RNN Architectures

EE807: Recent Advances in Deep Learning

Lecture 4

Slide made by

Hyungwon Choi, Jongheon Jeong, and Sangwoo Mo

KAIST EE

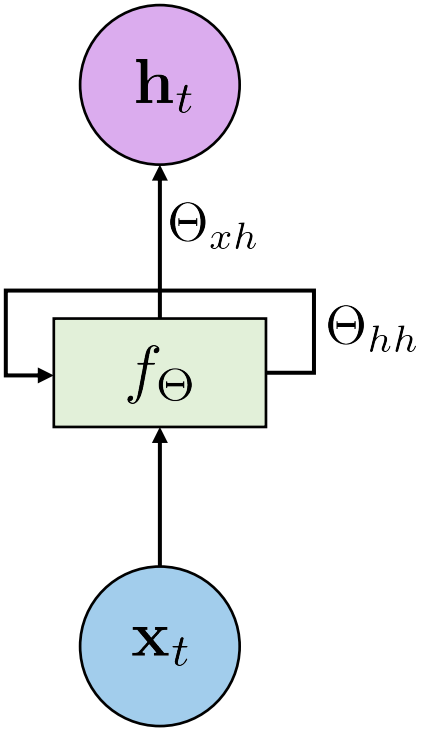
Recap: RNN basics

- Process a sequence of vectors by applying **recurrence formula** at every time step :

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t; \Theta)$$

New state — Old state — Input vector at time step t

Function parameterized by Θ



*reference: http://cs231n.stanford.edu/2017/

Recap: Vanilla RNN

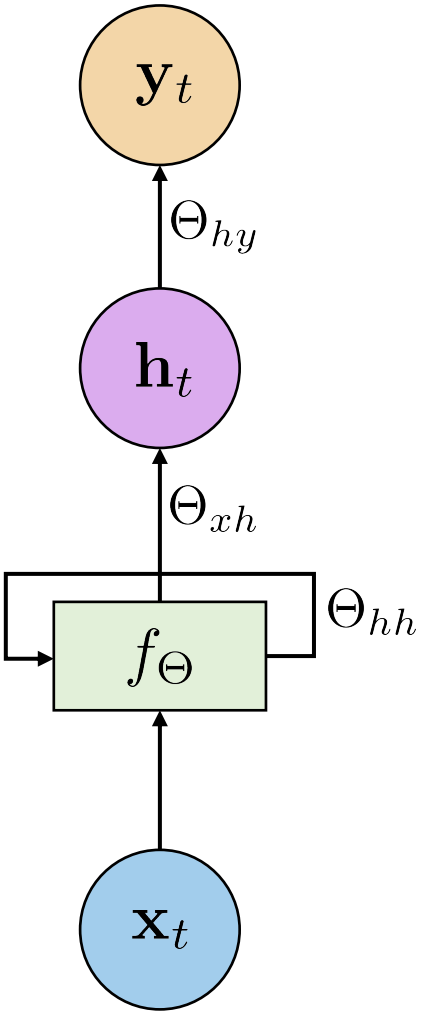
- Simple RNN
  - The state consists of a single "hidden" vector $\mathbf{h}_t$
  - Vanilla RNN (or sometimes called Elman RNN)

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t; \Theta)$$

$$\downarrow$$

$$\mathbf{h}_t = \tanh(\Theta_{hh}\mathbf{h}_{t-1} + \Theta_{xh}\mathbf{x}_t)$$

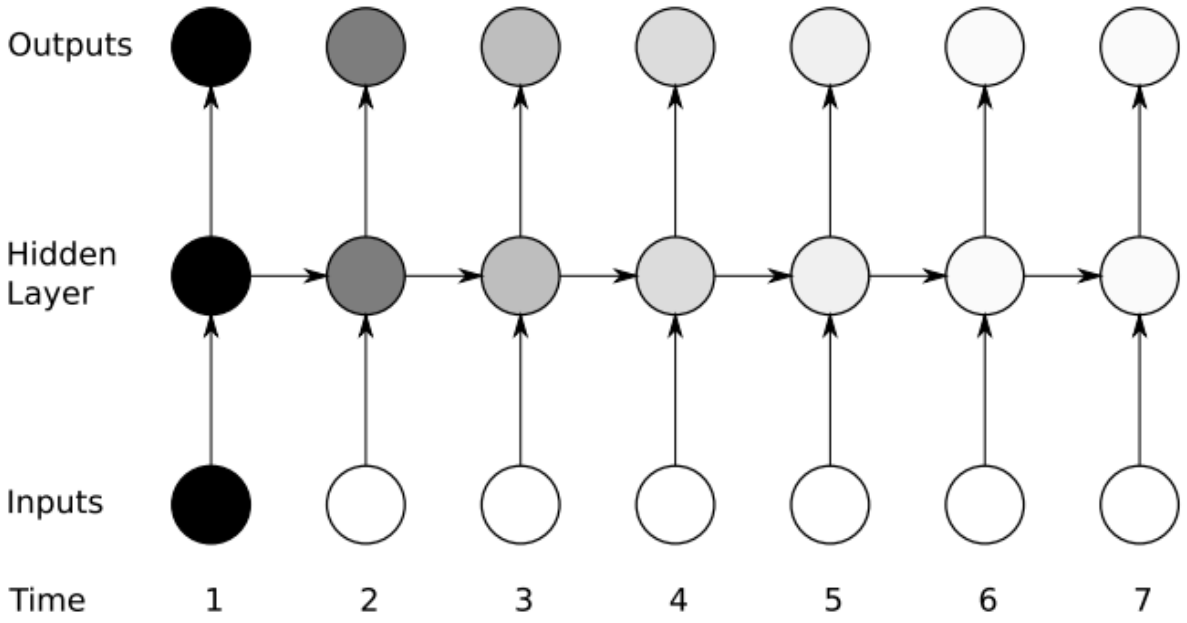
$$\mathbf{y}_t = \Theta_{hy}\mathbf{h}_t$$



*reference: http://cs231n.stanford.edu/2017/

Why do we develop RNN architectures?

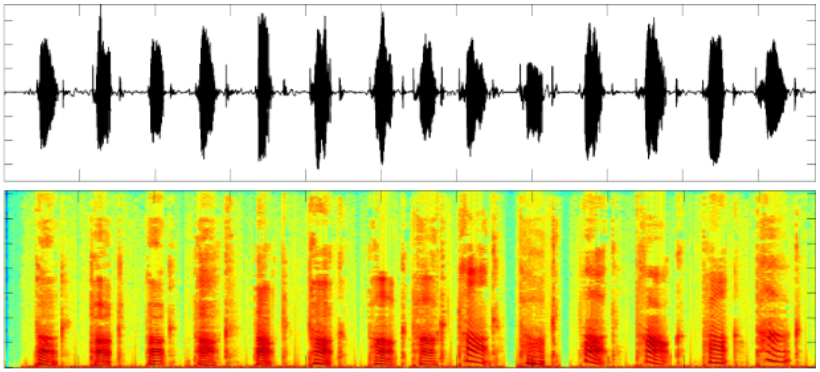
- For vanilla RNN, it is difficult to capture **long-term dependency**
- Vanishing gradient problem in vanilla RNN
  - The **gradient vanishes** over time
  - Which relates to **optimization difficulties** in CNN



*source: https://mediatum.ub.tum.de/doc/673554/file.pdf

Why do we develop RNN architectures?

- Many real-world temporal data is intrinsically **long-term**
  - Natural language
  - Speech
  - Video



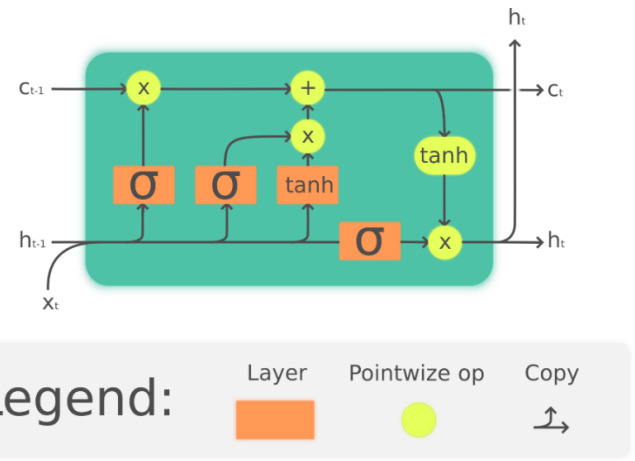
- In order to solve much complicated **real-world problems** we need a better RNN architecture to capture **long-term dependency** in the data

Table of Contents

Table of Contents

RNN Architectures: LSTM

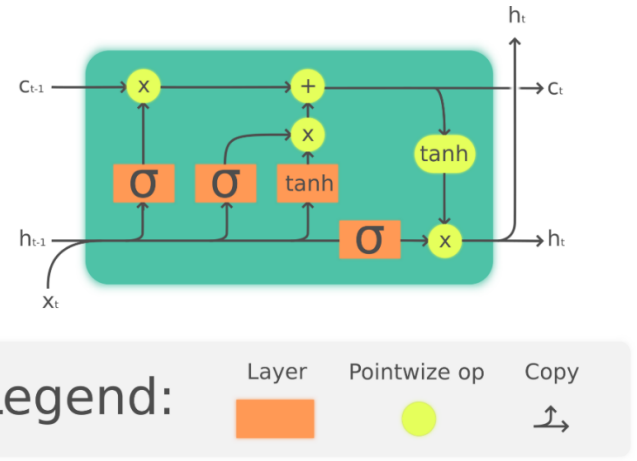
- **Long Short-Term Memory (LSTM)**
  - A special type of RNN unit
    - i.e., LSTM networks = RNN composed of LSTM units
  - Originally proposed by [Hochreiter and Schmidhuber, 1997]
  - Explicitly designed RNN to
    - Capture **long-term dependency**
    - More **robust to vanishing gradient problem**
  - Composed of a cell, an input gate, an output gate, and a forget gate (will be covered in detail soon)



*source: https://en.wikipedia.org/wiki/Long_short-term_memory#/media/File:The_LSTM_cell.png

RNN Architectures: LSTM

- A very old model, but why so popular?
  - Popularized by series of following works
    [Graves et al, 2013][Sutskever et al., 2014], ...
  - Work very well in variety of problems
    - Speech recognition
    - Machine translation
    - …

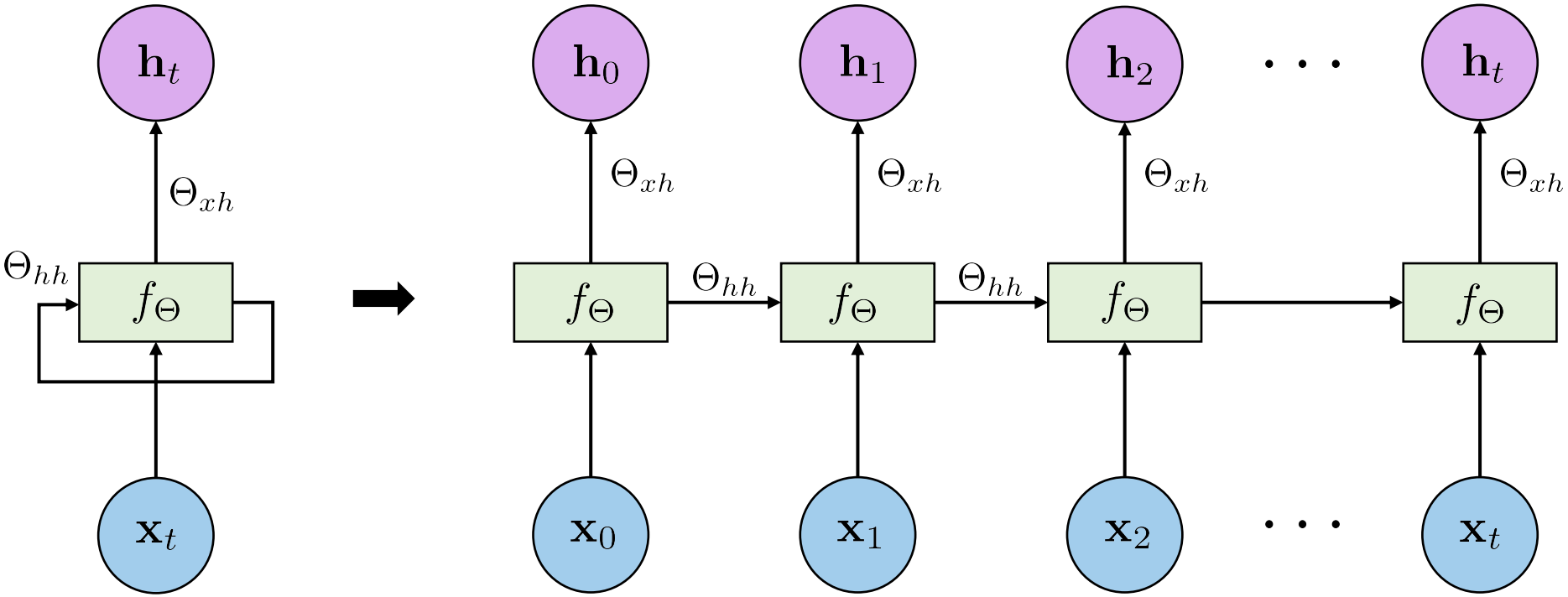


Next, comparison with Vanilla RNN

*source: https://en.wikipedia.org/wiki/Long_short-term_memory#/media/File:The_LSTM_cell.png

