Sequencial Data

RNN

1. when we design a RNN, we usually don't propagate the gradients from through the whole sequence, because it is computationally expensive, actually, a N-to-N RNN while have $N(N+1)/2$ gradients summing up. Instead, we use a something called truncated backpropagation, which means we just choose a chunk of sequence, and forward and backward through it to update the weights.
2. one-hot encoding just extracts one column out of the weight matrix after multipying the input with the weight matrix. So a seperate embedding layer is often used, it can be a pretrained word embedding.
3. after outputing the probalilities of the next word at each step, we have multiply ways to decide on which word to use:
   - determinant: choose the word with the highest probability, this might provide sensible sentences, but it can only provide a single kind of it, because after training is done, given a same input, the output will always be the same.
   - sample by the probability distribution: this is a stochastic approach, we sample a word from the probability distribution, and the output will be different every time. But chances are that the output will go astray at some step by choosing a word with low probability, and than it will continue to have bad influence the generation of next words.
   - exhaustive search: ideally, we mean to find a sequence of length T to maximize the probability

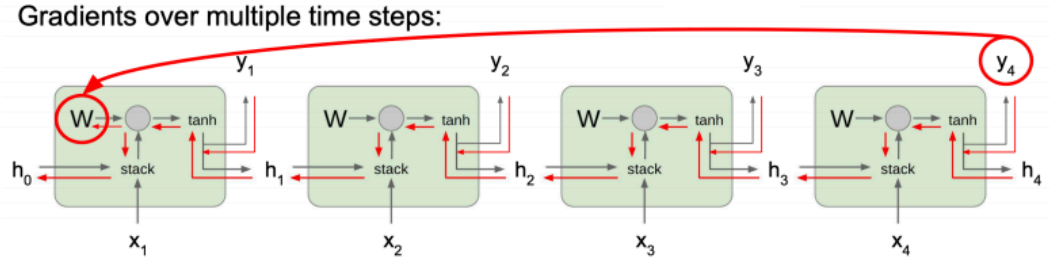
     $$P(y|x) = P(y_1|x)P(y_2|y_1, x)\cdots P(y_T|y_1, \cdots, y_{T-1}, x)$$

     . we can compute the possibility of all sequences and choose the one with the highest probability. But then we need to compute the possibilities of C^T sequences, where C is the vocabulary size.
   - beam search: this is a optimization of exhaustive search, we keep track of k most probable partially generated sequences. For the input token, we compute the probabilities of the second token, and we choose the top k tokens. Then for each of these k tokens, we compute the probabilities of the third token, and we choose the top k tokens, now we have k^2 tokens, we keep top k of them with highest partial probalilities. We repeat this process until we reach the end of the sequence.
4. one undesirable shortcoming of RNN is that it's incapable of learning long-term dependencies
   - the first reason is because of the training strategy: we use truncated backpropagation, which means the model learns logic dependencies only in the window size N, a.k.a, when it trys to predict the next word, it only considers it's relation with the previous N words.
   - the second reason comes from the gradient flow in vanilla RNN:

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Gradients over multiple time steps:



$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W} \qquad \frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}$$

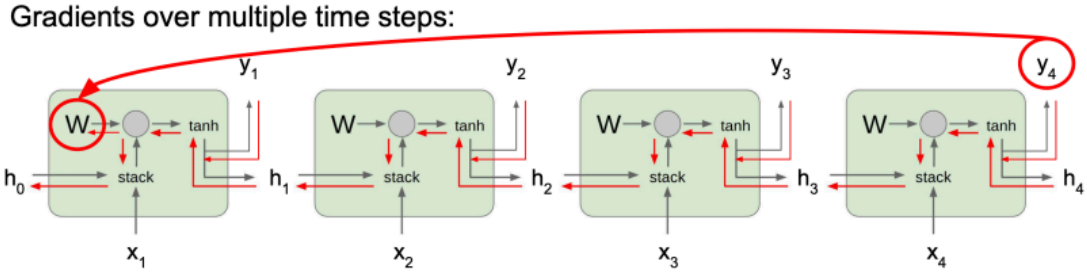
$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T}\frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T}\left(\prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}}\right)\frac{\partial h_1}{\partial W}$$

