Language Modeling

By Kaveri Kale(PhD)
IIT Bombay

Introduction

- A language model learns to predict the probability of a sequence of words.

$$P(W) = P(w_1, w_2, ..., w_n)$$

- Language Model (LM) actually a grammar of a language as it gives the probability of word that will follow

$$P(w_5 | w_1, w_2, w_3, w_4)$$

- In Machine Translation, you take in a bunch of words from a language and convert these words into another language. Now, there can be many potential translations that a system might give you and you will want to compute the probability of each of these translations to understand which one is the most accurate.
- Word Ordering : $P(\textit{the cat is small}) > P(\textit{small the is cat})$

- This ability to model the rules of a language as a probability gives great power for NLP related tasks. Language models are used in speech recognition, machine translation, part-of-speech tagging, parsing, Optical Character Recognition, handwriting recognition, information retrieval, and many other daily tasks.

- There are many sorts of applications for Language Modeling, like: Machine Translation, Spell Correction, Speech Recognition, Summarization, Question Answering, Sentiment analysis , part-of-speech tagging, parsing, Optical Character Recognition, handwriting recognition, information retrieval, and many other daily tasks.

Language Modeling

- Language model is required to represent the text to a form understandable from the machine point of view.
- One way the other way text needs to be normalized. Normalized means that indexed text and query terms must have same form. For example, U.S.A. and u.s.a. are to be considered same (remove dots and case fold - lowercase) and return same result while querying any of them. Thus, an implicit word classes equivalence must be defined.
- language modeling must consider the correlated ordering of tokens. This is because every language is based on some grammar, where order has a lot of influence on the meaning of a text.
- To complete the NLP tasks we must provide a measure to enable comparison operations and thus assessment method for grading our model. This measure is probability. This involves all kinds of tasks, for example:
- **Machine translation:** translating a sentence saying about height it would probably state that $P(tall\ man) > P(large\ man)$ as the ‘large’ might also refer to weight or general appearance thus, not as probable as ‘tall’.
- **Spelling Correction:** Spell correcting sentence: “Put you name into form”, so that P(name into form)>P(name into from)
- **Speech Recognition:** Call my nurse: $P(Call\ my\ nurse) \gg P(coal\ miners)$, I have no idea. $P(no\ idea.) \gg P(No\ eye\ deer.)$
- **Summarization, question answering, sentiment analysis etc.**
- We use language model everyday

  e.g. Google search, whatsapp

The Probabilistic language Modeling

- The probability of a sentence S (as a sequence of words w_i) is :

$$P(S) = P(w_1, w_2, w_3, \ldots, w_n)$$

- Now it is important to find the probability of upcoming word. It is an everyday task made e.g. while typing your mobile keyboard. We will settle the conditional probability of w_4 depending on all previous words.
- For a 4 word sentence this conditional probability is:

$$P(S) = P(w_1, w_2, w_3, w_4) \equiv P(w_4 | w_1, w_2, w_3)$$

- **Probability Chain Rule:**

$$p(w_1, \ldots, w_n) = p(w_1)\, p(w_2|w_1)\, p(w_3|w_1 w_2)\, p(w_4|w_1 w_2 w_3), \ldots\ldots, p(w_n|w_1, \ldots, w_{n-1})$$

Example:

$$P(\text{“ its water is so transparent ”}) = P(its)\, P(water|its)\, P(is|its\ water)\, P(so|its\ water\ is)\, P(transparent|its\ water\ is\ so)$$

- To estimate each probability a straightforward solution could be to use simple counting.

$$P(w_5|w_1, w_2, w_3, w_4) = \frac{count(w_1, w_2, w_3, w_4, w_5)}{count(w_1, w_2, w_3, w_4)}$$

  but this gives us to many possible sequences to ever estimate. Imagine how much data (occurrences of each sentence) we would have to get to make this counts meaningful.

- To cope with this issue we can simplify by applying the Markov Assumption, which states that it is enough to pick only one, or a couple of previous words as a prefix:

$$P(w_{1,..}, w_n) \approx \prod_i P(w_i|w_{i-k}, ..., w_{i-1})$$

where k is the number of previous tokens that we consider.

N-gram models

- An N-gram is a contiguous (order matters) sequence of items, which in this case is the words in text. The n-grams depends on the size of the prefix.
- **Definition**: An N-gram is a sequence of N tokens (or words).
  Consider the sentence: “I love reading blogs about NLP”

1. **unigrams:** A 1-gram (or unigram) is a one-word sequence.

   “I”, “love”, “reading”, ”blogs”, “about”, “NLP”

   $$P(w_1, w_2, \ldots, w_n) = \prod_i P(w_i)$$

2. **bigrams:** A 2-gram (or bigram) is a two-word sequence of words.

   “I love”, “love reading”, “reading blogs”, “blogs about”, “about NLP”

   $$P(w_i | w_{1,} w_2, \ldots, w_n) = \prod_i P(w_i | w_{i-1})$$

3. **trigrams:** A 3-gram (or trigram) is a three-word sequence of words.

