1)$ tokens, $y_{t-(n-1):t-1}$, rather than the full history, i.e. we set

$$p(y_t | y_{1:t-1}) = p(y_t | y_{t-(n-1)}, \dots, y_{t-1}), \text{ (by assumption)} \quad (3)$$

$$= \frac{p(y_{t-(n-1)}, \dots, y_{t-1}, y_t)}{p(y_{t-(n-1)}, \dots, y_{t-1})}, \text{ (by conditional probability)} \quad (4)$$

- To compute these n -gram and $n - 1$ -gram probabilities, we **count**:

$$p(y_t | y_{1:t-1}) \approx \frac{\text{count}(y_{t-(n-1)}, \dots, y_{t-1}, y_t)}{\text{count}(y_{t-(n-1)}, \dots, y_{t-1})} \quad (5)$$

- These models are **fixed window** models of size n : considers the last $(n - 1)$ tokens to predict the next

Problems with n -gram models

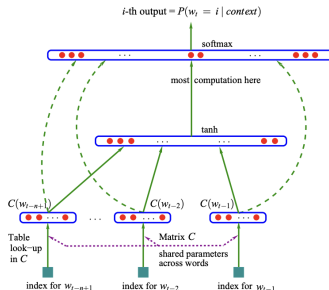
- **Sparsity:** If n is too large, it's statistically infeasible to get good estimates of the probabilities

$$p(\text{Ryan} | \text{the bug was created by}) = \frac{\text{count}(\text{the bug was created by Ryan})}{\text{count}(\text{the bug was created by})} \quad (6)$$

- Problem 1: if "the bug was created by Ryan" never occurred, then this is given probability 0 \rightarrow solved by *smoothing* (adding small δ to each word count)
- Problem 2: if "the bug was created by" never occurred, we have division by 0 \rightarrow solved by *backoff* (condition on $n - 2, \dots$ words)
- Increasing n makes sparsity worse: typically can only go to $n = 5$
- **Storage:** you need to store the counts for all n -grams you saw in the corpus (and the $n - 1$ grams)
 - Increasing n or the corpus increases model size

Neural language models

- Recall the language model by Bengio et al. [2003]:



- Model could also learn *distributed representation of words*, i.e. can produce **word embeddings**
- Context width still bound by n , but statistically feasible to estimate neural language models for much larger values of n
- Main challenge: training neural networks were more computationally expensive
 - Bengio et al. [2003] trained network on 14 million (14×10^6) words
 - Brants et al. [2007] trained 5-gram model on 2 trillion tokens (2×10^{12})

Fixed-window neural language model

- Slightly simplified version of Bengio et al. [2003]:

output distribution

$$\hat{y} = \text{softmax}(U\mathbf{h} + \mathbf{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

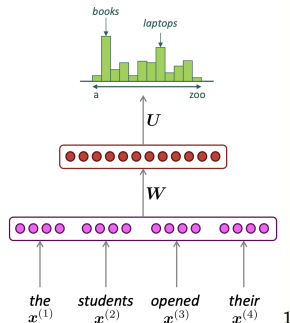
$$\mathbf{h} = f(W\mathbf{e} + \mathbf{b}_1)$$

concatenated word embeddings

$$\mathbf{e} = [e^{(1)}; e^{(2)}; e^{(3)}; e^{(4)}]$$

words / one-hot vectors

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \mathbf{x}^{(4)}$$

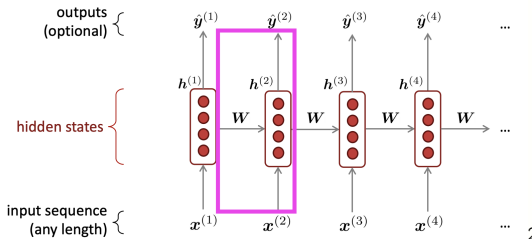


- No more sparsity problems
- Don't need to store all observed n -grams
- But:** fixed window is too limiting - enlarging window enlarges W

¹Manning et al. [2017]

