Sequence Modeling: Recurrent and Recursive Nets

Jie Tang

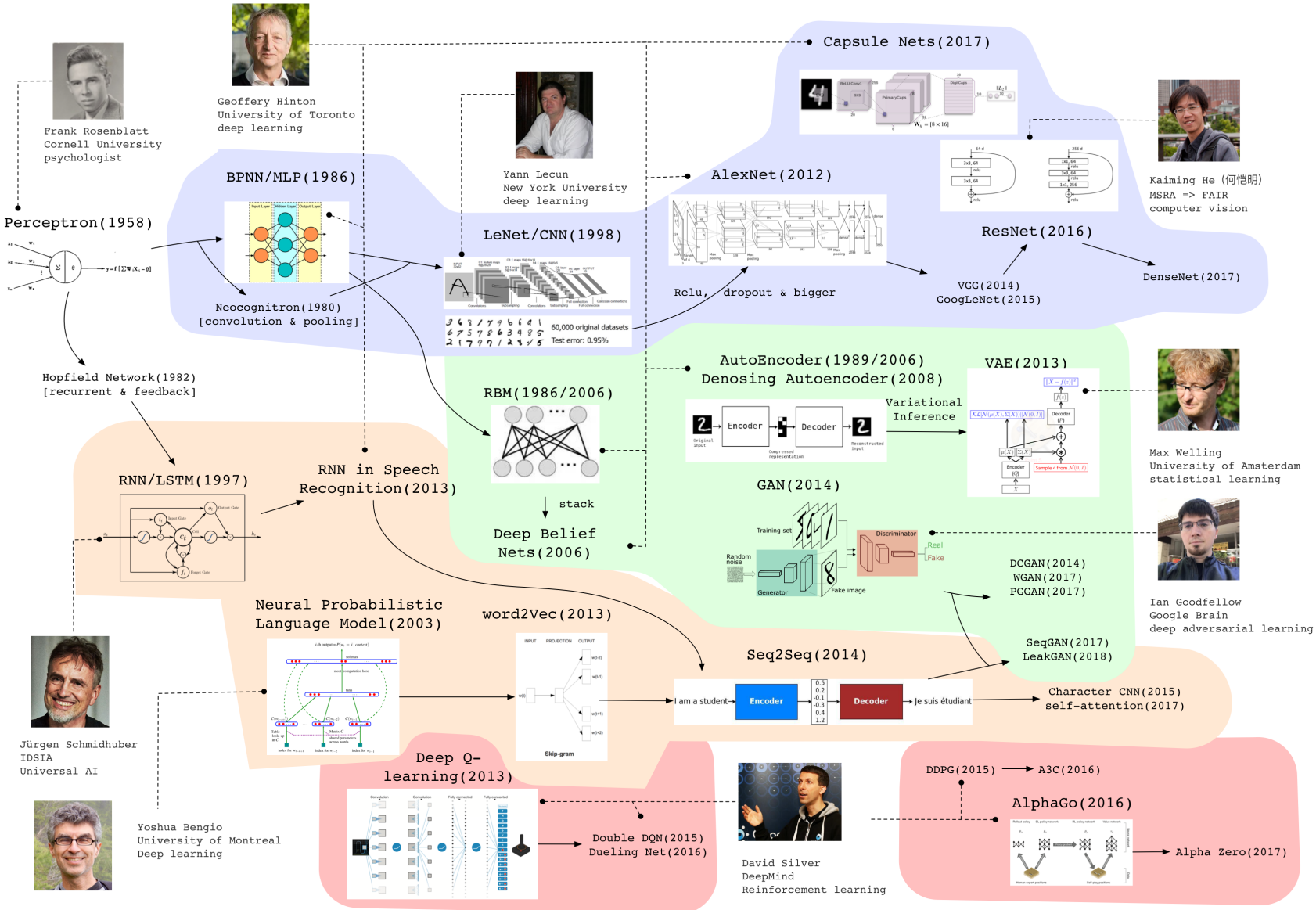
Tsinghua University

Overview

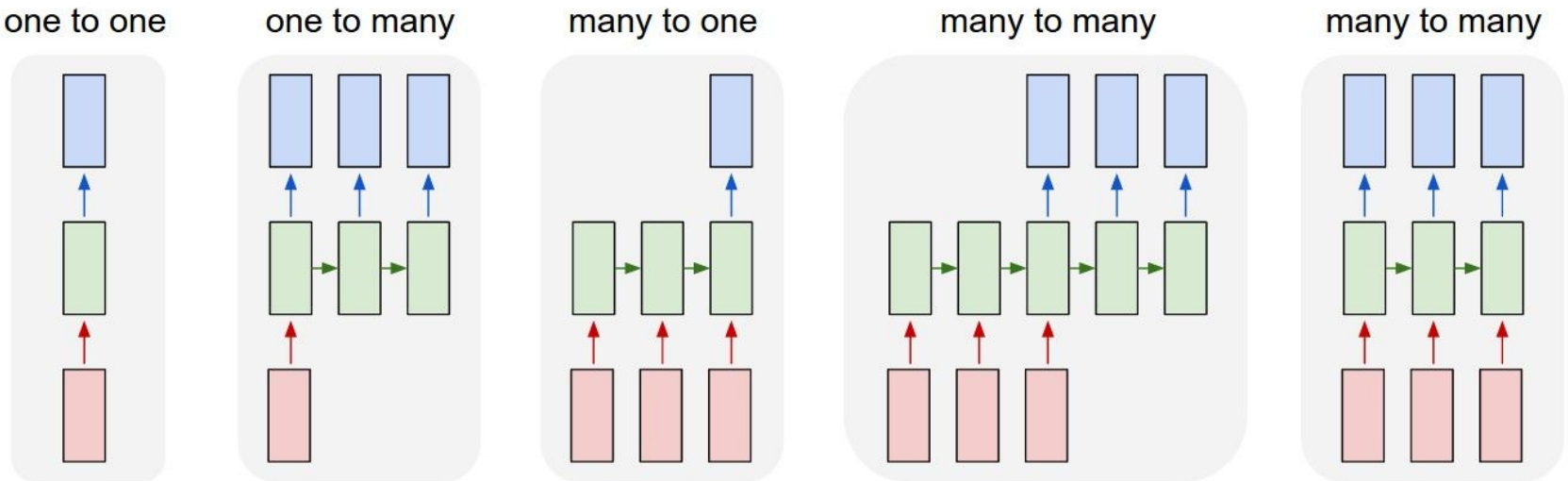
1. Introduction
2. Bidirectional RNN
3. Encoder-Decoder Architectures
4. Long-Term Dependencies
5. Attention as RNN
6. Pre-training of sequence model
7. Summary

Introduction

- A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle.
- Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs.
- The first RNN was invented in 1980s.


Frank Rosenblatt
Cornell University
psychologist
Geoffery Hinton
University of Toronto
deep learning
Yann Lecun
New York University
deep learning
Capsule Nets(2017)
Kaiming He (何恺明)
MSRA => FAIR
computer vision
BPNN/MLP(1986)
Perceptron(1958)
AlexNet(2012)
LeNet/CNN(1998)
ResNet(2016)
Neocognitron(1980)
[convolution & pooling]
60,000 original datasets
Test error: 0.95%
Relu, dropout & bigger
VGG(2014)
GoogLeNet(2015)
DenseNet(2017)