   “I love reading”, “ love reading blogs”, “ reading blogs about”, “ blogs about NLP”

Estimate n-gram probabilities

- Estimation can be done with Maximum Likelihood Estimate (MLE):

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

- So the 2-gram estimate of sentence probability would be the product of all component tandems ordered as in the sentence.

$$P(w_1, \ldots, w_n) \approx \prod_i P(w_i|w_{i-1})$$

- In practice, the outcome should be represented in log form. There are two reasons for this. Firstly, if the sentence is long and the probabilities are really small, then such product might end in arithmetic underflow. Secondly, adding is faster - and when we use logarithm we know that:

$$\log(a * b) = \log(a) + \log(b)$$

Thus,

$$\log(P(w_1, \ldots, w_n)) \approx \sum_i \log(P(w_i|w_{i-1}))$$

Definition: Perplexity

Perplexity is the inverse probability of the test set normalised by the number of words, more specifically can be defined by the following equation:

$$PP(W) = P(w_1, w_2, ..., w_N)^{\frac{1}{N}}$$

e.g. Suppose a sentence consists of random digits [0–9], what is the perplexity of this sentence by a model that assigns an equal probability (i.e. P=1/10) to each digit?

$$PP(W) = (\frac{1}{10} * \frac{1}{10} * .... * \frac{1}{10})^{\frac{-1}{10}} = (\frac{1}{10}^{10})^{\frac{-1}{10}} = 10$$

Entropy of Language :

- Entropy of sequence of words : $H(w_{1,}w_2,...w_n)=-\sum P(w_1,w_2,...,w_n)\log_2 P(w_1,w_2,...,w_n)$
- The per-word entropy rate of sequence of words : $\frac{1}{n}H(w_1,w_2,...,w_n)=\frac{-1}{n}\sum P(w_1,...,w_n)\log_2 P(w_1,w_2,...,w_n)$
- Entropy of language : $L=\{w_1,w_2,...,w_n|1<n<\infty\}$

Definition : Cross Entropy

The cross entropy H(p,m) of a true distribution p and model distribution m is defined as:

$$H(p,m)=-\sum P(x)\log_2 m(x)$$

The lower the cross entropy is the closer it is to the true distribution.

Cross entropy of a Sequence of Words:

$$H(p,m)=-\lim_{n\to\infty}\frac{1}{n}\sum P(w_1,...,w_n)\log_2 m(w_1,...,w_n)$$

Perplexity and Entropy

$$PP(w_1,...,w_n)=2^{H(w_1,...,w_n)}$$

Limitations of N-gram approach to Language Modeling

N-gram based language models do have a few drawbacks:

- The higher the N, the better is the model usually. But this leads to lots of computation overhead that requires large computation power in terms of RAM
- N-grams are a sparse representation of language. This is because we build the model based on the probability of words co-occurring. It will give zero probability to all the words that are not present in the training corpus

Recurrent Neural Networks (RNN)

- Why not standard neural network?

  There are primarily two problems with this:

  1. Inputs and outputs do not have a fixed length, i.e., some input sentences can be of 10 words while others could be <> 10. The same is true for the eventual output.
  2. We will not be able to share features learned across different positions of text if we use a standard neural network.
- RNNs are designed to take a series of input with no predetermined limit on size.
- We need a representation that will help us to parse through different sentence lengths as well as reduce the number of parameters in the model. This is where we use a recurrent neural network.

- Figure introduces the RNN architecture where each vertical rectangular box is a hidden layer at a time-step, t.
- At each time-step, there are two inputs to the hidden layer: the output of the previous layer h_{t-1}, and the input at that time-step x_t. The former input is multiplied by a weight matrix W_{hh} and the latter by a weight matrix W_{xh} to produce output features h_t, which are multiplied with a weight matrix W_{hy} and run through a softmax over the vocabulary to obtain a prediction output $\hat{y}$ of the next word.
- The RNN scans through the data in a left to right sequence. Note that the parameters that the RNN uses for each time step are shared. We will have parameters shared between each input and hidden layer W_{xh}, every timestep W_{hh} and between the hidden layer and the output W_{hy}.