34

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Gradients over multiple time steps:



$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

Almost always < 1
Vanishing gradients

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T}\left(\prod_{t=2}^{T} tanh'(W_{hh}h_{t-1} + W_{xh}x_t)\right)W_{hh}^{T-1}\frac{\partial h_1}{\partial W}$$

35

vanishing gradient means the loss of later tokens does not have too much influence on earlier steps, a.k.a, the model does not know updating the weights to make previous predictions more suitable to get the right prediction for much later tokens, because the gradients are too small. The model only learns short-term dependencies because the gradients are large and it knows that goes down along this direction can help to make the following few tokens predicts better.

5. summarize:

- RNN advantages:
  i. can process any length input
  ii. computation for step t can (in theory) use information from many steps back
  iii. model size doesn't increase for longer input
  iv. same weihghts applied on every timestep, so there is symmetry in how inputs are processed
- RNN shortcomings:
  i. recurrent computation is slow
  ii. in practice, difficult to access information from many steps back

LSTM

RNN is hard to learn long-term dependencies, one possible solution is to add seperate meomery.

1. model:

Long Short Term Memory (LSTM)

LSTM

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

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

- On step t, there is a hidden state h_t and a cell state c_t
  - Both of them are of same length
- The cell state stores long-term information
- The LSTM can read, erase, and write information from the cell
  - This cell becomes conceptually rather like RAM in a computer

Slides credit: Stanford CS 224 48

Long Short Term Memory (LSTM)

LSTM

Four gates

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

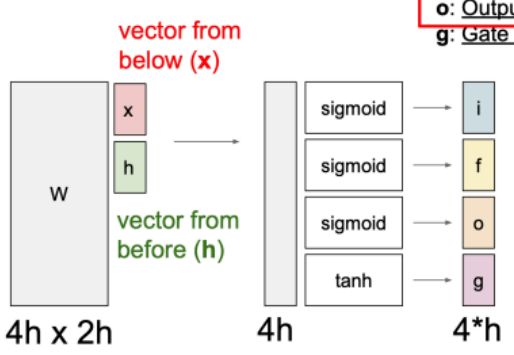
Cell state → $c_t = f \odot c_{t-1} + i \odot g$

Hidden state → $h_t = o \odot \tanh(c_t)$

- The selection of which information is erased/written/read is controlled by three corresponding gates
- The gates are also vectors of length *n*
- On each timestep,each element of the gates can be **open**(1), **closed**(0),or somewhere in-between
- The gates are dynamic: their value is computed based on the current context

Long Short Term Memory (LSTM)

i: Input gate, whether to write to cell
f: Forget gate, Whether to erase cell
o: Output gate, How much to reveal cell
g: Gate gate (?), How much to write to cell



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

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

NOTE: the short-term memory h is deemed is a chunk of long-term memory c; $\odot$ means elements wise multiplication.

2. gradient flow:

- LSTM alleviate the vanishing gradient problem because most easily vanished gradient liek $\partial L/\partial h_t$ or $\partial L/\partial W$ is now a summation of multiple sub-gradients, like $\partial L/\partial W = \partial L/\partial f_t \cdot \partial f_t/\partial W + \partial L/\partial i_t \cdot \partial i_t/\partial W + \partial L/\partial o_t \cdot \partial o_t/\partial W + \partial L/\partial g_t \cdot \partial g_t/\partial W$. This helps to balance the gradient.
- LSTM makes it easier to preserve information for many steps. For example, if $f = 1$ and $i = 0$, then the information is preserved indefinitely
- LSTM doesn't guarantee no vanishing/exploding gradients, but it helps to alleviate the problem, and more importantly, it's easier to learn long-term dependencies.