Hopfield Network(1982)
[recurrent & feedback]
AutoEncoder(1989/2006)
Denosing Autoencoder(2008)
VAE(2013)
Variational
Inference
Encoder
Decoder
RBM(1986/2006)
stack
Deep Belief
Nets(2006)
Max Welling
University of Amsterdam
statistical learning
RNN/LSTM(1997)
RNN in Speech
Recognition(2013)
GAN(2014)
Training set
Random
noise
Generator
Discriminator
Real
Fake
Fake image
DCGAN(2014)
WGAN(2017)
PGGAN(2017)
Ian Goodfellow
Google Brain
deep adversarial learning
Neural Probabilistic
Language Model(2003)
word2Vec(2013)
Skip-gram
Seq2Seq(2014)
I am a student
Encoder
Decoder
Je suis étudiant
SeqGAN(2017)
LeakGAN(2018)
Character CNN(2015)
self-attention(2017)
Jürgen Schmidhuber
IDSIA
Universal AI
Yoshua Bengio
University of Montreal
Deep learning
Deep Q-
learning(2013)
Double DQN(2015)
Dueling Net(2016)
David Silver
DeepMind
Reinforcement learning
DDPG(2015)
A3C(2016)
AlphaGo(2016)
Alpha Zero(2017)

Recurrent Networks offer a lot of flexibility...