RNN Architectures: Vanilla RNN

- Vanilla RNN (unrolled)

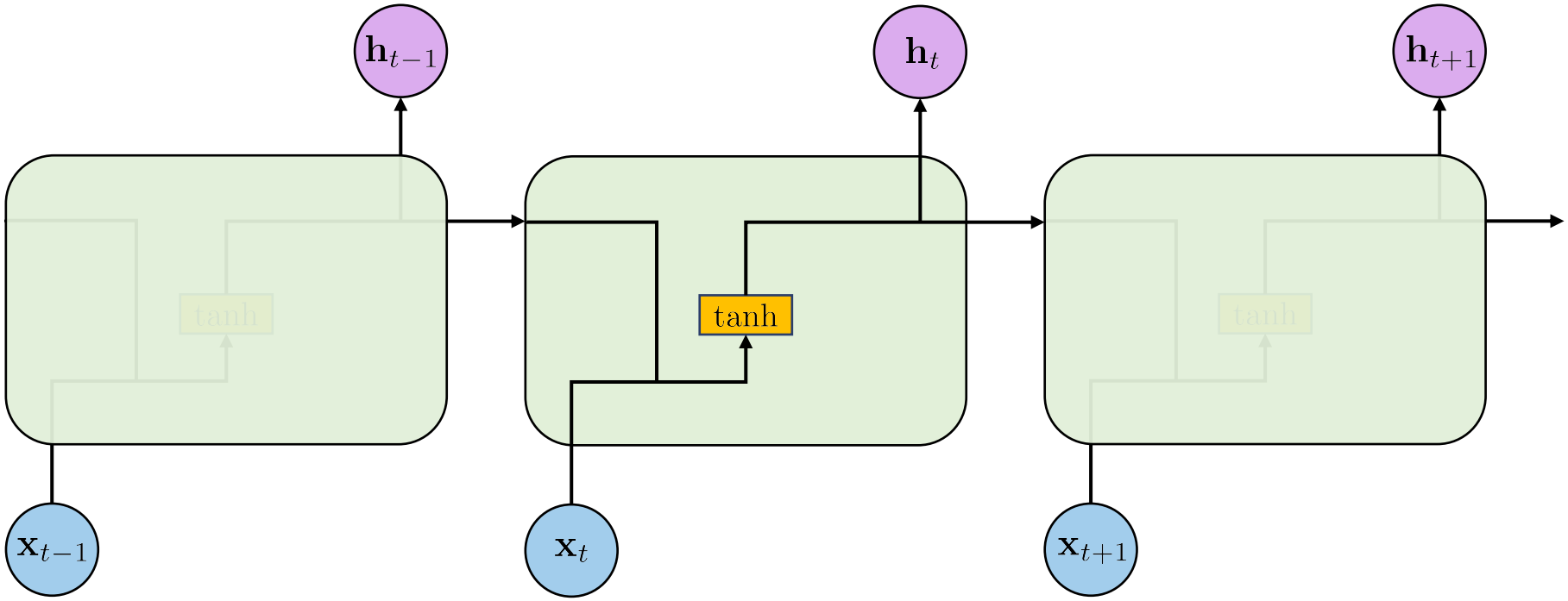


*reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN Architectures: Vanilla RNN

- Repeating modules in **Vanilla RNN** contains a **single layer**

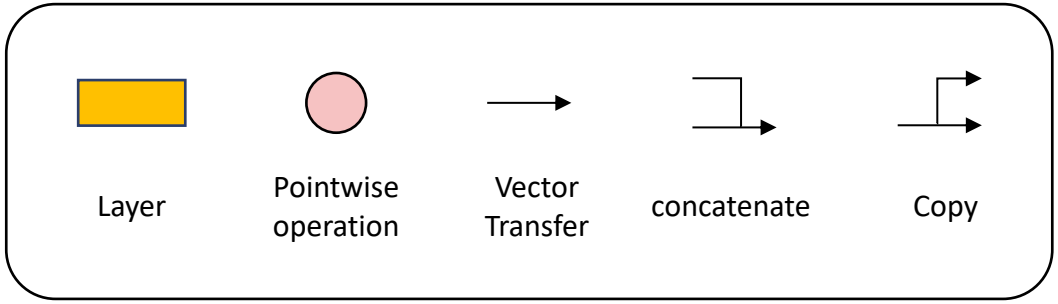
$$\mathbf{h}_t = \tanh(\Theta_{hh}\mathbf{h}_{t-1} + \Theta_{xh}\mathbf{x}_t)$$

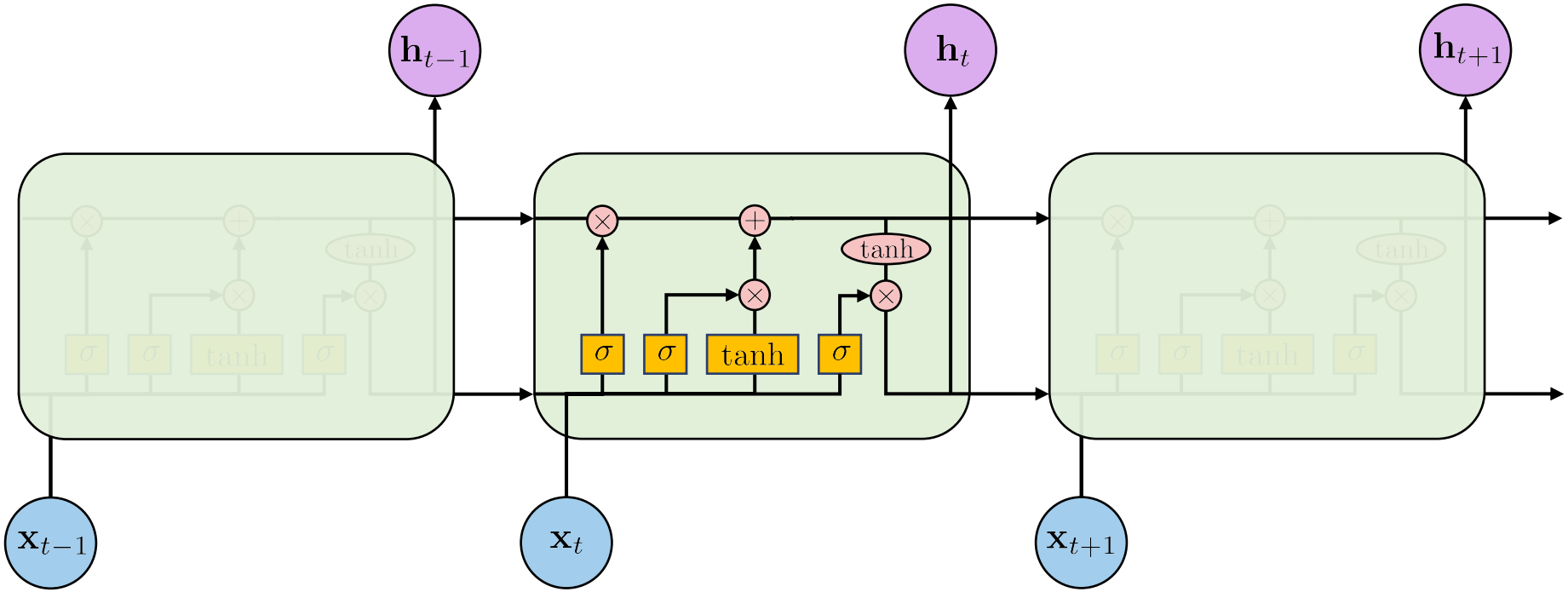


*reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN Architectures: LSTM

- Repeating modules in **LSTM**

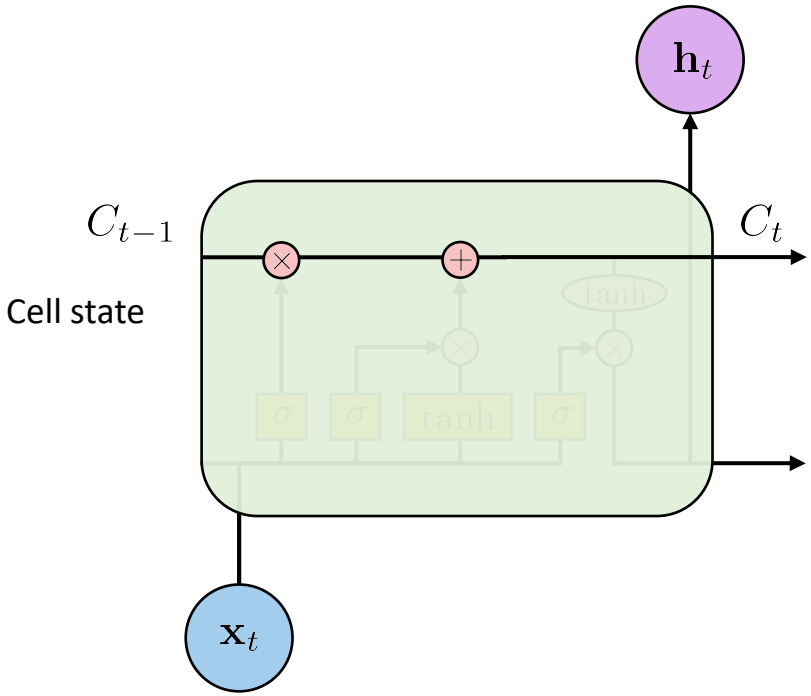


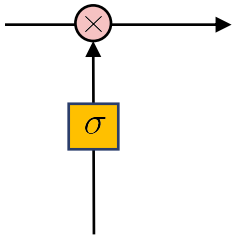


*reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN Architectures: LSTM

- The core idea behind LSTM
  - Able to control **how much information to preserve** from previous state
  - The horizontal line running through the top of the diagram (i.e., the **cell state** or **memory**)
    - Only **linear interactions** from the output of each "gates" (**prevent vanishing gradient**)
    - Control how much to remove or add information to the cell state





Gates : Way to optionally let information through

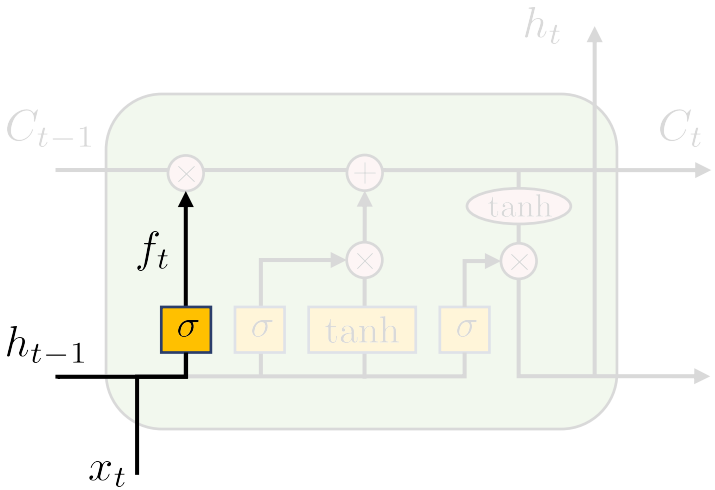
Next, LSTM step-by-step computation

*reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Step 1 : Decide what **information** we're going to **throw away** from the **cell state**