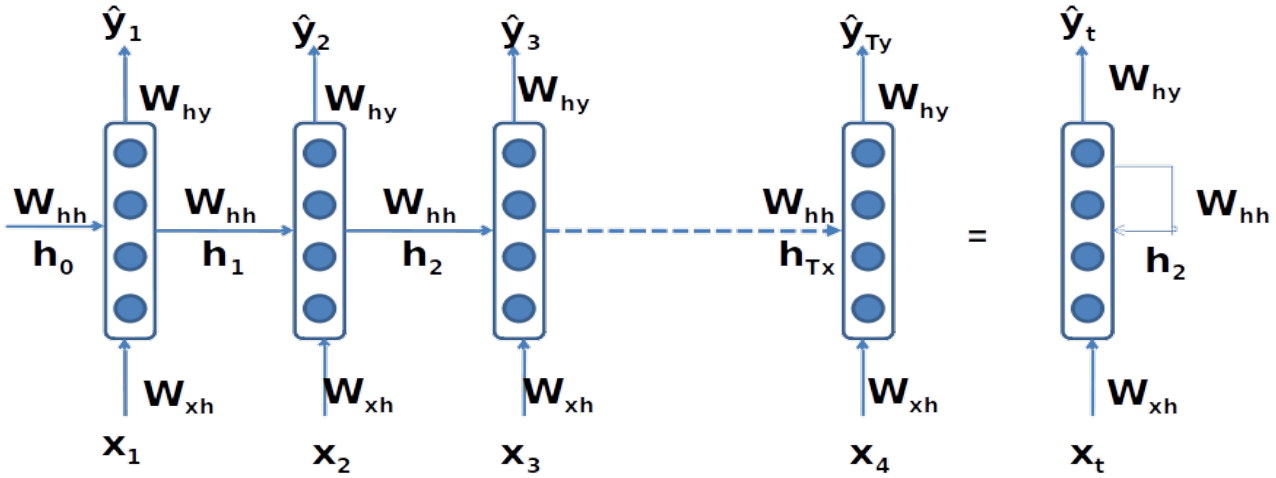


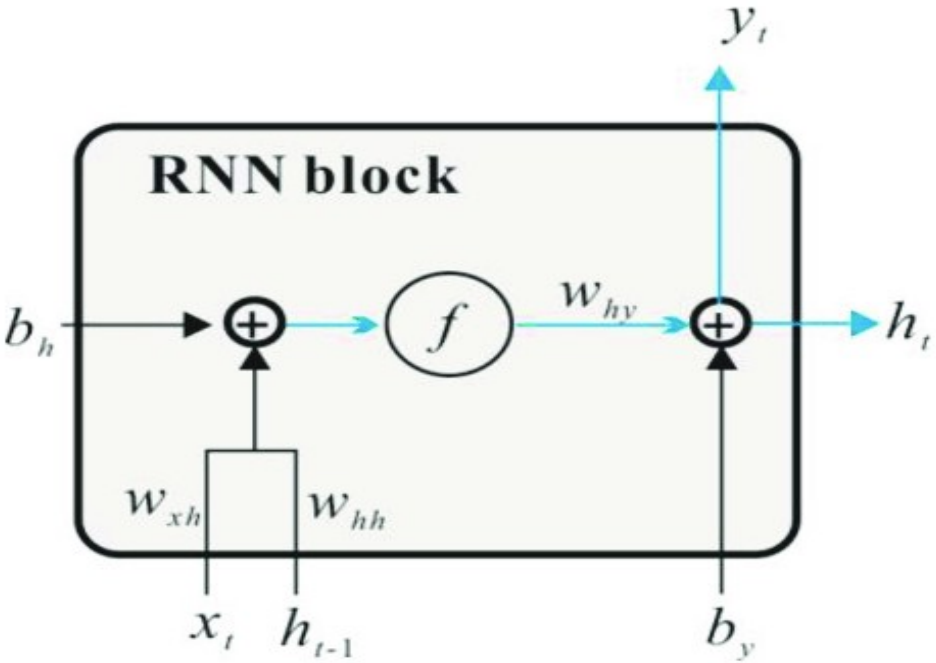
Figure : Recurrent Neural Network

- The inputs and outputs of each single neuron are illustrated in Figure

$$h_t = \sigma(W_{hh} h_{t-1} + W_{xh} x_t + b_h)$$

$$\hat{y}_t = softmax(W_{hy} h_t + b_y)$$

- What is interesting here is that the same weights W_{hh} and W_{xh} are applied repeatedly at each timestep. Thus, the number of parameters the model has to learn is less, and most importantly, is independent of the length of the input sequence.



Backpropagation Through Time

- We have a loss function which we need to minimize in order to generate accurate predictions. The loss function is given by:

$$L_t(\hat{y}_t, y_t) = -y_t \log y_t - (1 - y_t) \log(1 - \hat{y}_t)$$

$$L(\hat{y}, y) = \sum_{t=1}^{T} L_t(\hat{y}_t, y_t) \sum_{k=1}^{n} a_k$$

To calculate gradient we pass message backward it is called **Backpropagation Through Time**.