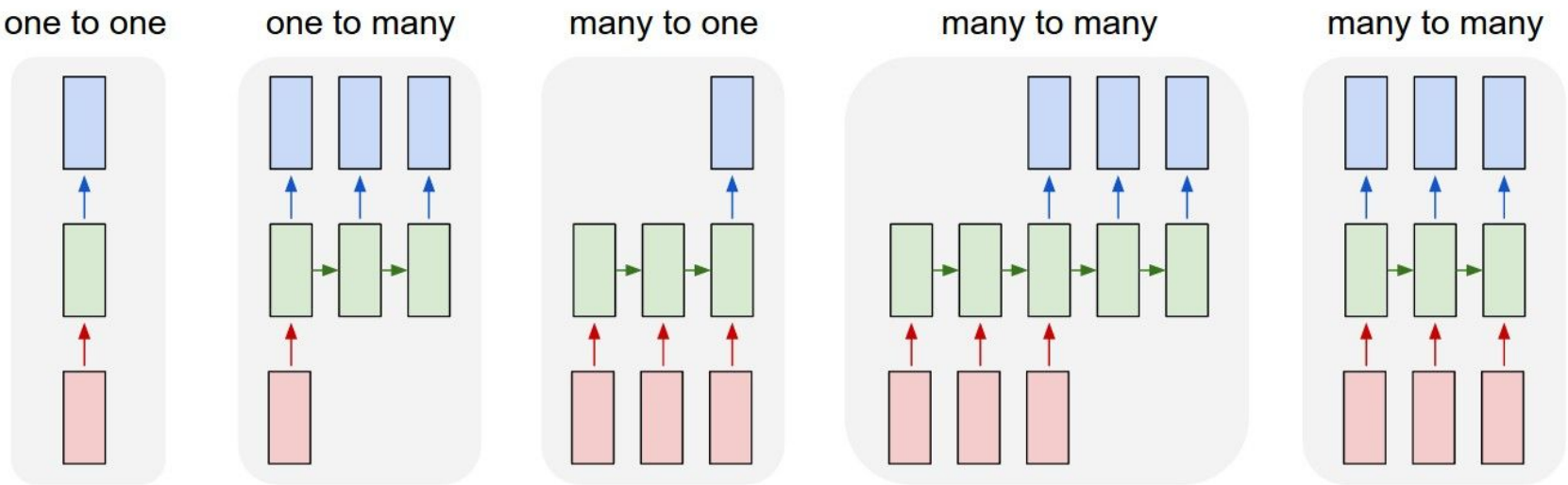


[1]

Simple neural network

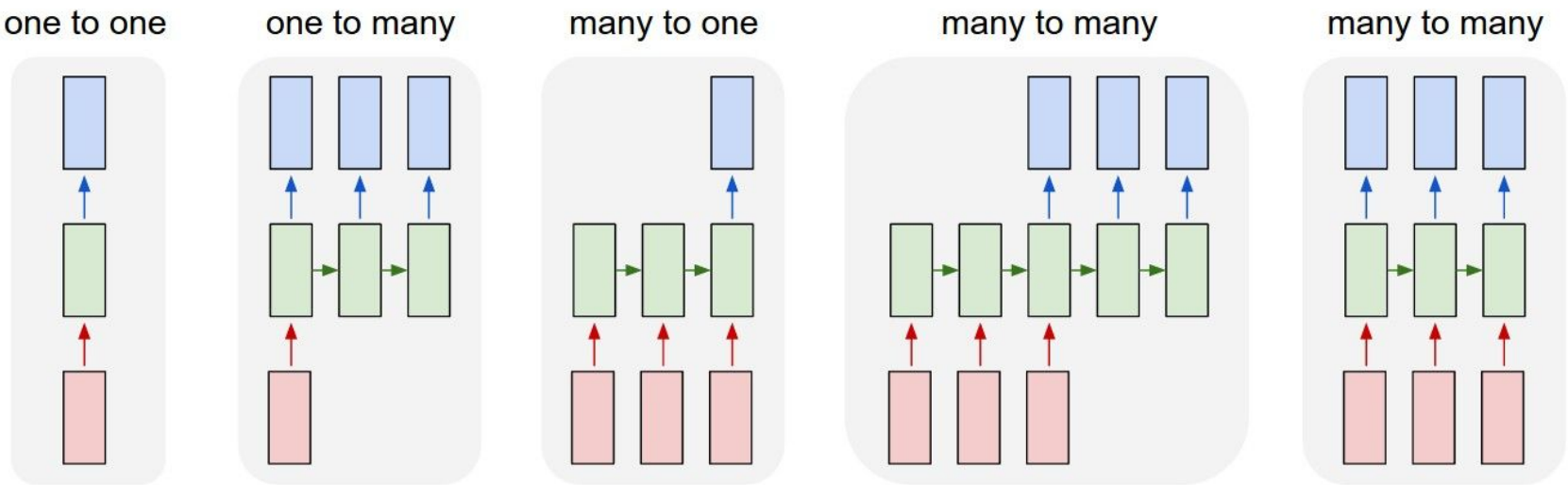
[1] The example is from Fei-Fei Li's slides.