- A sigmoid layer called "**Forget gate**" f_t
- Looks at h_{t-1}, x_t and outputs a number between 0 and 1 for each cell state C_{t-1}
  - If 1: completely keep, if 0: completely remove

- e.g., language model trying to predict the next word based on all previous ones
  - The cell state might include the gender of the present subject
  - so that the correct pronouns can be used
  - When we see a new subject, we want to forget the gender of the old subject

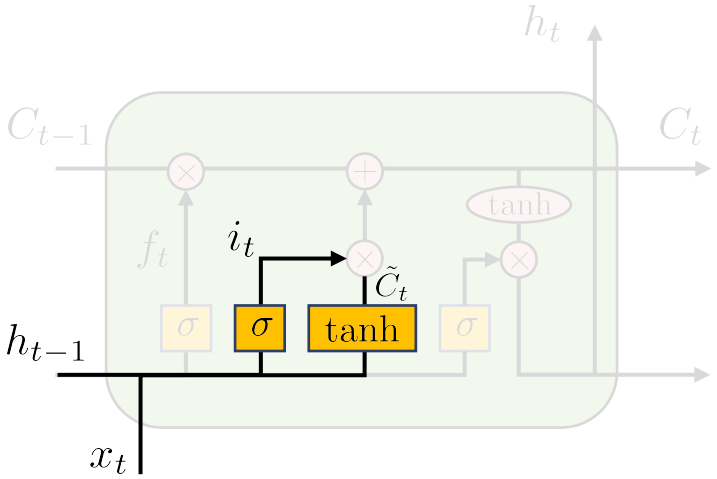


$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

*reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Step 2 : Decide what **information** we're going to **store** in the cell state and **update**

- First, a sigmoid layer called the "**Input gate**" i_t decides which values to update
- Next, a tanh layer creates a vector of new candidate values $\tilde{C}_t$



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

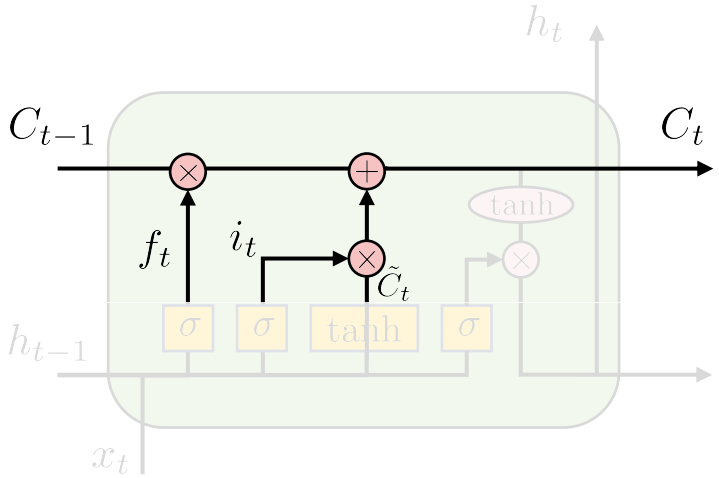
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

*reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Step 2 : Decide what **information** we're going to **store** in the cell state and **update**

- First, a sigmoid layer called the "**Input gate**" i_t decides which values to update
- Next, a tanh layer creates a vector of new candidate values $\tilde{C}_t$

- Then, update the old cell state C_{t-1} into the new cell state C_t
  - Multiply the old state by f_t
  - Add $i_t * \tilde{C}_t$, new candidate values scaled by how much to update



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

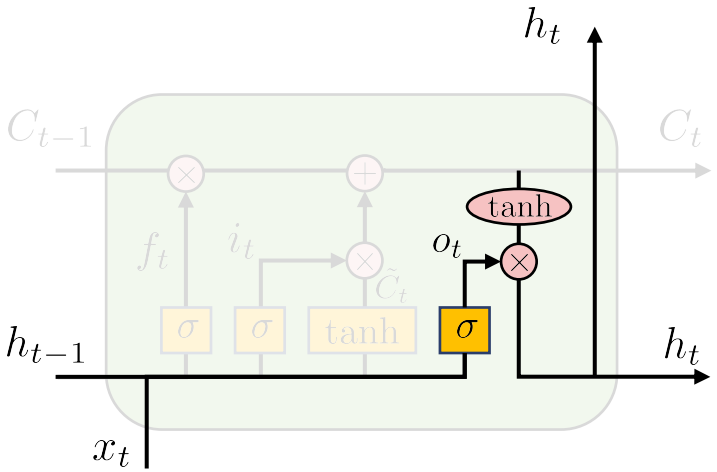
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

*reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Step 3 : Decide what we're going to **output**

- A sigmoid layer called "Output gate" o_t
- First go through o_t which decides **what parts** of the cell state **to output**
- Then, put the cell state C_t through tanh (push the values to be between -1 and 1) and multiply it by o_t

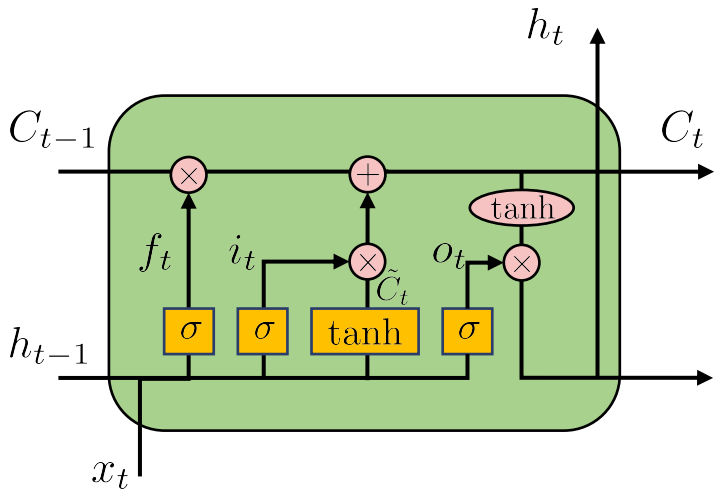


$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

*reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- Overall LSTM operations



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

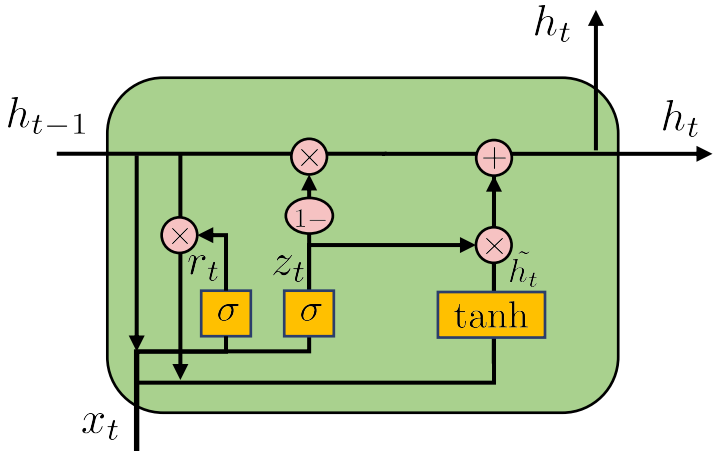
Standard LSTM

Next, Variants of LSTM

*reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN Architectures: GRU

- Gated Recurrent Unit (GRU) [Cho et.al, 2014]
  - Combines the forget and input gates into a single "**update gate**" z_t
    - Controls the **ratio of information to keep** between previous state and new state
  - **Reset gate** r_t controls how much information to forget
  - Merges the cell state C_t and hidden state h_t
  - (+) Resulting in **simpler model (less weights)** than standard LSTM



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h_t} = \tanh(W \cdot [r_t * h_{t-1}, x_t])$$

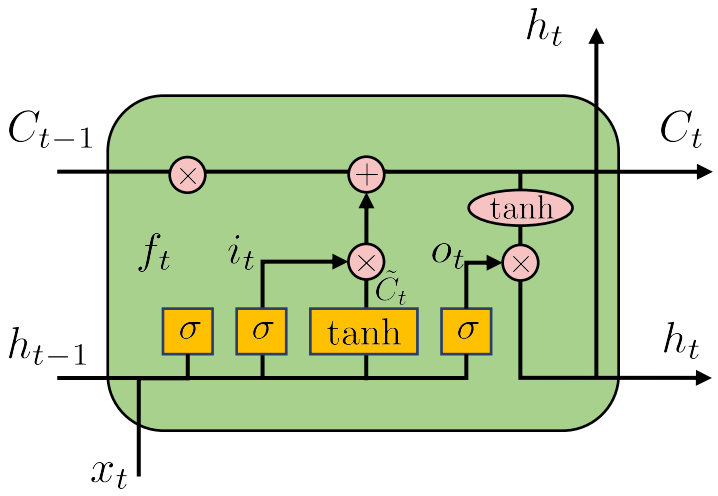
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h_t}$$

Gated Recurrent Unit

*reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

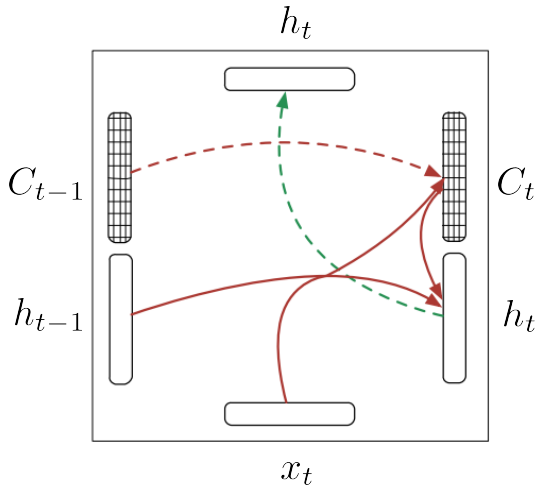
RNN Architectures: Stacked LSTM

- Standard LSTM



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

- With simplified diagram
  - C : cell state (or memory)
  - h : hidden state (or output)
  - Dashed line indicates identity transformation



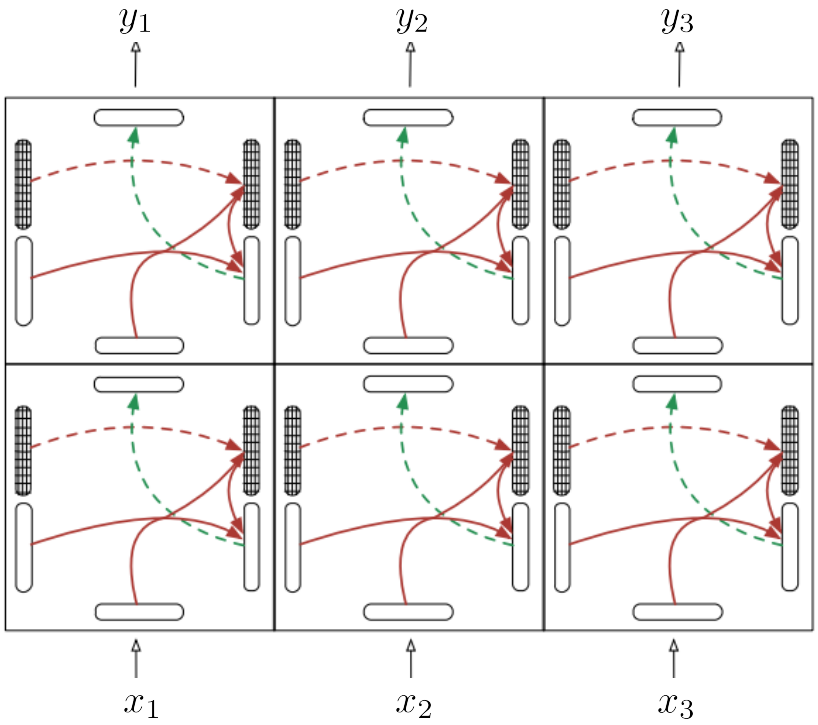
*source: https://arxiv.org/pdf/1507.01526.pdf