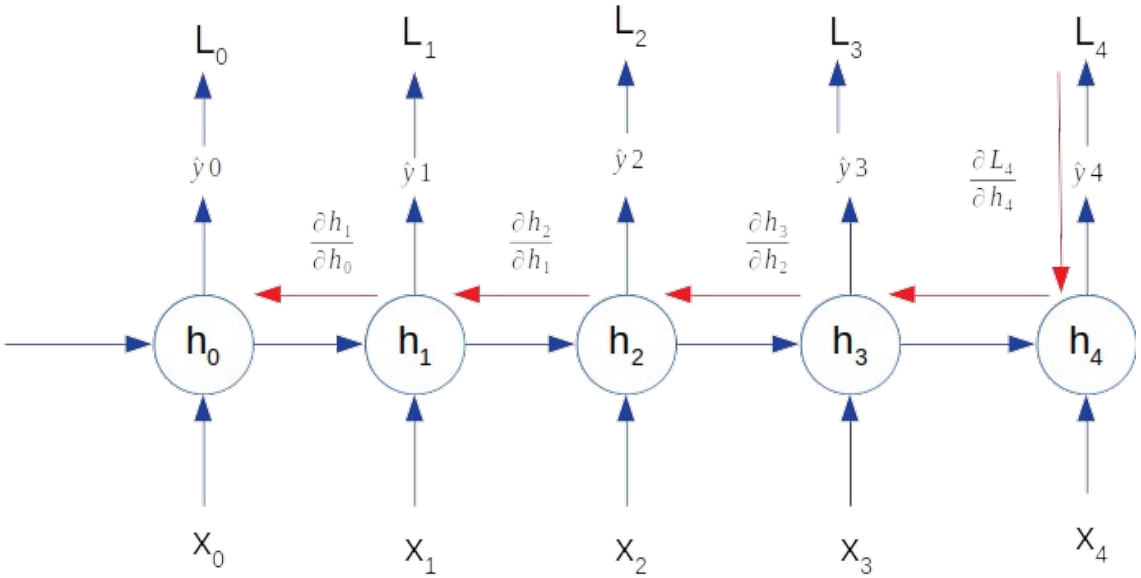
Backpropagation is done at each point in time. At timestep T, the derivative of the loss L with respect to weight matrix W is expressed as follows:

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

To calculate these gradients we use the chain rule of differentiation.

$$\frac{\partial L_4}{\partial W} = \sum_{t=0}^{4} \frac{\partial L_4}{\partial \hat{y}_4} \frac{\partial y_4}{\partial h_4} \frac{\partial h_4}{\partial h_k} \frac{\partial h_k}{\partial W}$$

In practice, it is difficult to access information from many steps back due to problems like vanishing and exploding gradients.