Recurrent Networks offer a lot of flexibility...



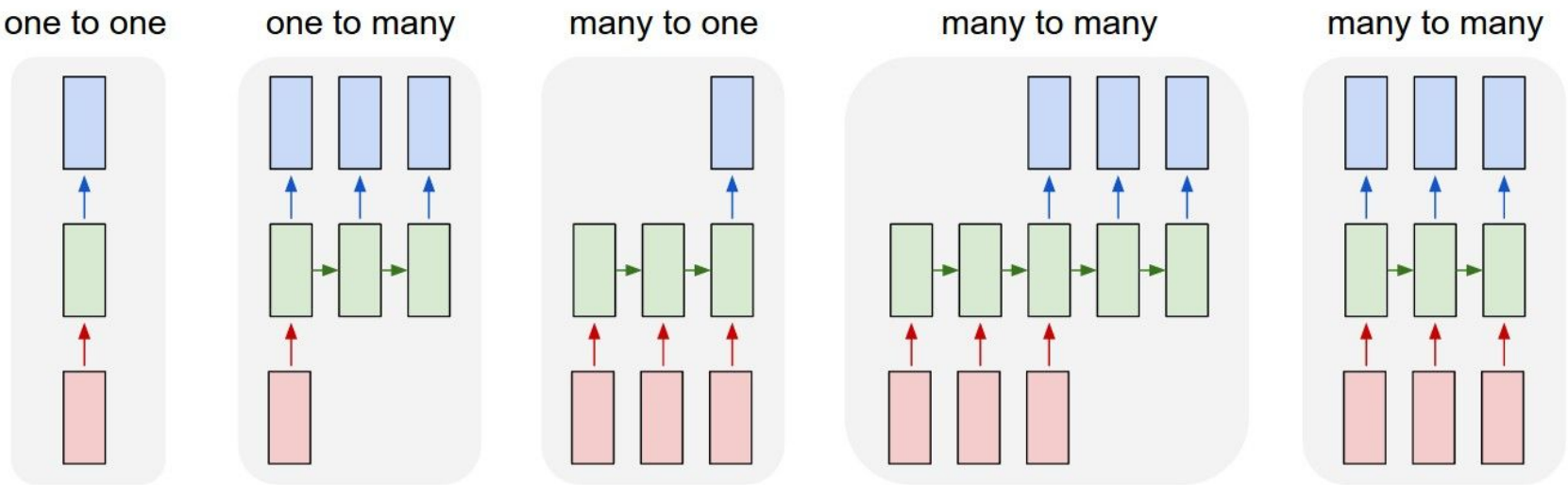
e.g., Image captioning: image$\rightarrow$ sequence-of-words

Recurrent Networks offer a lot of flexibility...