RNN Architectures: Stacked LSTM

- Stacked LSTM [Graves et al, 2013]
  - Adds capacity by simply stacking LSTM layers on top of each other
  - Output of 1st layer LSTM goes into 2nd layer LSTM as an input
  - But no vertical interactions

2nd layer LSTM

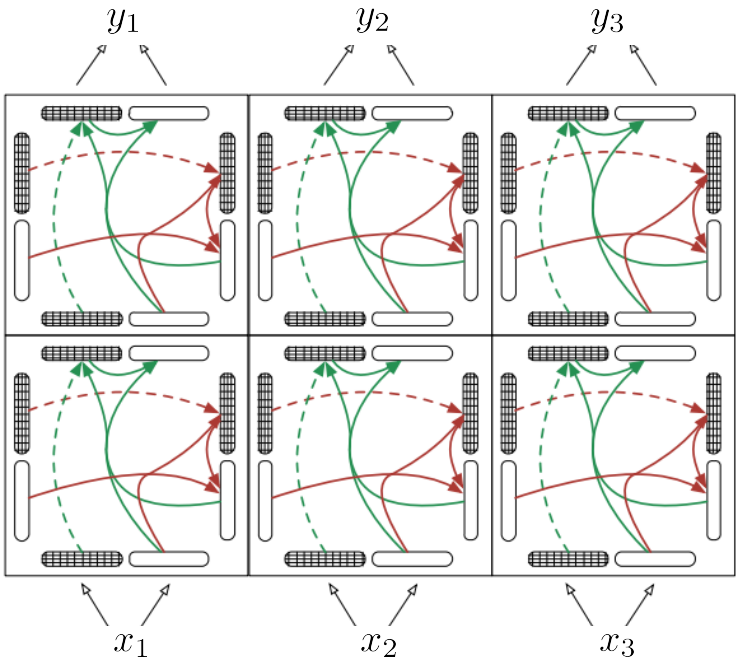
1st layer LSTM



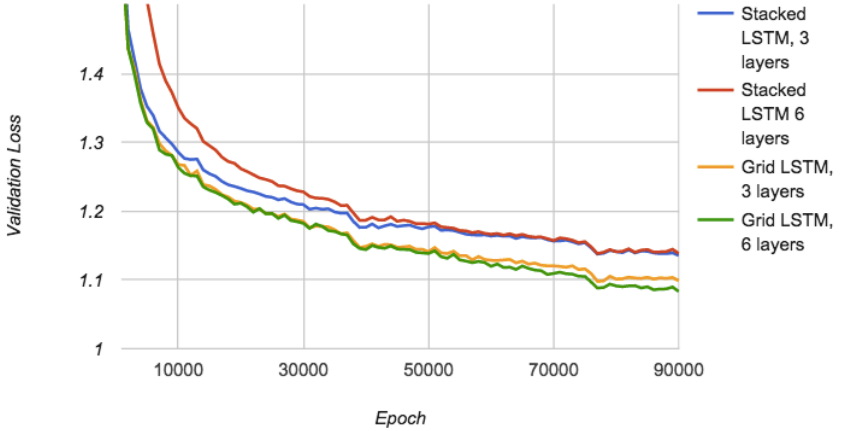
*source: https://arxiv.org/pdf/1507.01526.pdf

RNN Architectures: Grid LSTM

- Grid LSTM [Kalchbrenner et al., 2016]
  - Extended version of stacked LSTM
  - LSTM units have **memory** connections **along depth dimension** as well as temporal dimension



2D Grid LSTM



	BPC	Parameters	Alphabet Size	Test data
Stacked LSTM (Graves, 2013)	1.67	27M	205	last 4MB
MRNN (Sutskever et al., 2011)	1.60	4.9M	86	last 10MB
GFRNN (Chung et al., 2015)	1.58	20M	205	last 5MB
Tied 2-LSTM	**1.47**	16.8M	205	last 5MB

Performance on wikipedia dataset
(lower the better)

*source: https://github.com/coreylynch/grid-lstm

RNN Architectures: Limitation

- What is the limitation of all previous models?
  - They learn representations only from **previous** time steps
  - Useful to learn **future** time steps in order to
    - Better understand the **context**
    - Eliminate ambiguity

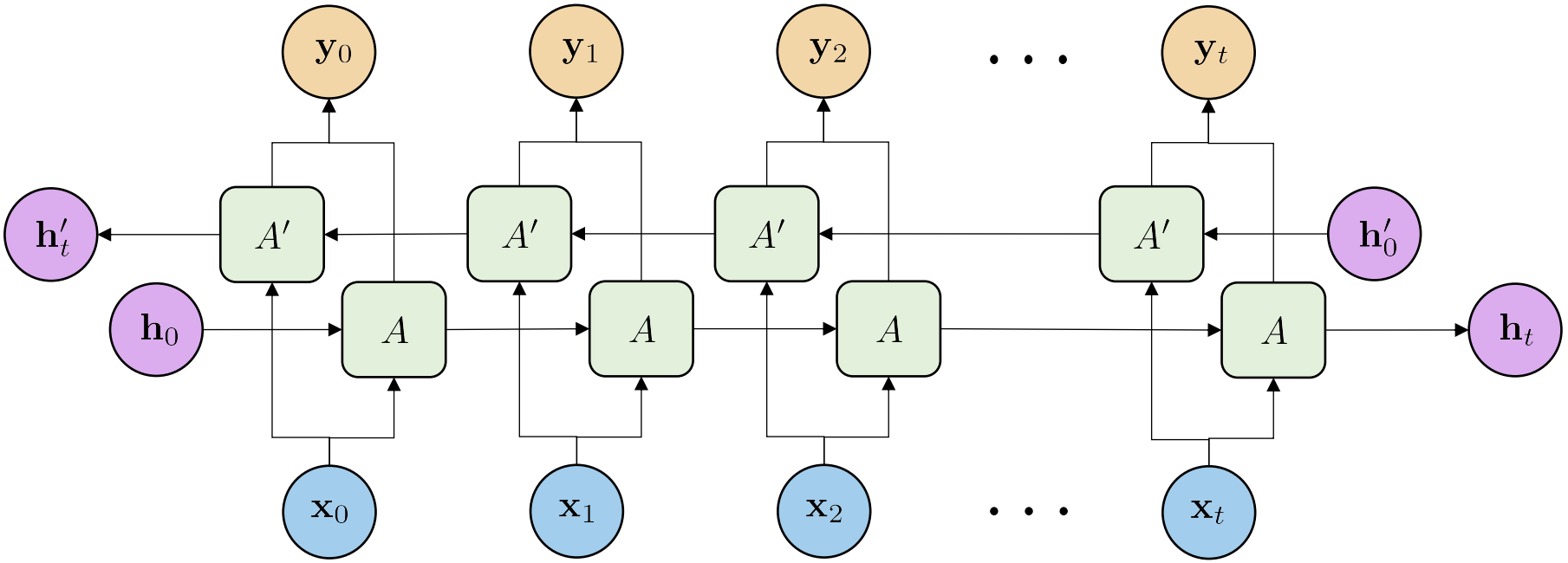
- Example
  - "**He said, Teddy** bears are on sale"
  - "**He said, Teddy** Roosevelt was a great President"
  - In above two sentences, only seeing previous words is not enough to understand the sentence

- Solution
  - Also look ahead ➔ Bidirectional RNNs

 *reference: https://towardsdatascience.com/introduction-to-sequence-models-rnn-bidirectional-rnn-lstm-gru-73927ec9df15

RNN Architectures: Bidirectional RNNs

- We can also extend RNNs into bi-directional models
  - The repeating blocks A could be **any types of RNNS** (Vanilla RNN, LSTM, or GRU)
  - The only difference is that there are additional paths from future time steps



Next, Comparison of Variants

 *reference: https://towardsdatascience.com/introduction-to-sequence-models-rnn-bidirectional-rnn-lstm-gru-73927ec9df15

RNN Architectures: Comparisons

- Which architecture is the best?
  - There is no clear winner; it **depends largely on the task**
  - More empirical exploration of RNNs can be found here [Jozefowicz et al., 2015]

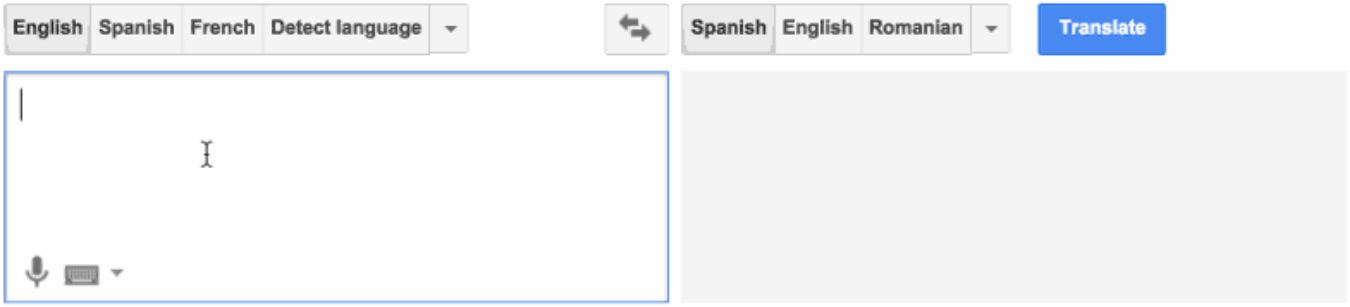
Model	Advantages	Disadvantages
LSTM	- Capable of modeling long-term sequential dependencies better than simple RNN - More robust to vanishing gradients than simple RNN	- Increases **computational complexity** - Higher **memory requirement** than RNN due to multiple memory cells
Stacked LSTM	- Models long-term sequential dependencies due to deeper architecture	- **Higher memory requirement** and **computational complexity** than LSTM due to stack of LSTM cells
Bidirectional LSTM	- Predicts both in the future and past context of the input sequence better than LSTM	- Increases **computational complexity** than LSTM
GRU	- Capable of modeling long-term sequential dependencies - Less memory requirements than LSTM	- Higher **computational complexity** and **memory requirements** than RNN due to multiple hidden state vectors
Grid LSTM	- Models multidimensional sequences with increased grid size	- **Higher memory requirement** and **computational complexity** than LSTM due to multiple recurrent connections

*source: https://arxiv.org/pdf/1801.01078.pdf

Table of Contents

Machine Translation

- What is machine translation?
  - Task of automatically **converting** source **text in one language to another** language
  - No single answer due to ambiguity/flexibility of human language (challenging)



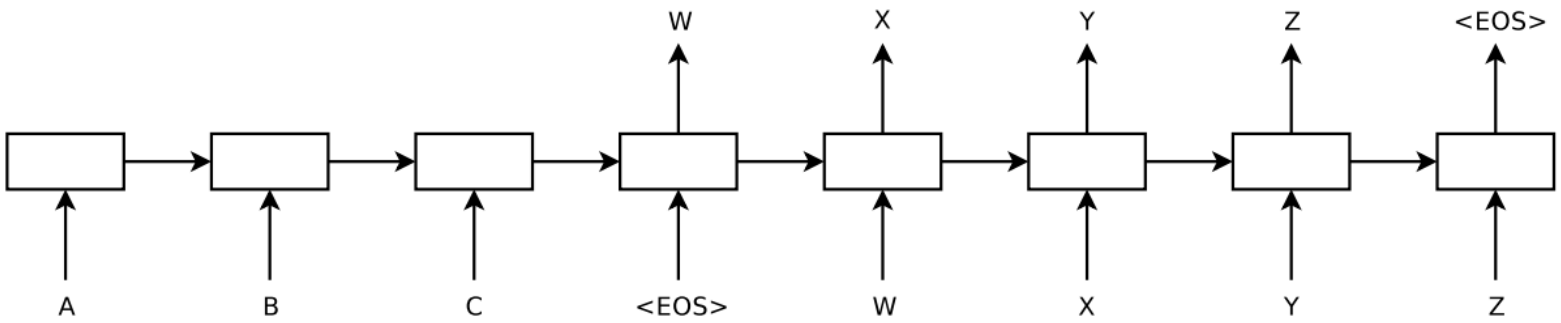
- Classical machine translation methods
  - Rule-based machine translation (RBMT)
  - Statistical machine translation (SMT; use of statistical model)