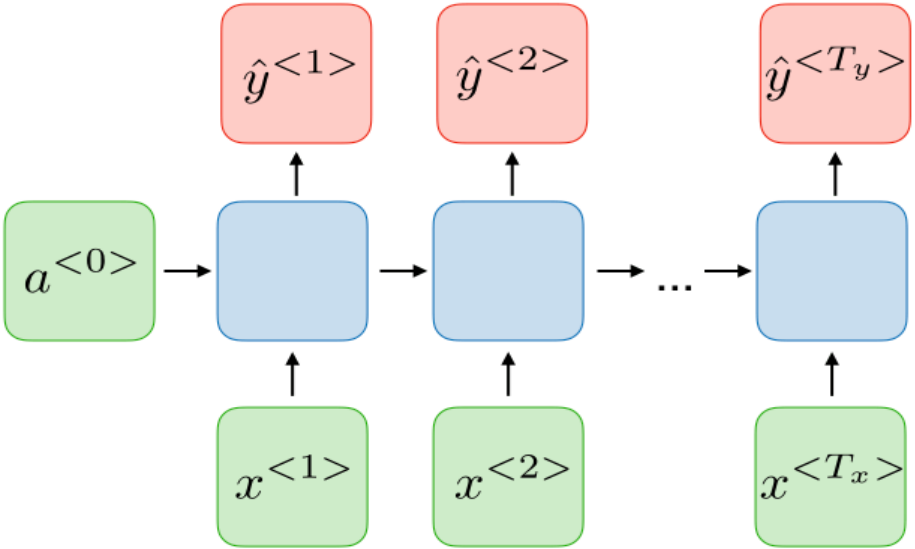


Types of RNN

1. Many-to-many

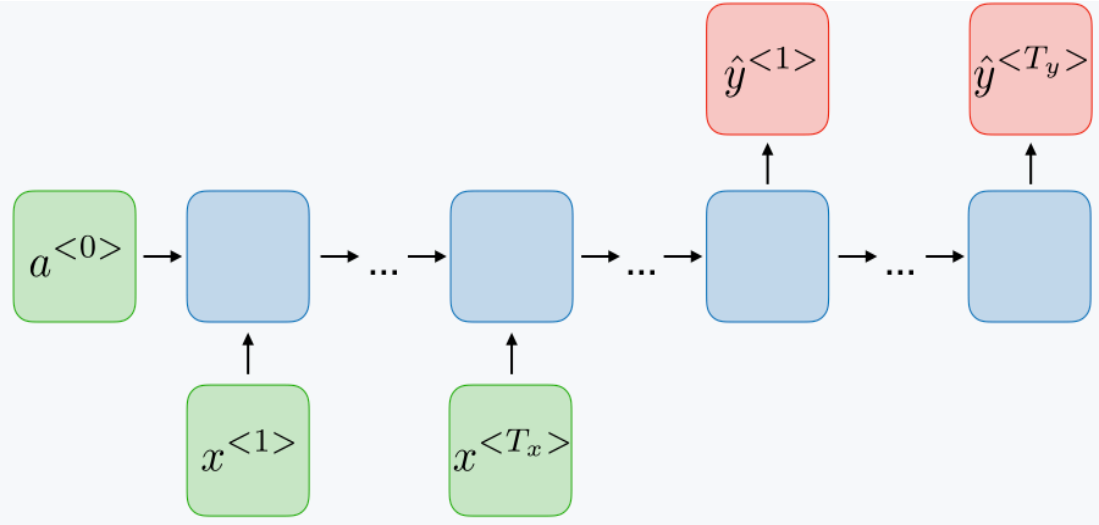
- Tx = Ty

Application : Named entity recognition



- Tx != Ty

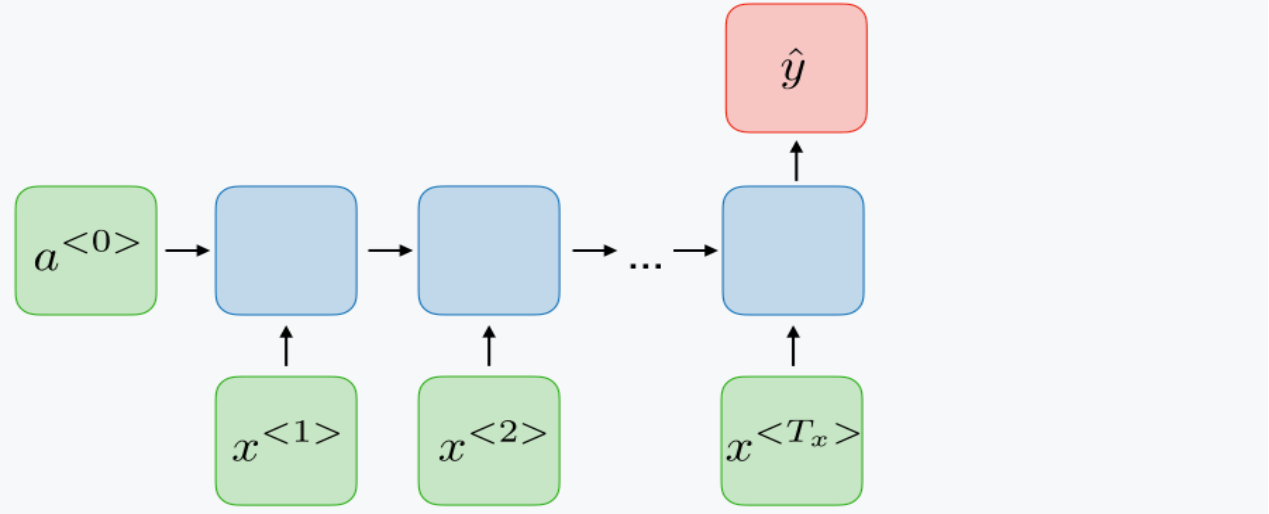
Application : Machine Translation



2. Many-to-one :

Tx > 1, Ty = 1

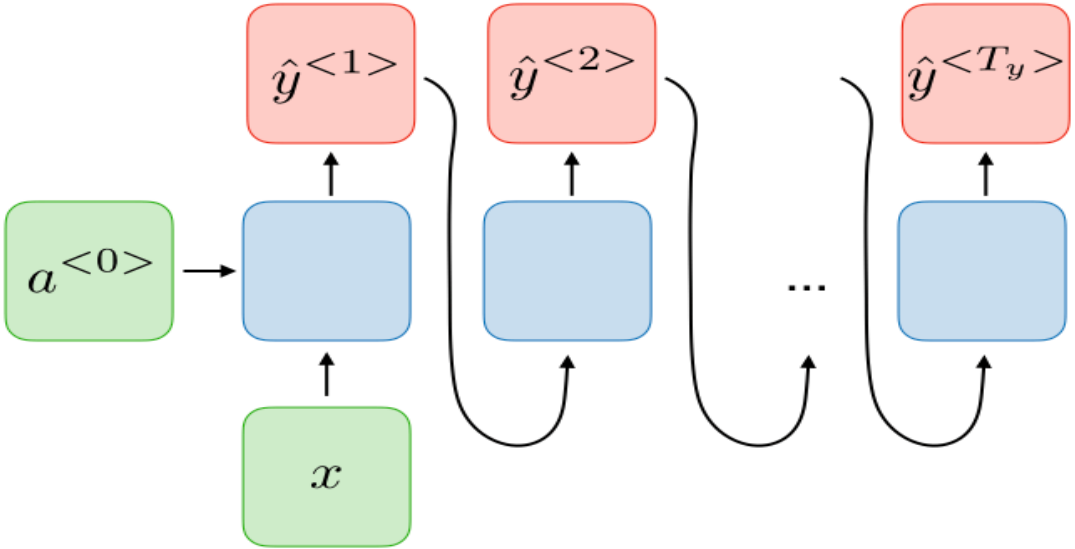
Application : Sentiment classification



3. One-to-many:

Tx = 1, Ty > 1

Application : Music generation

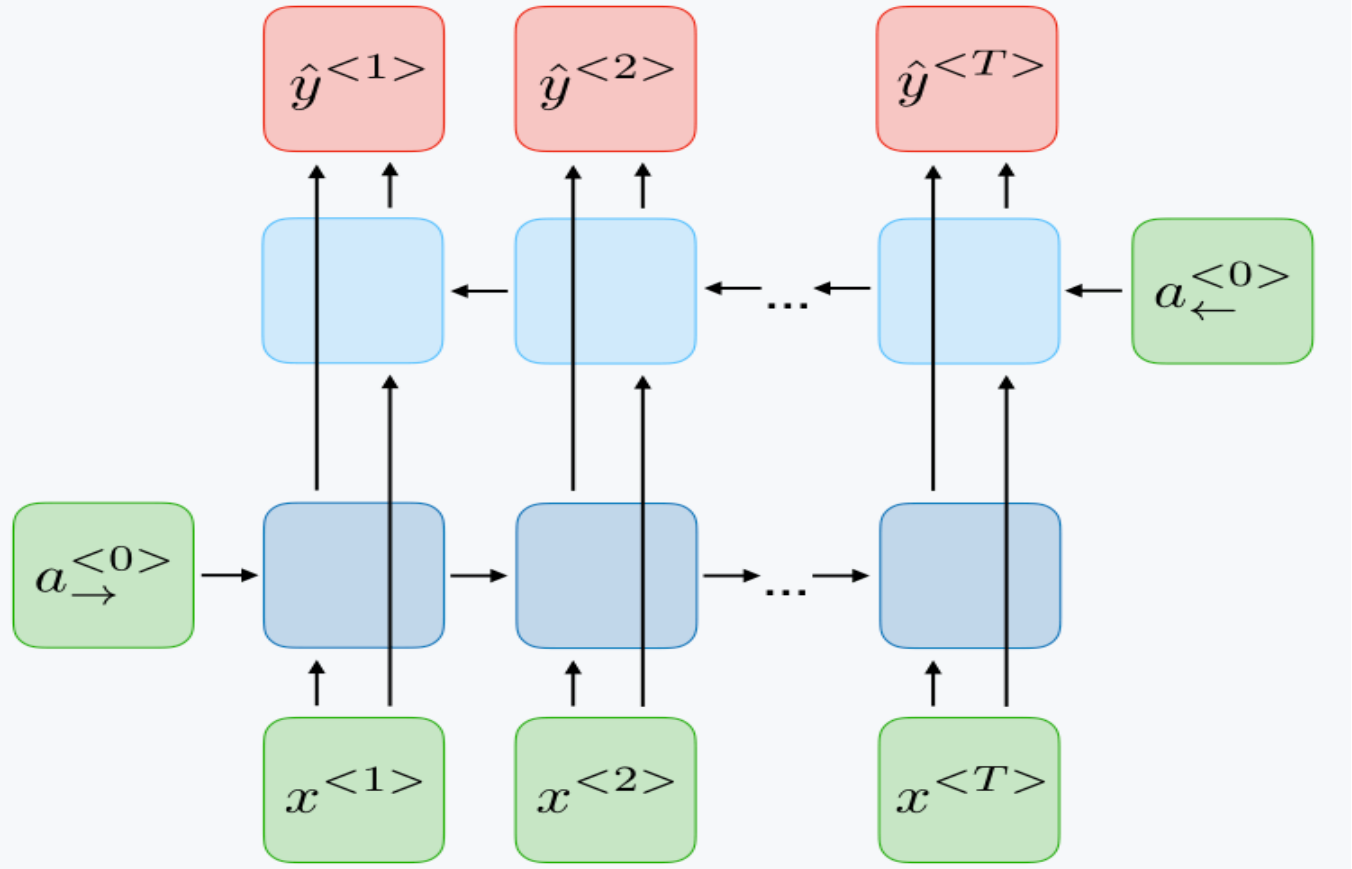


4. One-to-one:

Tx = Ty = 1

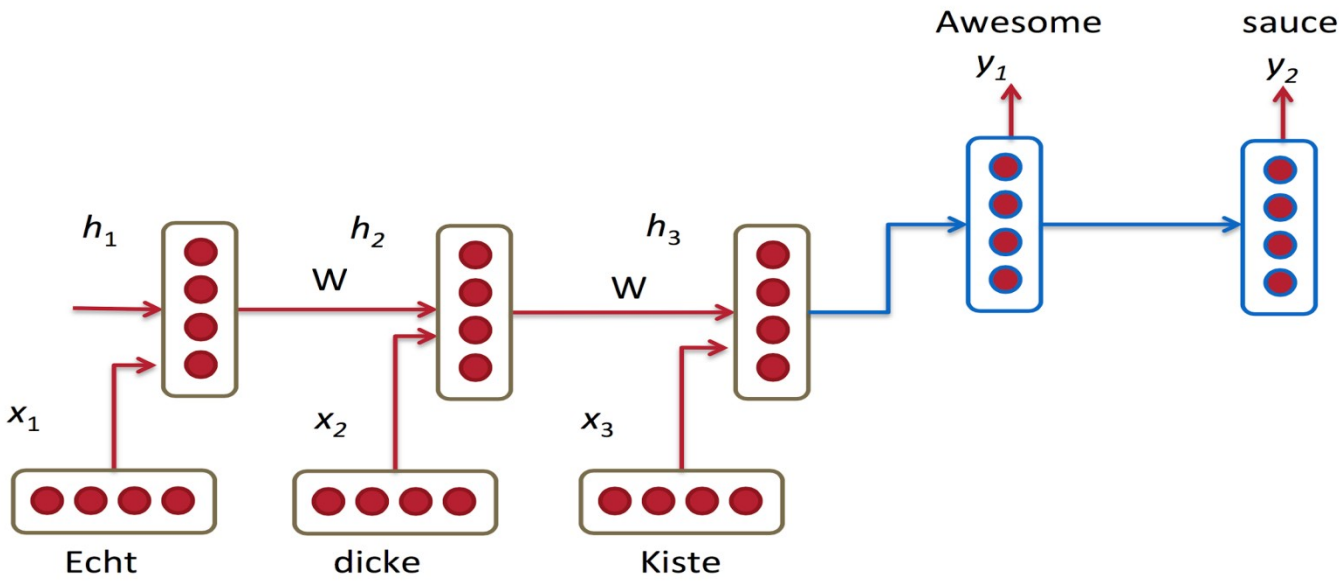
Application : Traditional neural network

Bidirectional RNN



RNN for Machine Translation

- Machine Translation is similar to language modeling in that our input is a sequence of words in our source language (e.g. German). We want to output a sequence of words in our target language (e.g. English).