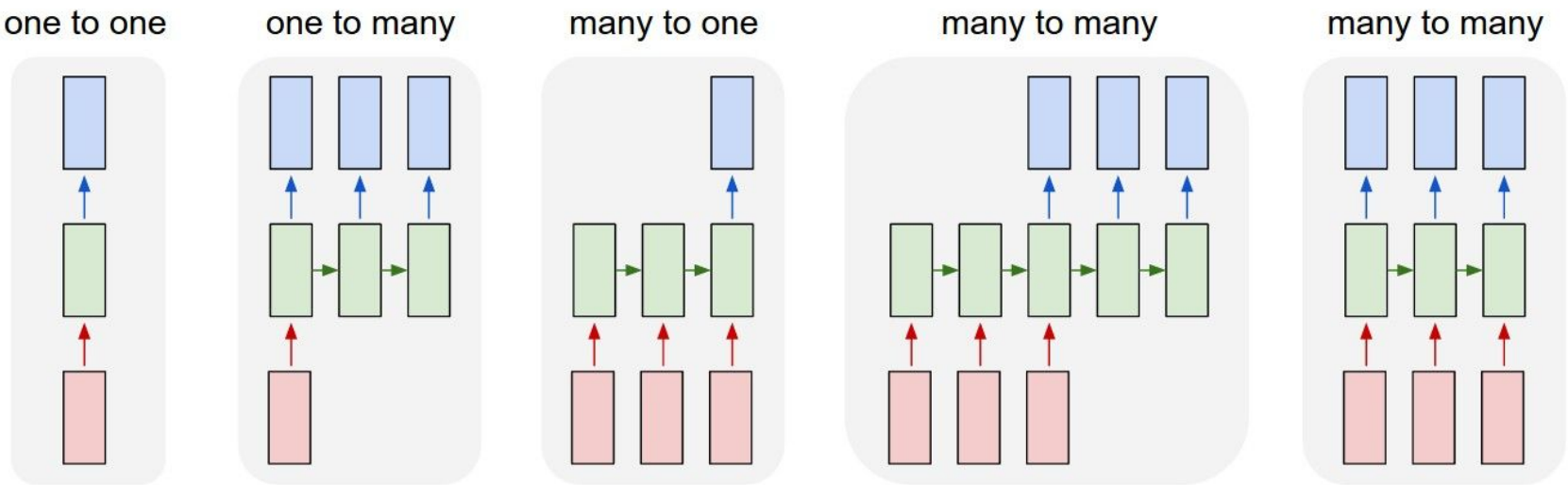
e.g., Sentiment analysis: sequence-of-words $\rightarrow$ sentimen

Recurrent Networks offer a lot of flexibility...



e.g., Machine translation: sequence $\rightarrow$ sequence

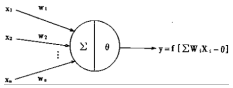
Recurrent Networks offer a lot of flexibility...



e.g., POS Tagging: sequence $\rightarrow$ sequence



Frank Rosenblatt
Cornell University
psychologist



Perceptron(1958)

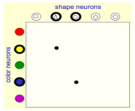
Hopfield Network(1982)
[recurrent & feedback]



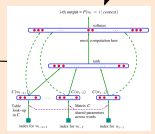
Yoshua Bengio
University of Montreal
Deep learning



Geoffery Hinton
University of Toronto
deep learning



Distributed
representation(1986)



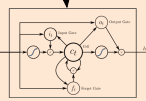
Neural Probabilistic
Language Model(2003)



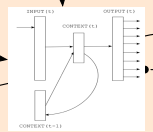
Tomas Mikolov
Google -> Facebook
Deep learning
NLP



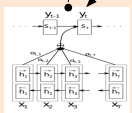
Jürgen Schmidhuber
IDSIA
Universal AI

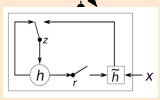


LSTM(1997)



RNN based language
model(2010)

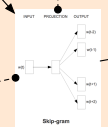




attention seq2seq GRU & seq2seq
(2014.9.1) (2014,6.3)



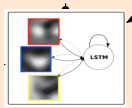
Seq2Seq
(2014.9.10)



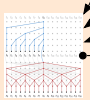
Deepwalk (2014)
Network embedding

word2Vec(2013)

Glove (2014)
count-based word
embedding



"show attend and
tell"(2015)



ByteNet (2016)
[convolution]



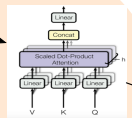
Nal Kalchbrenner
deepmind
deeplearning



Ilya Sutskever
Google -> openAI
Deep learning

context2vec(2016)
contextualized embedding

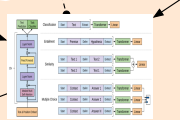
ResNet (2016)