- Neural Machine Translation (NMT)
  - Use of **neural network models to learn a statistical model** for machine translation

Breakthroughs in NMT: Sequence-to-Sequence Learning

- Difficulties in Neural Machine Translation
  - Intrinsic difficulties of MT (ambiguity of language)
  - Variable length of input and output sequence (difficult to learn a single model)

- The core idea of **sequence-to-sequence** model [Sutskever et al., 2014]
  - **Encoder-Decoder** architecture (input → vector → output)
  - Use one RNN network (Encoder) to **read input sequence** at a time to obtain large **fixed-length vector representation**
  - Use another RNN (Decoder) to extract the output sequence from that vector



Input sequence “ABC” and output sequence “WXYZ”

Breakthroughs in NMT: Sequence-to-Sequence Learning

- Encoder
  - Reads the input sentence, a sequence of vectors $\mathbf{x} = (x_1, \ldots, x_T)$ into a vector c
  - Use RNNs such that $h_t = f(x_t, h_{t-1})$ and $c = q(\{h_1, \ldots, h_T\})$, where f and q are some non-linear functions
  - LSTMs as f and $q(\{h_1, \ldots, h_T\}) = h_T$ (in the original seq2seq model)

- Decoder
  - Trained to predict the next word $y_{t'}$ given the context vector c and the previously predicted words $\{y_1, \ldots, y_{t'-1}\}$
  - Defines a probability over the translation $\mathbf{y}$ by decomposing the joint probability into the ordered conditionals:

$$p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t | \{y_1, \ldots, y_{t'-1}\}, c),$$

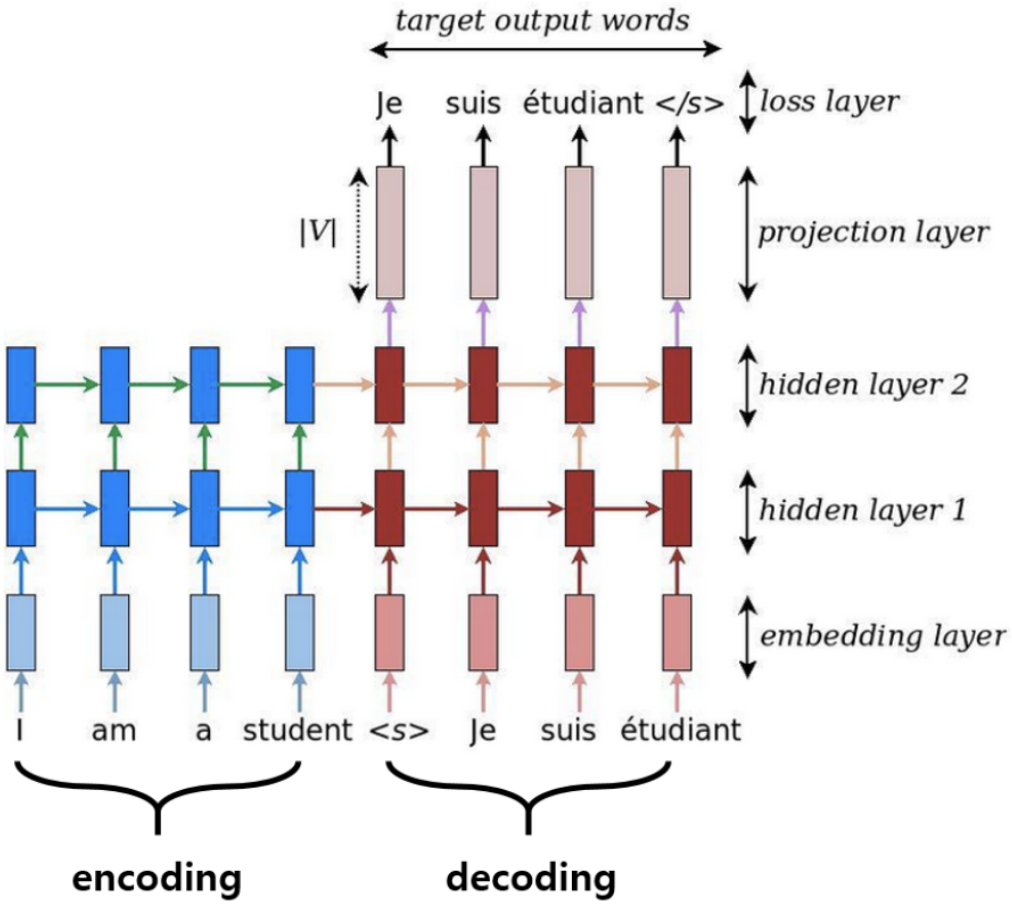
    where $\mathbf{y} = (y_1, \ldots, y_T)$.
  - The conditional probability is modeled as

$$p(y_t | \{y_1, \ldots, y_{t'-1}\}, c) = g(y_{t-1}, s_t, c),$$

    where g is a nonlinear, potentially multi-layered function that outputs the probability of y_t and s_t is the hidden state of the RNN

Breakthroughs in NMT: Sequence-to-Sequence Learning

- Example of the seq2seq model
  - For English → French task
  - With 2-layer LSTM for encoder and encoder



*source: https://towardsdatascience.com/seq2seq-model-in-tensorflow-ec0c557e560f

Breakthroughs in NMT: Sequence-to-Sequence Learning

- Results on WMT'14 English to French dataset
  - Measure : BLEU(Bilingual Evaluation Understudy) score
    - Widely used quantitative measure for MT task
  - On par with the state-of-the-art SMT system (without any neural network)
  - Achieved better results than the previous baselines

Method	test BLEU score (ntst14)
Baseline System [29]	33.30
Cho et al. [5]	34.54
State of the art [9]	**37.0**
Rescoring the baseline 1000-best with a single forward LSTM	35.61
Rescoring the baseline 1000-best with a single reversed LSTM	35.85
Rescoring the baseline 1000-best with an ensemble of 5 reversed LSTMs	**36.5**
Oracle Rescoring of the Baseline 1000-best lists	~45

- Simple but very powerful in MT task

Next, Seq2seq with attention

*source: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning

Breakthroughs in NMT: Joint Learning to Align and Translate

- NMT by Joint Learning to Align and Translate [Bahdanau et al., 2015]

- Problem of original encoder-decoder (or seq2seq) model
  - Need to **compress** all the necessary information of a source sentence into a **fixed-length vector**
  - Very **difficult** **to cope with long sentences**, especially when the test sequence is longer than the sentences in the training corpus

- Extension of encoder-decoder model + **attention** mechanism
  - Encode input sentence into a sequence of vectors
  - And **chooses a subset of these vectors adaptively** while decoding the translation
  - Frees the neural network model from having to squash all the information into a single fixed-length vector

Breakthroughs in NMT: Joint Learning to Align and Translate

- Define each conditional probability as:

$$p(y_i|\{y_1,\ldots,y_{i-1}\},\mathbf{x}) = g(y_{i-1},s_i,c_i),$$

where s_i is an RNN hidden state for time i computed by $s_i = f(s_{i-1},y_{i-1},c_i)$.

- Distinct context vector c_i for each target word y_i
- The context vector c_i is computed as weighted sum of h_i

$$c_i = \sum_{j=1}^{T} \alpha_{ij} h_j.$$

- The weight α_{ij} of each h_j is computed by

$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T} exp(e_{ik})},$$

where $e_{ij} = a(s_{i-1},h_j)$ is an alignment model which scores **how well the inputs** around position j and the **output** position i **match**.

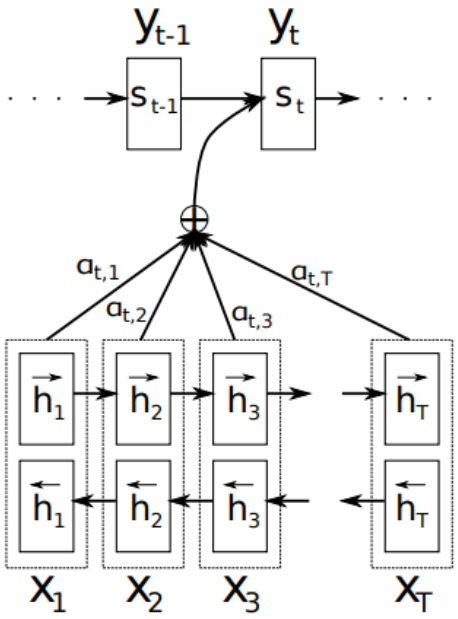
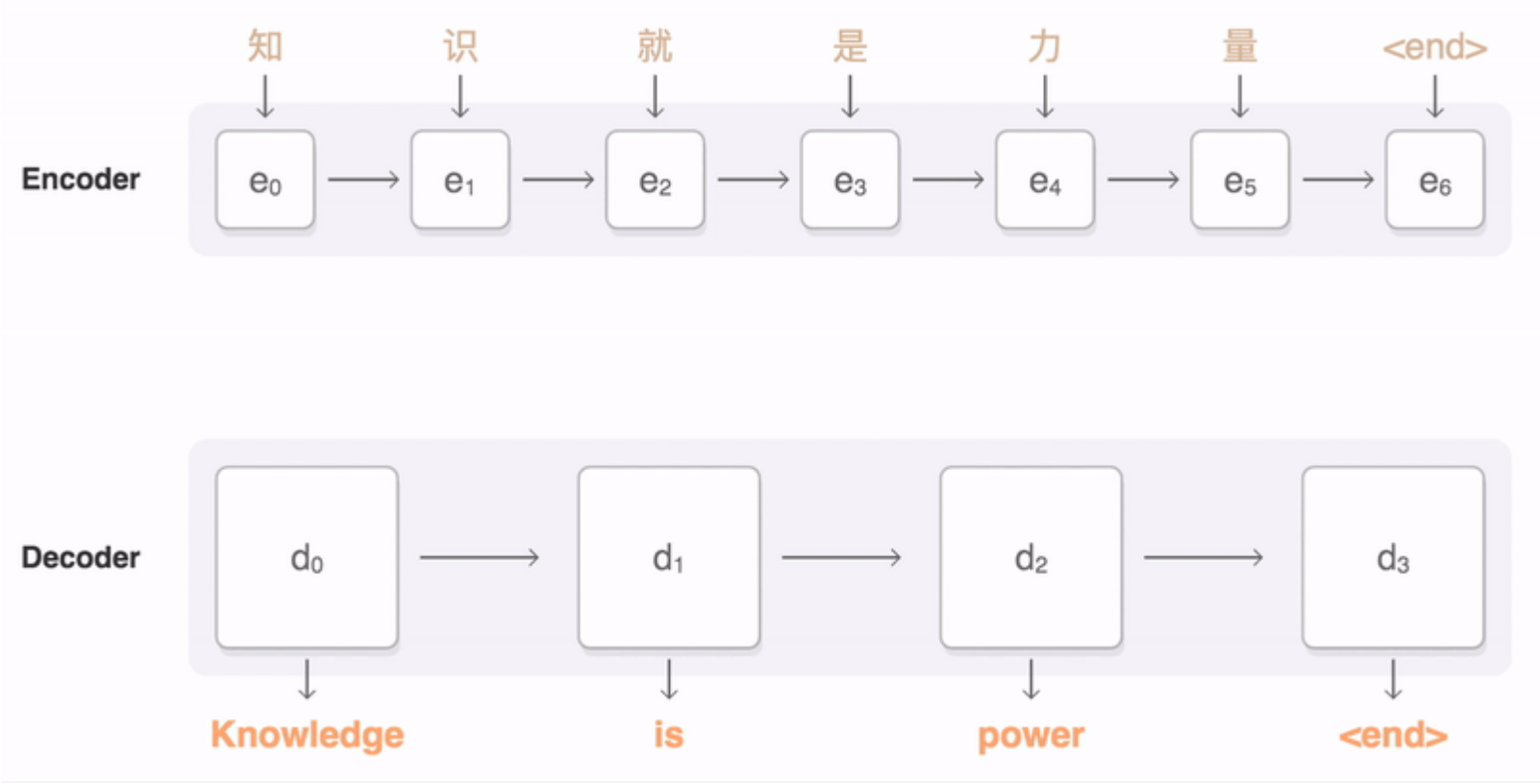