3. summary

- RNNs allow a lot of flexibility in architecture design
- vanilla RNN is simple but don't work very well
- common to use LSTM or GRU, their additive interactions improve gradient flow
- backward flow of gradients in RNN can explode or vanish. Explodeing is controlled with gradient clipping. Vanishing is controlled with additive interactions.
- Better/simpler architectures are a hot topic of current research, as well as new parasigms for reasoning over sequences.
- Better understanding (both theroretical and empirical) is needed

Bidirectional RNN

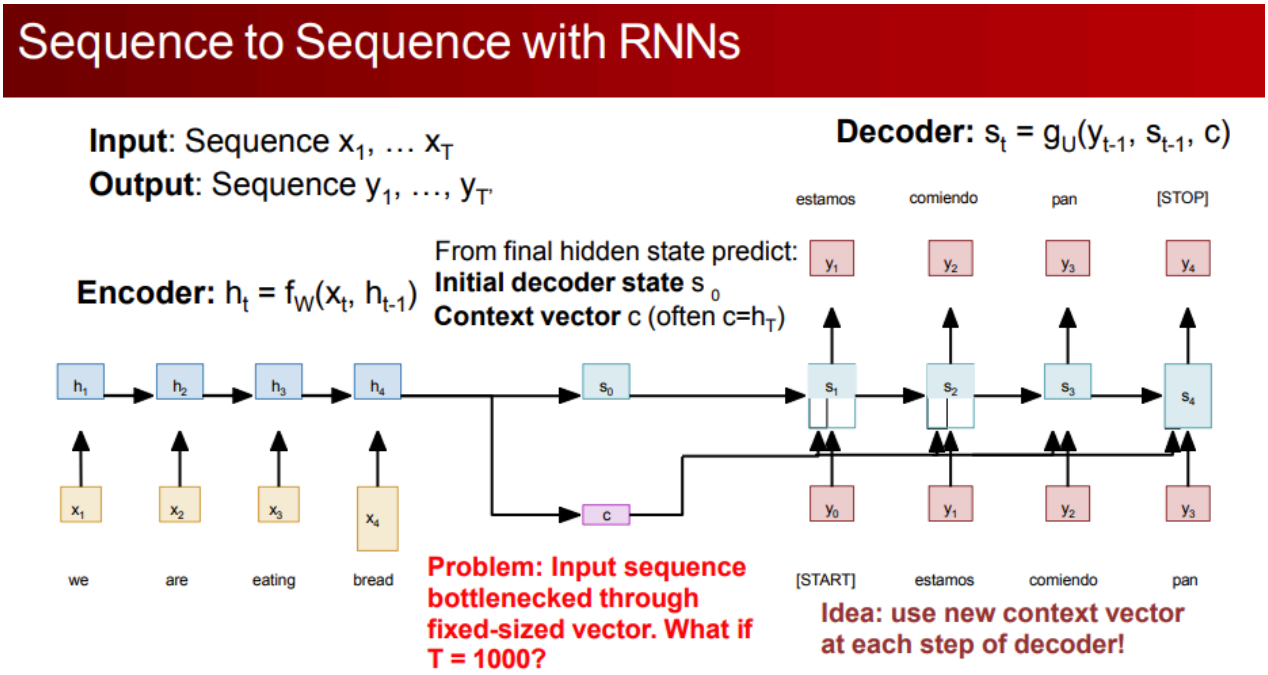
Bidirectional RNN is usually used when we need to have a better understanding of the context, because the perception field of a word now includes the words before and after it. In comparison, vanilla RNN only sees the words before it.

The shortcoming of bidirectional RNN is that it cannot be applied to do generation tasks, because it need to know words before and after to get the hidden state of current word, that's contradictory to the generation task.