Recurrent Neural Networks

- RNNs [Hopfield, 1982; Rumelhart et al., 1985] are capable of conditioning the model on *all* previous tokens (in theory)



- Core idea: the hidden layer from previous timestep provides a form of **memory** or **context** that informs decisions to be made later in the sequence
 - In theory, context embodied in previous hidden layers can provide information extending to the beginning of the sequence
- The **same** weights are applied at every timestep
- The model size does not increase for longer input sequence lengths

Recurrent Neural Networks for language modelling

- Instead of a fixed-window approach, sequences are processed by presenting one item/token at a time to the network: (image taken from Stanford CS244n slides)

output distribution

$$\hat{y}^{(t)} = \text{softmax}(U h^{(t)} + b_2) \in \mathbb{R}^{|V|}$$

hidden states

$$h^{(t)} = \sigma(W_h h^{(t-1)} + W_e e^{(t)} + b_1)$$

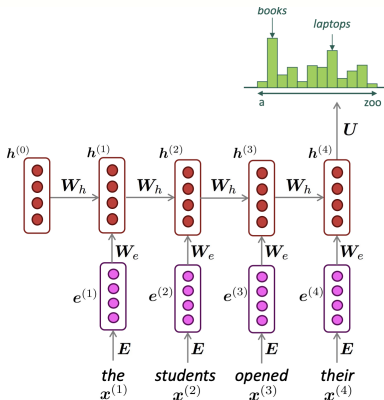
$h^{(0)}$ is the initial hidden state

word embeddings

$$e^{(t)} = E x^{(t)}$$

words / one-hot vectors

$$x^{(t)} \in \mathbb{R}^{|V|}$$



3

Training RNN language models: Cross-Entropy loss

- Given a large corpus of text with a sequence of words $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(T)}$, we feed this into a RNN-LM and compute an **output distribution** $\hat{\mathbf{y}}^{(t)}$ for every time step t
 - $\hat{\mathbf{y}}^{(t)}$ is a probability distribution for every word, conditional on all the words prior to t
 - We know the next word: $\mathbf{y}^{(t)}$ is a one-hot encoding of $\mathbf{x}^{(t+1)}$
- Use the **cross-entropy loss** between $\hat{\mathbf{y}}^{(t)}$ and the *true* word $\mathbf{y}^{(t)}$

$$J^{(t)}(\theta) = \text{CE}(\mathbf{y}^{(t)}, \hat{\mathbf{y}}^{(t)}) = - \sum_{w \in V} \mathbf{y}_w^{(t)} \log \hat{\mathbf{y}}_w^{(t)} = - \log \hat{\mathbf{y}}_{\mathbf{x}^{(t+1)}}^{(t)} \quad (7)$$

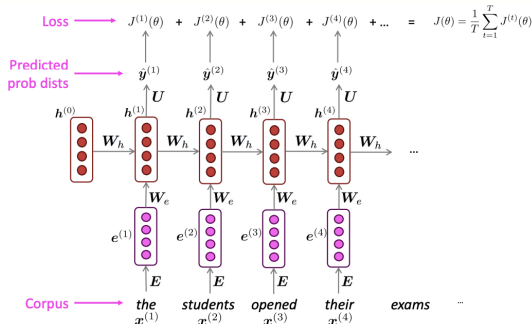
- Average** over the loss for the entire corpus:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^T - \log \hat{\mathbf{y}}_{\mathbf{x}^{(t+1)}}^{(t)} \quad (8)$$

- Typically computing loss and gradients for the whole corpus is too expensive
- In practice, we split the corpus into sentences or documents and use them as *batches* in stochastic gradient descent

Training RNN language models: teacher-forcing

- **Teacher-forcing**: when making a prediction of the word at time t , $\mathbf{y}^{(t)}$, we use the previous words as input to the RNN-LM
 - Rather than using what the model has predicted so far



4

- But when **generating** text with RNN-LMs, the sampled output at time t , $\mathbf{y}^{(t)}$ does become the next step's input