Illustration of the model

Breakthroughs in NMT: Joint Learning to Align and Translate

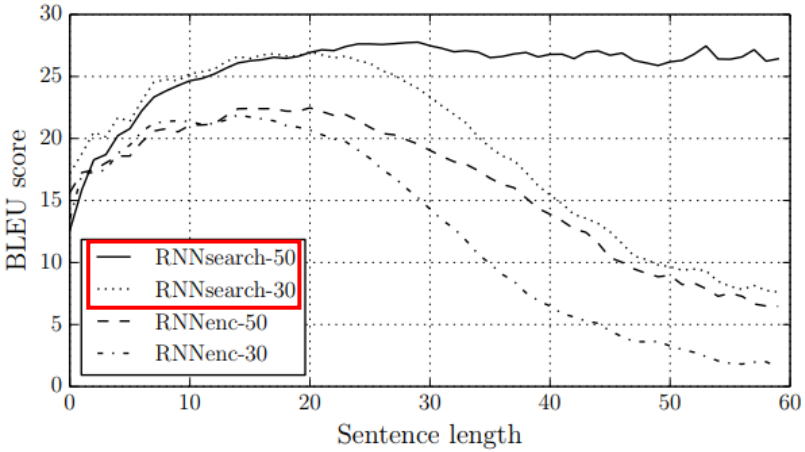
- Graphical illustration of seq2seq with attention
  - e.g., Chinese to English

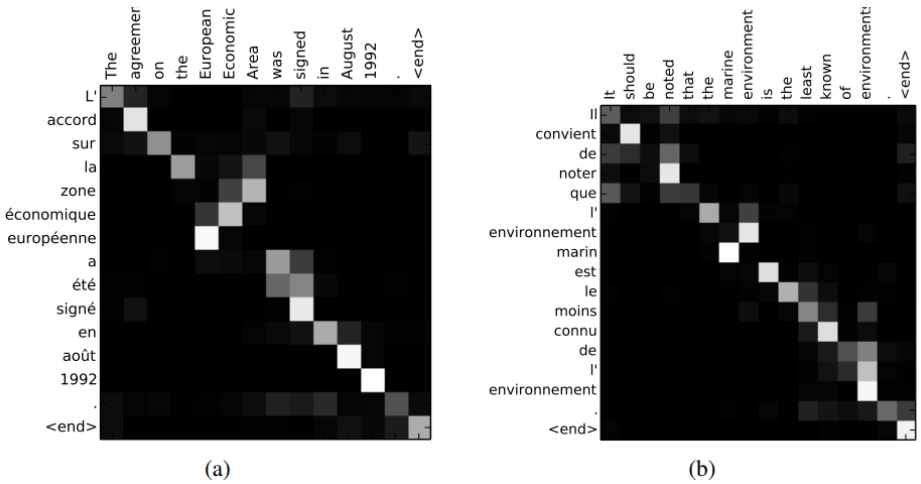


*source: https://google.github.io/seq2seq/

Breakthroughs in NMT: Joint Learning to Align and Translate

- Results
  - RNNsearch (proposed) is better than RNNenc (vanilla seq2seq)
  - RNNsearch-50: model trained with sentences of length up to 50 words



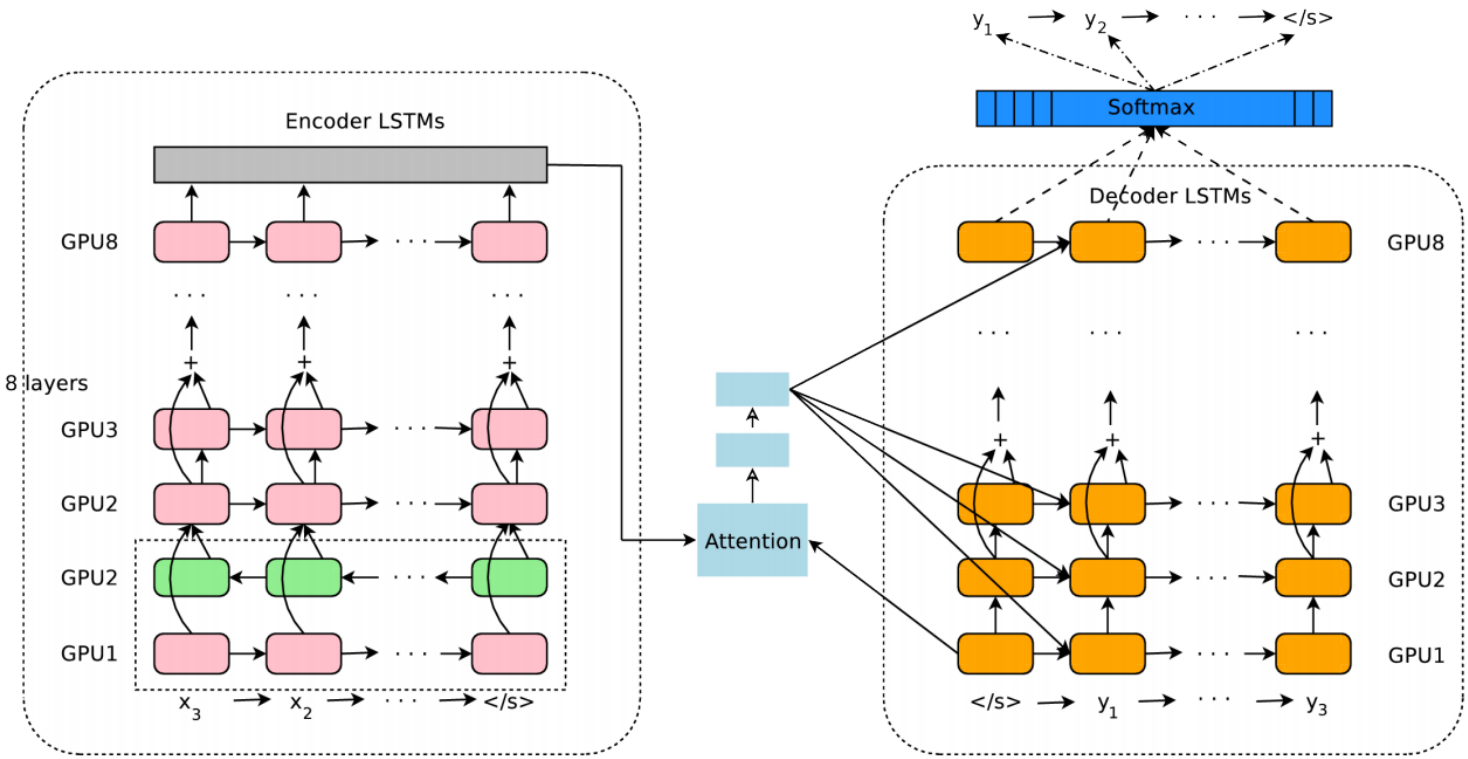


Sample alignment results

Next, Google's NMT

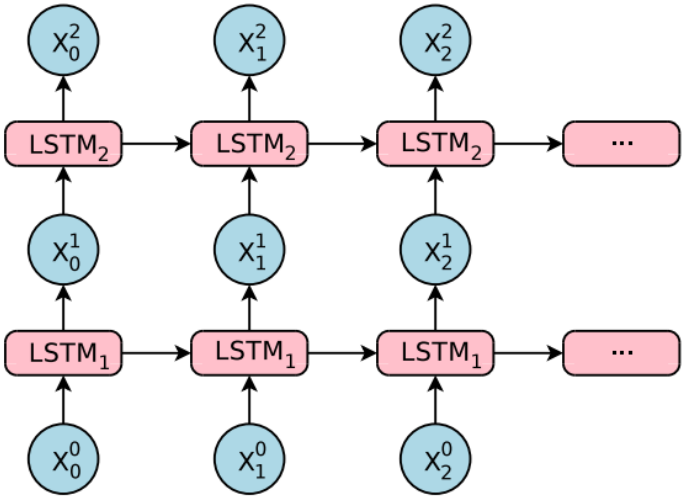
Google's Neural Machine Translation (GNMT)

- Google's NMT [Wu et al., 2016]
  - Improves over previous NMT systems on **accuracy** and **speed**
  - 8-layer LSTMS for encoder/decoder with attention
  - Achieve **model parallelism** by assigning each LSTM layer into different GPUs
  - Add **residual connections in standard LSTM**
  - … and lots of domain-specific details to apply it to production model

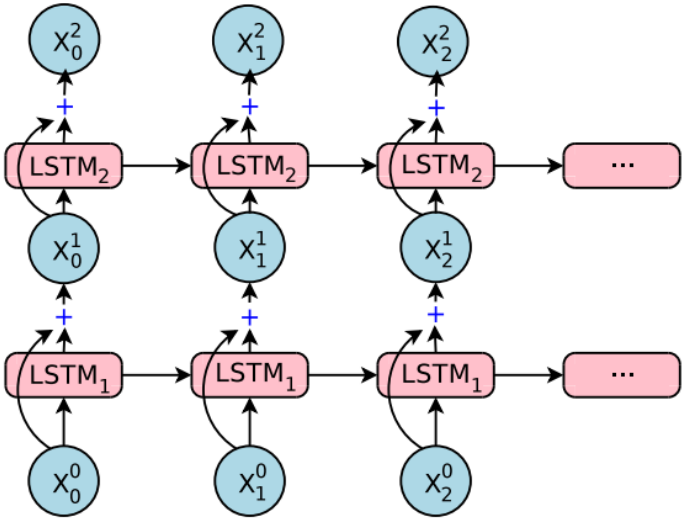


Google's Neural Machine Translation (GNMT)

- Adding **residuals connections** in stacked LSTM
  - In practice, LSTM has also problem of **vanishing gradient** when stacking more layers
  - Empirically, 4-layer works okay, 6-layer has problem, 8-layer does not work at all
  - Apply residual connections with 8-layer stacked LSTM worked best



Standard stacked LSTM



Stacked LSTM with residual connections

Google's Neural Machine Translation (GNMT)

- Results
  - State-of-the-art results on various MT datasets
  - Also comparable with Human expert

Table 5: Single model results on WMT En→De (newstest2014)

Model	BLEU	CPU decoding time per sentence (s)
Word	23.12	0.2972
Character (512 nodes)	22.62	0.8011
WPM-8K	23.50	0.2079
WPM-16K	24.36	0.1931
WPM-32K	24.61	0.1882
Mixed Word/Character	24.17	0.3268
PBMT [6]	20.7	
RNNSearch [37]	16.5	
RNNSearch-LV [37]	16.9	
RNNSearch-LV [37]	16.9	
Deep-Att [45]	20.6	