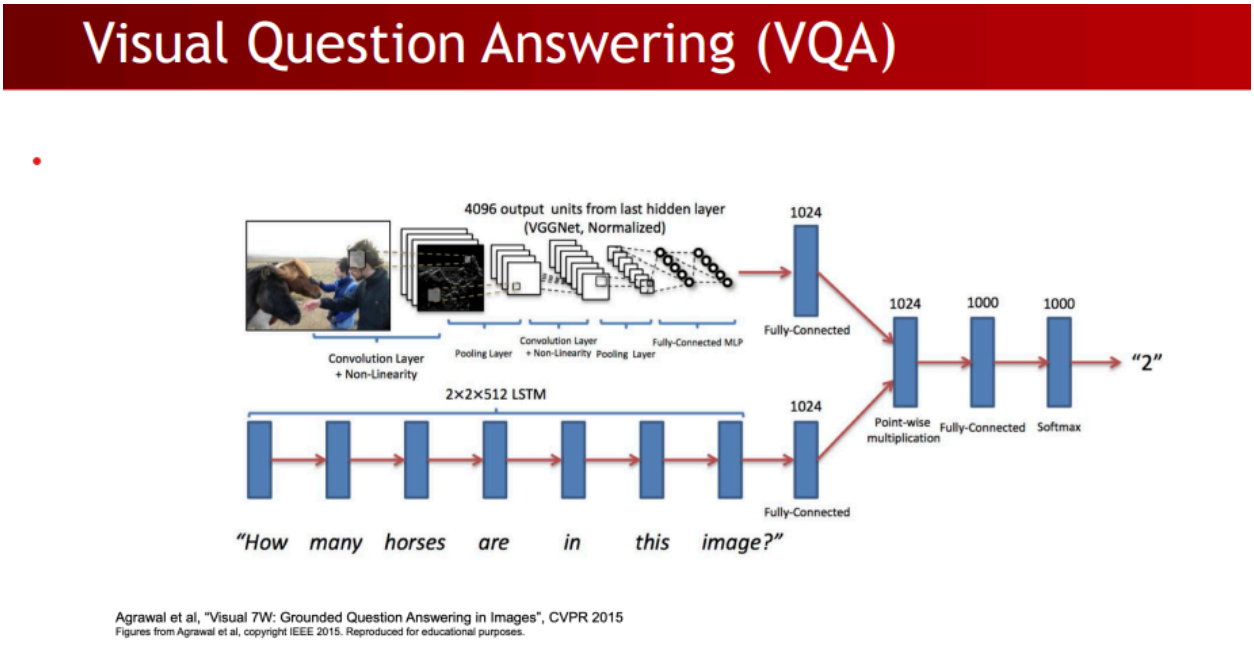
Sequence to Sequence with RNNs

A common task for RNN is to input an article and ask the model to generate a summary, or input an image and ask the model to generate a caption. These are sequence to sequence tasks.

1. vanilla method: use RNN to go through the whole input sequence(bidirectional RNN can be used here to capture the context), then we got a hidden state which, in theory, captures the content of the whole sequence. Then we use the hidden state as input to another RNN, which generates the output sequence.
   - training method: co-train of the first and the second RNN is difficult, because when the first RNN updates, the distribution that the second RNN learns from shifts simultaneously. One strategy is to use a pre-trained model as the first RNN(encoder), and only train the second RNN(decoder), or co-train a small header of the pre-trained model together. Is this way, we need to make sure the pre-trained model can output enough features for the second RNN to generate the output sequence. For example, we may use a encoder pre-trained on ImageNet to do semantic understanding task, but this can actually be problematic. Because one question can be "it the man facing right or left in the image", but when we train the first RNN, we may used some data augmentation like image flipping, and the pre-trained model can simply leave out orientation information in the output feature, so it's impossible for the second RNN to generate the correct output.
   - pipeline:



- the fixed size of context vector limits the representation power of the input to the second RNN.
- analysis:



74

this modal looks fine, but one significant limitation is that the features extracted from the image is fixed no matter what the input text is. Which means the CNN needs to extract extremely rich features to cope with various questions.