Evaluating Language Model performance

- Standard evaluation metric for language models is **perplexity**

$$\text{perplexity} = \exp(J(\theta)) \quad (9)$$

- Is a measure of confusion (or uncertainty) in the model
- *Lower* perplexity is considered better and implies more confidence in predicting the next word in the sequence (compared to the ground truth)

Problems with RNNs

- Computation is **slow**
 - The model is inherently sequential and cannot be parallelised easily
- Difficulty in **training**
 - In practice, it is difficult to access information from many steps back due to **vanishing** or **exploding** gradients
 - During backpropagation, hidden layers are subject to repeated multiplications (determined by the length of the sequence)
 - Gradients often get gradually drive to zero (vanishing gradients)
 - The hidden layers/states and weights are asked to do a *lot* of work:
 - Provide information useful for the current decision
 - Update and carry forward information required for future decisions

LSTMs

- Long short-term memory (LSTM) networks [Hochreiter and Schmidhuber, 1997; Gers et al., 2000] are the most commonly used extension to RNNs
- Core idea: divide context management into two problems:
 1. Remove information that is no longer needed from the context
 2. Add information that is likely to be needed later on in decision making
- LSTMs try to *learn* how to manage context by adding in several **specialised neural units**
 - They add an explicit **context** layer along with the usual recurrent hidden layer
 - They add additional **gates** that can add and remove information to the context
- Gated Recurrent Units (GRUs) [Cho et al., 2014] are simpler alternatives to LSTMs which has fewer parameters and are a bit faster than LSTMs

LSTMs: a summary

- Essentially replaces “vanilla” RNN blocks with LSTM blocks that are able to preserve information over longer timesteps
 - In practice, could preserve information to about 100 timesteps rather than 7 in “vanilla” RNNs
- At step t , there is a **hidden state vector** $\mathbf{h}^{(t)}$ and a **context state vector** $\mathbf{c}^{(t)}$ which together store **long-term information**
- LSTM network can read, erase and write information from the cell
 - The selection of what gets read/erased/written is controlled by **gates**
- We'll go through the architecture in the next few slides:
 - Let \circ be the **Hadamard product**, i.e. the element-wise multiplication operation
 - See "**Understanding LSTM Networks**" [Olah, 2015] for a bit more of a deeper explanation and walkthrough

LSTMs: architecture and gated activation functions I

To compute the current context state $\mathbf{c}^{(t)}$:

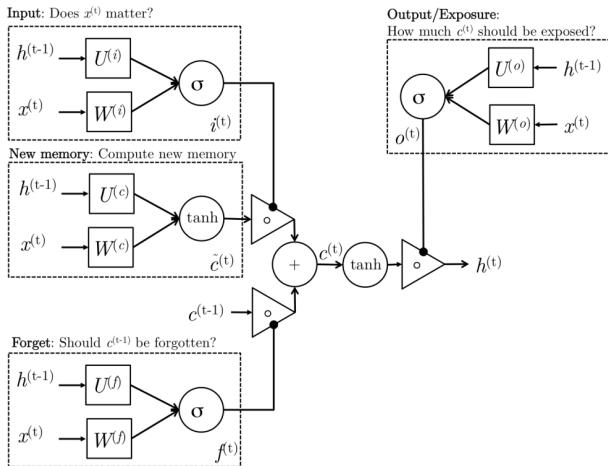
- **New memory generation:** $\tilde{\mathbf{c}}^{(t)} = \tanh(W^{(c)}\mathbf{x}^{(t)} + U^{(c)}\mathbf{h}^{(t-1)} + \mathbf{b}_c)$
 - Use input $\mathbf{x}^{(t)}$ and past hidden state $\mathbf{h}^{(t-1)}$ to generate new memory $\tilde{\mathbf{c}}^{(t)}$ which includes information about the new word
- **Input gate:** $\mathbf{i}^{(t)} = \sigma(W^{(i)}\mathbf{x}^{(t)} + U^{(i)}\mathbf{h}^{(t-1)} + \mathbf{b}_i)$
- **Forget gate:** $\mathbf{f}^{(t)} = \sigma(W^{(f)}\mathbf{x}^{(t)} + U^{(f)}\mathbf{h}^{(t-1)} + \mathbf{b}_f)$
- **Final memory generation / context state:** $\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \circ \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \circ \tilde{\mathbf{c}}^{(t)}$
 - $\mathbf{f}^{(t)}$ assess if the past memory is useful for the current memory cell - *gates* the past context state $\mathbf{c}^{(t-1)}$
 - $\mathbf{i}^{(t)}$ assesses if new word is important to the new memory - determines if input word is worth preserving and *gates* the new memory $\tilde{\mathbf{c}}^{(t)}$