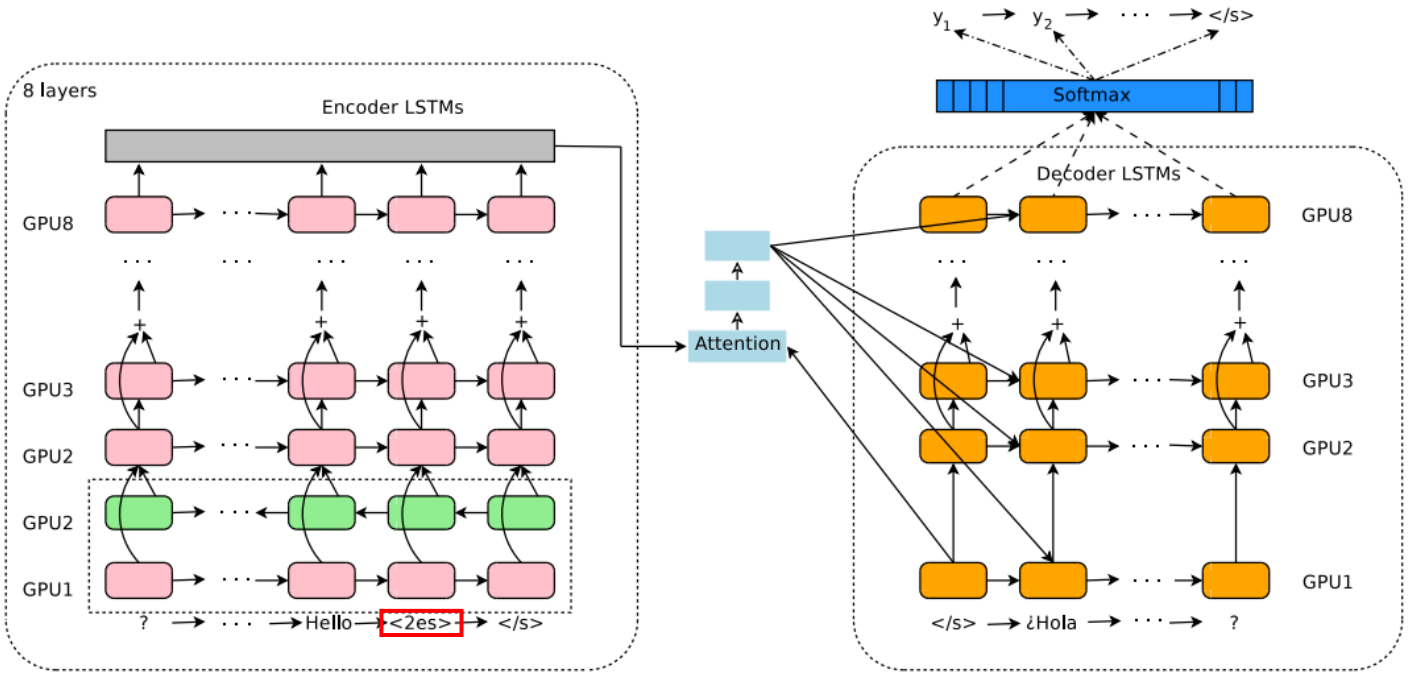
GNMT with different configurations

Table 10: Mean of side-by-side scores on production data

	PBMT	GNMT	Human	Relative Improvement
English → Spanish	4.885	5.428	5.504	87%
English → French	4.932	5.295	5.496	64%
English → Chinese	4.035	4.594	4.987	58%
Spanish → English	4.872	5.187	5.372	63%
French → English	5.046	5.343	5.404	83%
Chinese → English	3.694	4.263	4.636	60%

Google's Multilingual Neural Machine Translation (Multilingual GNMT)

- Further improved in [Johnson et al., 2016]
- Extensions to make this model to be Multilingual NMT system by adding **artificial token** to indicate the required **target language**
  - e.g., the token "<2es>" indicates that the target sentence is in Spanish
  - Can do **multilingual NMT** using a **single model** w/o increasing the parameters



Next, Transformer (self-attention)

Transformer (Self-attention)

- Motivation:
  - Prior works use RNN/CNN to solve **sequence-to-sequence** problems
  - **Attention** already handles *arbitrary length* of sequences, *easy to parallelize*, and not suffer from *forgetting* problems… Why should one use RNN/CNN modules?

- Idea:
  - Design architecture **only using** attention modules
  - To extract features, the authors use self-attention, that features **attend on itself**
    - Self-attention has many advantages over RNN/CNN blocks

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

n: sequence length, d: feature dimension, k: (conv) kernel size, r: window size to consider
Maximum path length: maximum traversal between any two input/outputs **(lower is better)**

*Cf. Now self-attention is widely used in other architectures, e.g., CNN [Wang et al., 2018] or GAN [Zhang et al., 2018]

Transformer (Self-attention)

- Multi-head attention: The building block of the Transformer
  - In previous slide, we introduced ***additive* attention** [Bahdanau et al., 2015]
  - Here, the context vector is a **linear combination** of
    - weight $\alpha_{t,i}$, a function of inputs $[x_i]$ and output y_t
    - and input hidden states $[h_i]$

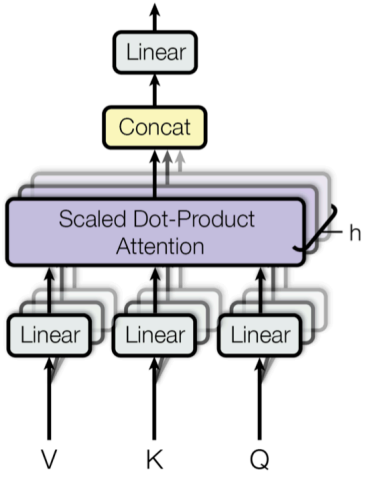
$$c_t = \sum_{i=1}^{n} \alpha_{t,i} h_i$$

  - In general, attention is a function of **key** K, **value** V, and **query** Q
    - **key** $[x_i]$ and **query** y_t defines weights $\alpha_{t,i}$, which are applied to **value** $[h_i]$
    - For sequence length T and feature dimension d, (K, V, Q) are $T \times d$, $T \times d$, and $1 \times d$ matrices

  - Transformer use ***scaled dot-product* attention**

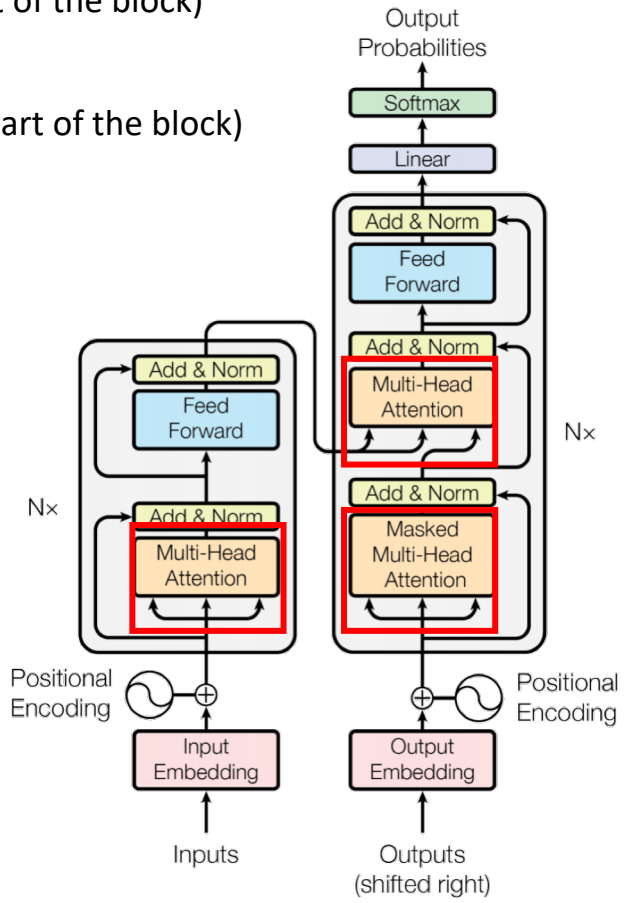
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

  - In addition, transformer use **multi-head attention**, ***ensemble*** of attentions



Transformer (Self-attention)

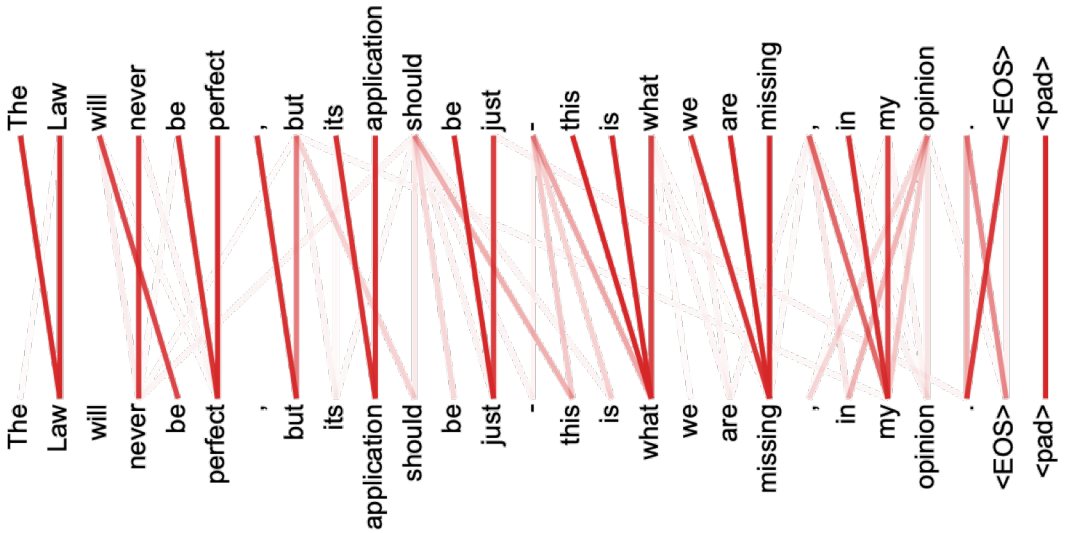
- Transformer:
  - The final **transformer** model is built upon the (multi-head) attention blocks
  - First, extract features with **self-attention** (see lower part of the block)
  - Then decode feature with usual **attention** (see middle part of the block)
  - Since the model don't have a sequential structure, the authors give **position embedding** (some handcrafted feature that represents the location in sequence)



Transformer (Self-attention)

- Results: Transformer architecture shows good performance for languages

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

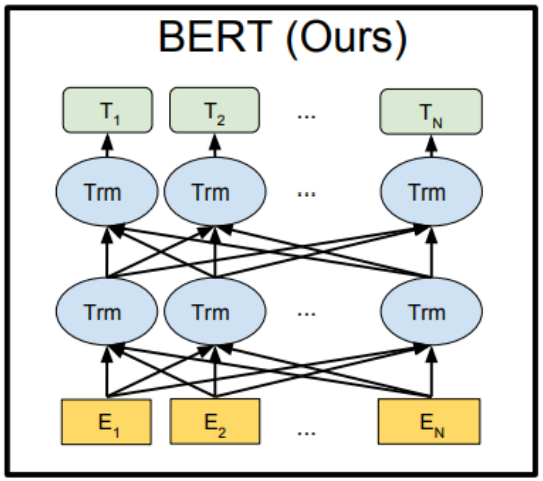


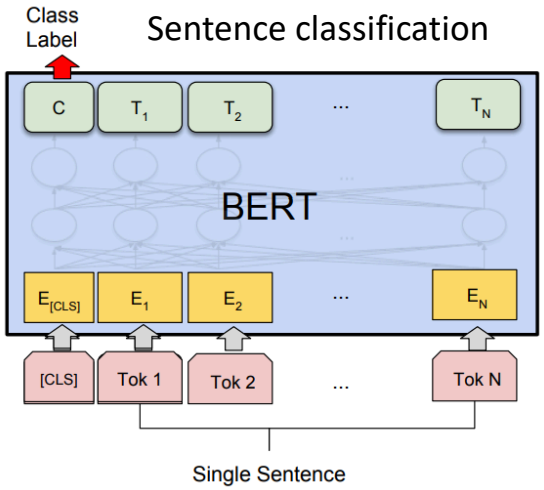
Next, BERT

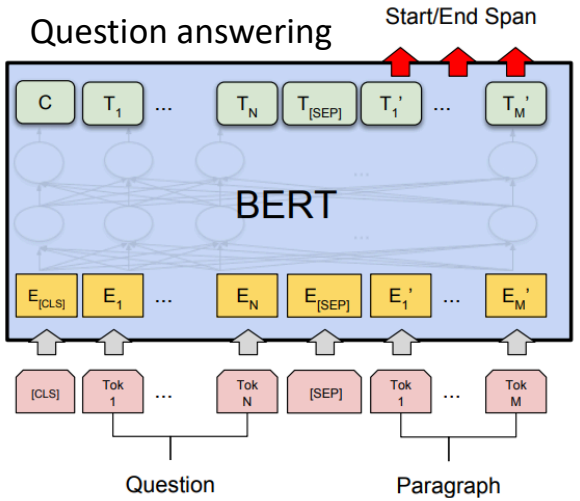
BERT (Bidirectional Encoder Representations from Transformers)

- Motivation:
  - Many **success of CNN** comes from ImageNet-pretrained networks
  - Can train a universal encoder for natural languages?

- Method:
  - Pretrain a bidirectional transformer with two self-supervised tasks
    - Tasks: masked language model, next sentence prediction
    - Cf. While BERT is encoder-only, one can also train encoder-decoder (e.g., GPT-2)
  - Use fixed BERT encoder, and fine-tune simple 1-layer decoder for each task







BERT (Bidirectional Encoder Representations from Transformers)

- Results:
  - Even without task-specific complex architectures, BERT achieves **SOTA** for **11 NLP tasks**, including classification, question answering, tagging, etc.