- A key difference is that our output only starts after we have seen the complete input, because the first word of our translated sentences may require information captured from the complete input sequence.



Advantages of RNN

RNNs have several advantages:

1. They can process input sequences of any length
2. The model size does not increase for longer input sequence lengths
3. Computation for step t can (in theory) use information from many steps back.
4. The same weights are applied to every timestep of the input, so there is symmetry in how inputs are processed

Disadvantages of RNN

1. Computation is slow - because it is sequential, it cannot be parallelized
2. In practice, it is difficult to access information from many steps back due to problems like vanishing and exploding gradients.

Gated Recurrent Unit

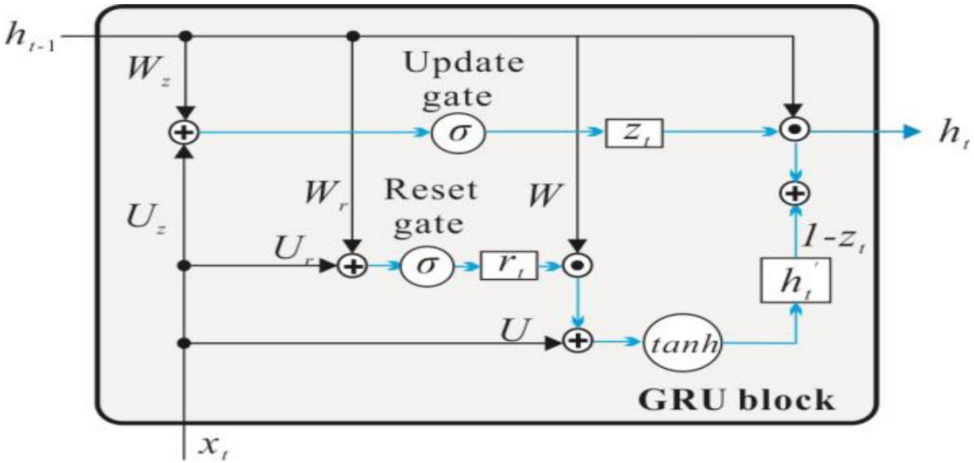
•Well, although RNNs can theoretically capture long-term dependencies, they are very hard to actually train to do this.
•Gated recurrent units are designed in a manner to have more persistent memory thereby making it easier for RNNs to capture long-term dependencies.
•Let us see mathematically how a GRU uses h_{t-1} and x_t to generate the next hidden state h_t . We will then dive into the intuition of this architecture.

$$z_t = \sigma\left(W_z x_t + U_z h_{t-1}\right) \quad \text{(Update gate)}$$

$$r_t = \sigma\left(W_r x_t + U_r h_{t-1}\right) \quad \text{(Reset gate)}$$

$$\tilde{h}_t = \tanh\left(r_t * U h_{t-1} + W x_t\right) \quad \text{(New memory)}$$

$$h_t = \left(1 - z_t\right) * \tilde{h}_t + z_t * h_{t-1} \quad \text{(Hidden state)}$$



GRU’s four fundamental operational

1. New memory generation: A new memory $\tilde{h}_t$ is the consolidation of a new input word x_t with the past hidden state h_{t-1}.