Conv Seq2seq (2017)
Facebook

Transformer
(2017)
Google AI

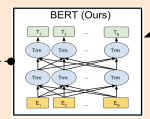
ELMo(2018)
dynamic embedding



OpenAI GPT(2018)
pretraining

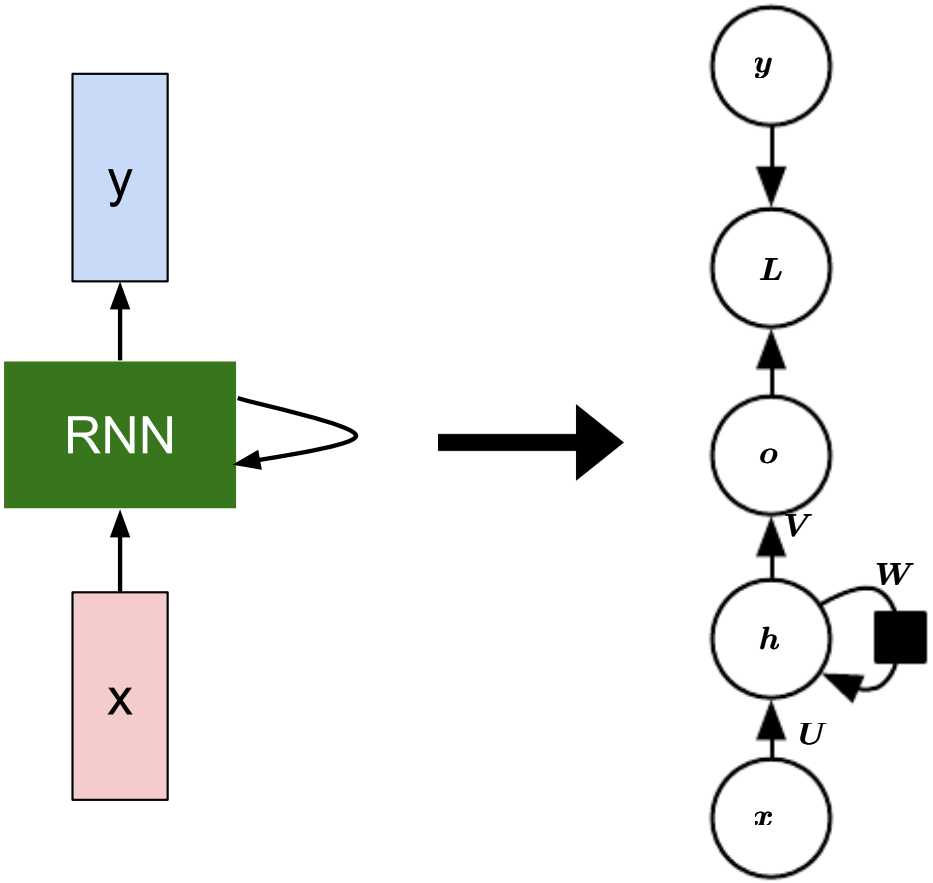


Jacob Devlin
microsoft -> google AI
Machine Translation



BERT(2018)