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

System	Dev F1	Test F1
ELMo+BiLSTM+CRF	95.7	92.2
CVT+Multi (Clark et al., 2018)	-	92.6
$\text{BERT}_{\text{BASE}}$	96.4	92.4
$\text{BERT}_{\text{LARGE}}$	**96.6**	**92.8**

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
$\text{BERT}_{\text{BASE}}$	81.6	-
$\text{BERT}_{\text{LARGE}}$	**86.6**	**86.3**
Human (expert)†	-	85.0
Human (5 annotations)†	-	88.0

Table of Contents

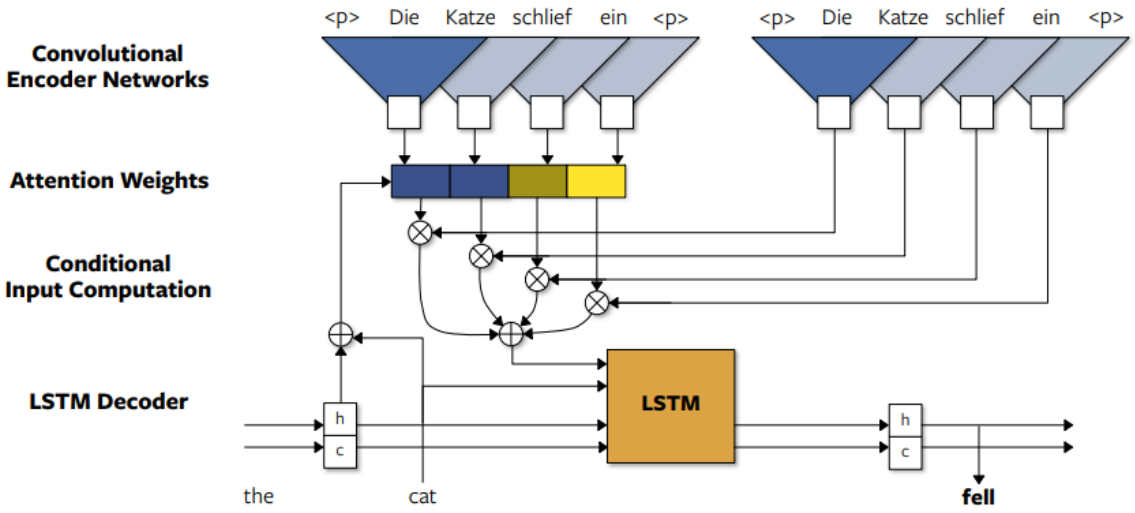
Problem in RNNs

- The most fundamental problem of RNNs
  - Require **heavy computations (slow)**
  - Especially when we stack multiple layers
  - GNMT solved this by model parallelism

- How to alleviate this issue in terms of architectures?
  - CNN encoder
  - CNN encoder + decoder
  - Optimizing RNNs (pruning approach)

Solution 1-1. CNN encoder

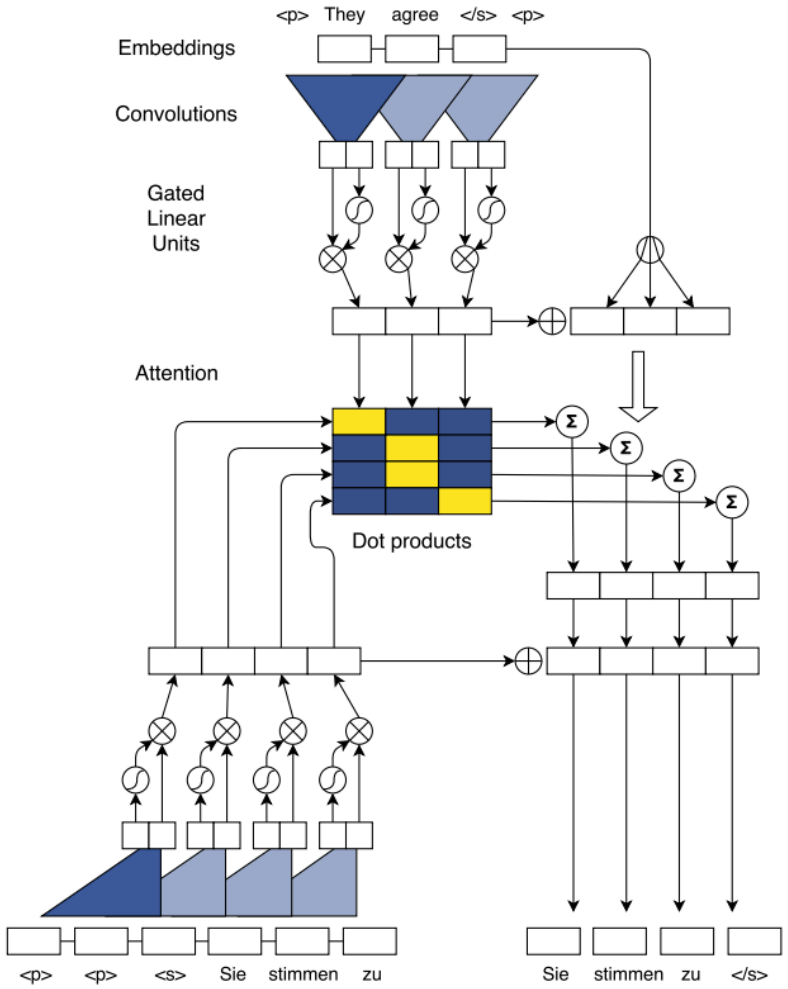
- A convolutional encoder model for Neural Machine Translation [Gehring et al., 2016]
  - CNN encoder + RNN decoder
  - Replace the RNN encoder with stack of 1-D convolutions with nonlinearities
    - Two different CNNs for attention score computation and conditional input aggregation
    - More parallelizable than using RNN



Encoder	Words/s	BLEU
2-layer BiLSTM	109.9	23.6
Deep Conv. 8/4	231.1	23.7
Deep Conv. 15/5	203.3	24.0

Solution 1-2. CNN encoder + decoder

- Convolutional sequence to sequence learning [Gehring et al., 2017]
  - CNNs for both encoder and encoder



WMT'14 English-French	BLEU
Wu et al. (2016) GNMT (Word 80K)	37.90
Wu et al. (2016) GNMT (Word pieces)	38.95
Wu et al. (2016) GNMT (Word pieces) + RL	39.92
ConvS2S (BPE 40K)	40.51

Performance

	BLEU	Time (s)
GNMT GPU (K80)	31.20	3,028
GNMT CPU 88 cores	31.20	1,322
GNMT TPU	31.21	384
ConvS2S GPU (K40) $b = 1$	33.45	327
ConvS2S GPU (M40) $b = 1$	33.45	221
ConvS2S GPU (GTX-1080ti) $b = 1$	33.45	142
ConvS2S CPU 48 cores $b = 1$	33.45	142
ConvS2S GPU (K40) $b = 5$	34.10	587
ConvS2S CPU 48 cores $b = 5$	34.10	482
ConvS2S GPU (M40) $b = 5$	34.10	406
ConvS2S GPU (GTX-1080ti) $b = 5$	34.10	256

Generation speed

Solution 2. Optimizing RNNs

- Exploring Sparsity in Recurrent Neural Networks [Narang et al., 2017]
  - **Pruning RNNs** to improve **inference time** with marginal performance drop
    - Simple heuristics to calculate the threshold
    - And apply that threshold to every binary mask corresponds to each weight
  - Reduces the size of the model by 90%
  - Significant inference time speed-up using sparse matrix multiply around $2\times$ to $7\times$

MODEL	# UNITS	CER	# PARAMS	RELATIVE PERF
RNN Dense Baseline	1760	10.67	67 million	0.0%
RNN Dense Small	704	14.50	11.6 million	-35.89%
RNN Dense Medium	2560	9.43	141 million	11.85%
RNN Sparse 1760	1760	12.88	8.3 million	-20.71%
RNN Sparse Medium	2560	**10.59**	11.1 million	0.75%
RNN Sparse Big	3072	**10.25**	16.7 million	3.95%
GRU Dense	2560	9.55	115 million	0.0%
GRU Sparse	2560	10.87	13 million	-13.82%
GRU Sparse Medium	3568	**9.76**	17.8 million	-2.20%

LAYER SIZE	SPARSITY	LAYER TYPE	TIME (μsec)	SPEEDUP
1760	0%	RNN	56	1
1760	95%	RNN	20	2.8
2560	95%	RNN	29	1.93
3072	95%	RNN	48	1.16
2560	0%	GRU	313	1
2560	95%	GRU	46	6.80
3568	95%	GRU	89	3.5

GEMM (General Matrix-Matrix Multiply) times comparison

Summary

- RNN architectures have developed in a way that
  - Can better model **long-term dependency**
  - **Robust** to vanishing gradient problems
  - While having **less memory or computational costs**

- Breakthroughs in machine translation
  - Seq2seq model with attention
  - Transformer with self-attention (and BERT)

- Alleviating the problem of RNNs' heavy computations
  - Convolutional sequence to sequence learning
  - Pruning approach

- There are various applications combining RNNs with other networks
  - Image caption generation, visual question answering(VQA), etc.
  - Will be covered in later lectures

References

[Hochreiter and Schmidhuber, 1997] "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.
link: http://www.bioinf.jku.at/publications/older/2604.pdf

[Graves et al., 2005] "Framewise phoneme classification with bidirectional LSTM and other neural network architectures." *Neural Networks* 18.5-6 (2005): 602-610.
Link: ftp://ftp.idsia.ch/pub/juergen/nn_2005.pdf

[Graves et al, 2013] "Speech recognition with deep recurrent neural networks." *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 2013.
Link: https://www.cs.toronto.edu/~graves/icassp_2013.pdf

[Cho et al., 2014] "Learning phrase representations using RNN encoder-decoder for statistical machine translation." *arXiv preprint arXiv:1406.1078* (2014).
Link: https://arxiv.org/pdf/1406.1078v3.pdf

[Sutskever et al., 2014] "Sequence to sequence learning with neural networks." *NIPS* 2014.
link : http://papers.nips.cc/paper/5346-sequence-to-sequence-learnin

[Sutskever et al., 2014] "Sequence to sequence learning with neural networks.“ NIPS 2014.

[Bahdanau et al., 2015] “"Neural machine translation by jointly learning to align and translate.“, ICLR 2015
Link: https://arxiv.org/pdf/1409.0473.pdf

[Jozefowicz et al., 2015] "An empirical exploration of recurrent network architectures." *ICML* 2015.
Link: http://proceedings.mlr.press/v37/jozefowicz15.pdf

[Bahdanau et al., 2015] Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." *ICLR 2015*
link : https://arxiv.org/pdf/1409.0473.pdf

References

[Kalchbrenner et al., 2016] "Grid long short-term memory." *ICLR 2016*
Link: https://arxiv.org/pdf/1507.01526.pdf

[Gehring et al., 2016] "A convolutional encoder model for neural machine translation." *arXiv preprint arXiv:1611.02344* (2016).
Link: https://arxiv.org/pdf/1611.02344.pdf

[Wu et al., 2016] "Google's neural machine translation system: Bridging the gap between human and machine translation." *arXiv preprint arXiv:1609.08144* (2016).
link: https://arxiv.org/pdf/1609.08144.pdf

[Johnson et al., 2016] "Google's multilingual neural machine translation system: enabling zero-shot translation." *arXiv preprint arXiv:1611.04558* (2016).
Link: https://arxiv.org/pdf/1611.04558.pdf

[Gehring et al., 2017] "Convolutional sequence to sequence learning." *arXiv preprint arXiv:1705.03122* (2017).
Link: https://arxiv.org/pdf/1705.03122.pdf

[Narang et al., 2017] "Exploring sparsity in recurrent neural networks.“, ICLR 2017
Link: https://arxiv.org/pdf/1704.05119.pdf

[Fei-Fei and Karpathy, 2017] “CS231n: Convolutional Neural Networks for Visual Recognition”*, 2017*. (Stanford University)
link : http://cs231n.stanford.edu/2017/

[Salehinejad et al., 2017] "Recent Advances in Recurrent Neural Networks." *arXiv preprint arXiv:1801.01078* (2017).
Link: https://arxiv.org/pdf/1801.01078.pdf