2. Reset Gate: The reset signal r_t is responsible for determining how important h_{t-1} is to the summarization $\tilde{h}_t$. The reset gate has the ability to completely diminish past hidden state if it finds that h_{t-1} is irrelevant to the computation of the new memory.

3. Update Gate: The update signal z_t is responsible for determining how much of h_{t-1} should be carried forward to the next state. For instance, if $z_t \approx 1$, then h_{t-1} is almost entirely copied out to ht . Conversely, if $z_t \approx 0$, then mostly the new memory $\tilde{h}t$ is forwarded to the next hidden state.

4. Hidden state: The hidden state ht is finally generated using the past hidden input h_{t-1} and the new memory generated $\tilde{h}_t$ with the advice of the update gate.

Long-Short-Term-Memories

LSTMs are explicitly designed to avoid the long-term dependency problem.
The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.
Let us first take a look at the mathematical formulation of LSTM units before diving into the intuition behind this design:

$$i_t = \sigma\left(W_i x_t + U_i h_{t-1}\right)$$ (Input gate)

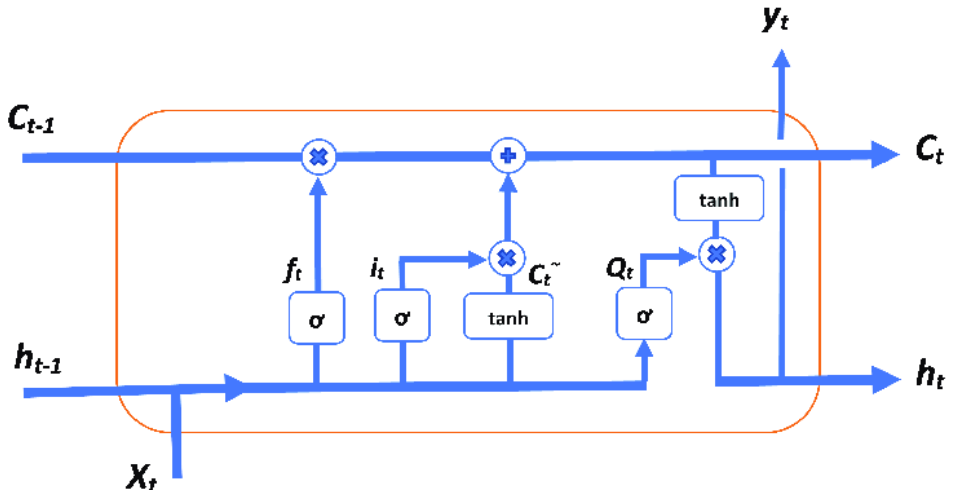
$$f_t = \sigma\left(W_f x_t + U_f h_{t-1}\right)$$ (Forget gate)

$$o_t = \sigma\left(W_o x_t + U_o h_{t-1}\right)$$ (Output/Exposure gate)

$$\tilde{c}_t = \tanh\left(W_c x_t + U_c h_{t-1}\right)$$ (New memory cell)

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$ (Final memory cell)

$$h_t = o_t * \tanh\left(c_t\right)$$



1. New memory generation: This stage is analogous to the new memory generation stage we saw in GRUs. We essentially use the input word xt and the past hidden state ht−1 to generate a new memory c˜t which includes aspects of the new word x (t) .

2. Input Gate: We see that the new memory generation stage doesn't check if the new word is even important before generating the new memory – this is exactly the input gate's function. The input gate uses the input word and the past hidden state to determine whether or not the input is worth preserving and thus is used to gate the new memory. It thus produces it as an indicator of this information.

3. Forget Gate: This gate is similar to the input gate except that it does not make a determination of usefulness of the input word – instead it makes an assessment on whether the past memory cell is useful for the computation of the current memory cell. Thus, the forget gate looks at the input word and the past hidden state and produces ft .

4. Final memory generation: This stage first takes the advice of the forget gate ft and accordingly forgets the past memory ct−1. Similarly, it takes the advice of the input gate it and accordingly gates the new memory c˜t . It then sums these two results to produce the final memory ct .

5. Output/Exposure Gate: This is a gate that does not explicitly exist in GRUs. It's purpose is to separate the final memory from the hidden state. The final memory ct contains a lot of information that is not necessarily required to be saved in the hidden state. Hidden states are used in every single gate of an LSTM and thus, this gate makes the assessment regarding what parts of the memory ct needs to be exposed/present in the hidden state ht . The signal it produces to indicate this is ot and this is used to gate the point-wise tanh of the memory

References

https://www.researchgate.net/

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

https://www.analyticsvidhya.com/blog/2019/01/sequence-models-deeplearning/

http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/

https://www.coursera.org/learn/nlp-sequence-models/