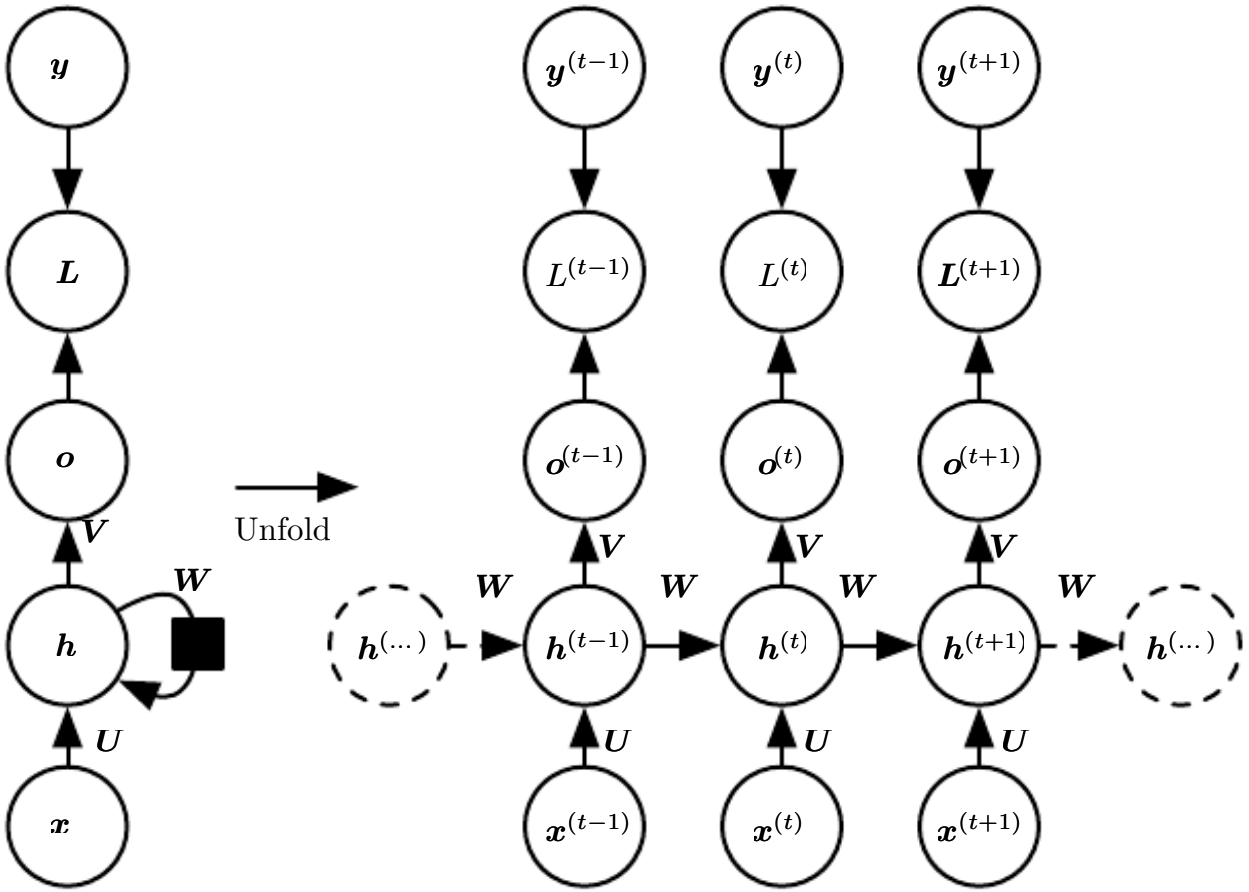
Recurrent Neural Networks — An Example



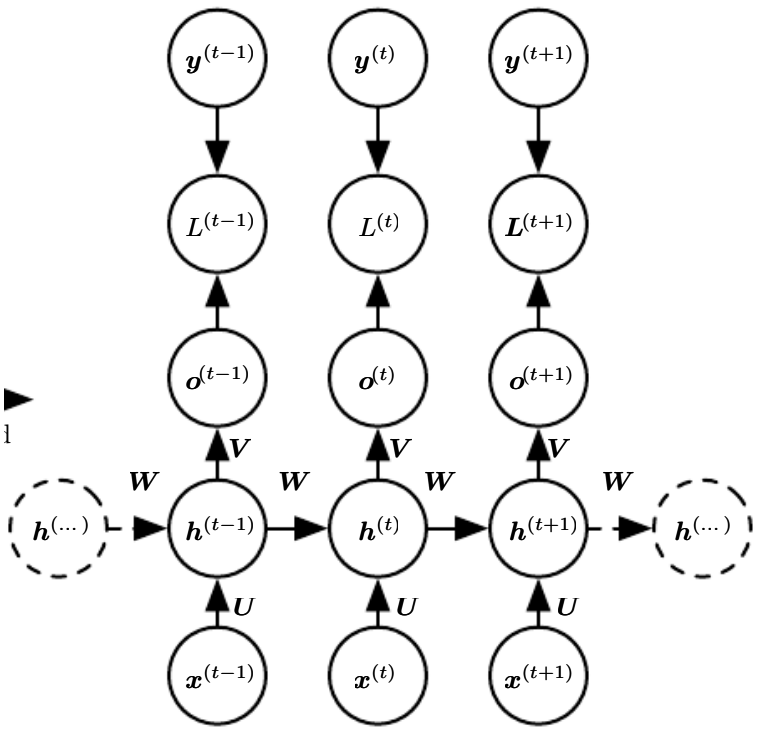
- Input x and output y are from the training data;
- o is the estimated output value;
- L is to evaluate the difference (loss) between the estimated output $\hat{y} = softmax(o)$ and the true output y;
- U, W, V are three weigh matrices.

Recurrent Neural Networks — An Example

- Let us unfold the graphical representation.