LSTMs: architecture and gated activation functions II

To compute the hidden state $\mathbf{h}^{(t)}$ (with the context state $\mathbf{c}^{(t)}$):

- **Output gate:** $\mathbf{o}^{(t)} = \sigma(W^{(o)}\mathbf{x}^{(t)} + U^{(o)}\mathbf{h}^{(t-1)} + \mathbf{b}_o)$
- **Hidden state:** $\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \circ \tanh(\mathbf{c}^{(t)})$
 - $\mathbf{o}^{(t)}$ indicates what parts of the memory $\mathbf{c}^{(t)}$ needs to be exposed to the hidden state $\mathbf{h}^{(t)}$

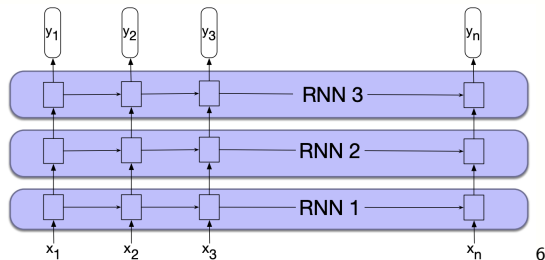
LSTMs: architecture and gated activation functions



5

Stacked / Deep RNNs

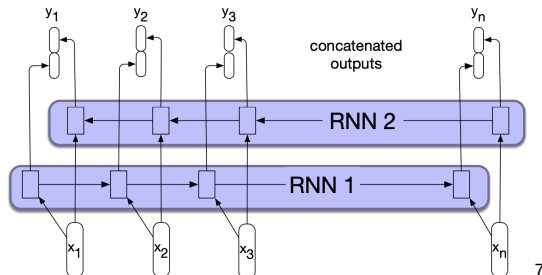
- We can use the entire sequence of outputs from one RNN as the input of another RNN to create a *stack* of RNNs



- Outputs of one layer serves as input to a subsequent layer
- Typically outperforms single-layer networks but increases training cost

Bidirectional RNNs

- Run **two** separate RNNs: left-to-right (we've seen above) and right-to-left
- Then **concatenate** the hidden states from both RNNs
 - Alternatives: element-wise mean or sum



- Effective if you have access to a full input sentence (to **encode** a sentence), e.g. sequence classification or sequence labelling
- “BERT”: **Bidirectional** Encoder Representations from Transformers

References

- Bengio, Y., Ducharme, R., and Vincent, P. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3:1137–1155.
- Brants, T., Popat, A. C., Xu, P., Och, F. J., and Dean, J. (2007). Large Language Models in Machine Translation.
- Brown, P. F., Della Pietra, V. J., Desouza, P. V., Lai, J. C., and Mercer, R. L. (1992). Class-Based n-gram Models of Natural Language. *Computational Linguistics*, 18(4):467–480.
- Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.
- Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471.
- Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.
- Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558.
- Jurafsky, D. and Martin, J. H. (2019). Speech and language processing (3rd (draft) ed.).
- Manning, C., Socher, R., Fang, G. G., and Mundra, R. (2017). CS224n: Natural Language Processing with Deep Learning.
- Olah, C. (2015). Understanding LSTM Networks.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985). Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.