Recurrent Neural Networks — An Example



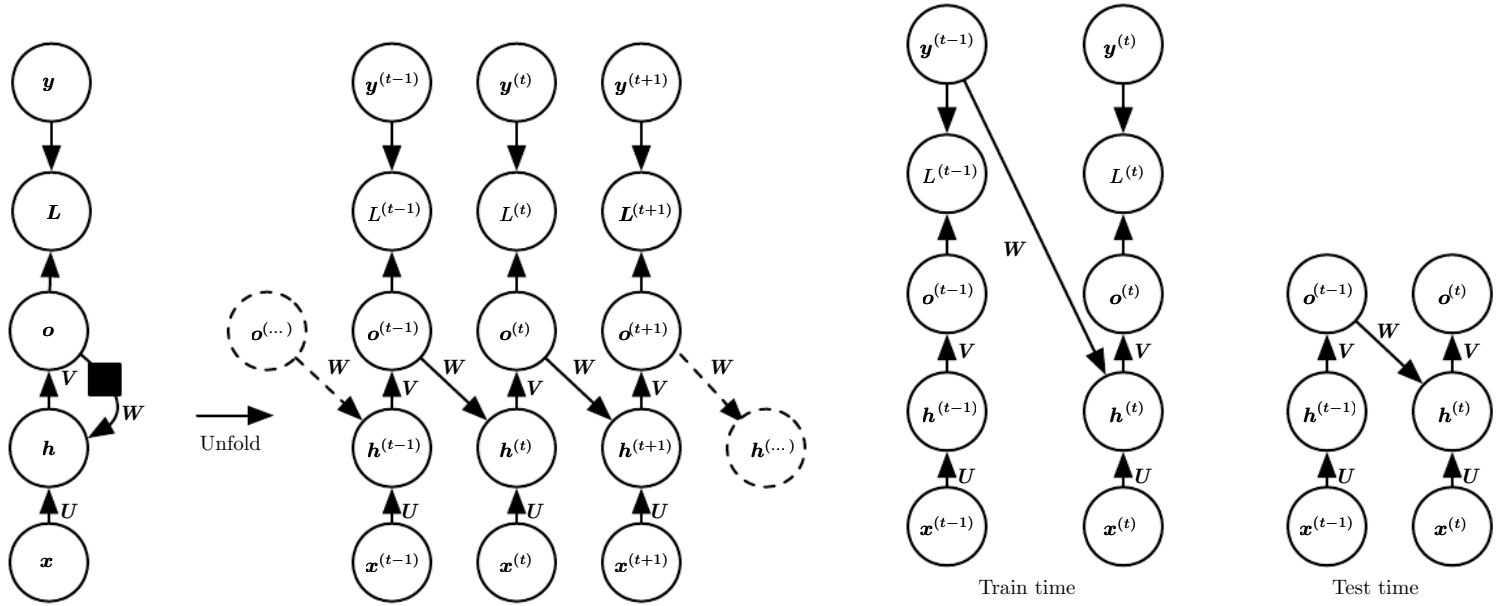
Update Equations

- $\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)})$
- $\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}$
- $\hat{\boldsymbol{y}}^{(t)} = \text{softmax}(\boldsymbol{o}^{(t)})$
- Objective function: $L = \sum_t L^{(t)} = \sum_t \log P(y^{(t)} | \{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(t)}\})$

Recurrent Neural Networks

- There are many different types of recurrent neural networks
  - Recurrent connections between hidden units
  - Recurrent connections from the output at one time step to the hidden units at the next time step
  - Recurrent connections between hidden units but produces only one output
  - ...

Recurrent Neural Networks – Output Recurrence



(a) Network with Output Recurrence (b) Teacher Forcing

Figure 1: Other Types of RNN. Output Recurrence: the only recurrence is from the output to the hidden layer; Teacher Forcing: training technique for RNN.

Conditional likelihood of Teacher Forcing:

$$\log p(y^{(1)}, y^{(2)}|x^{(1)}, x^{(2)}) = \log p(y^{(2)}|y^{(1)}, x^{(1)}, x^{(2)}) + \log p(y^{(1)}|x^{(1)}, x^{(2)})$$

Discussion on Output Recurrence

Advantage

- all the time steps are decoupled, training can be in parallel

Disadvantage

- lack hidden-to-hidden recurrence, strictly less powerful

Solution

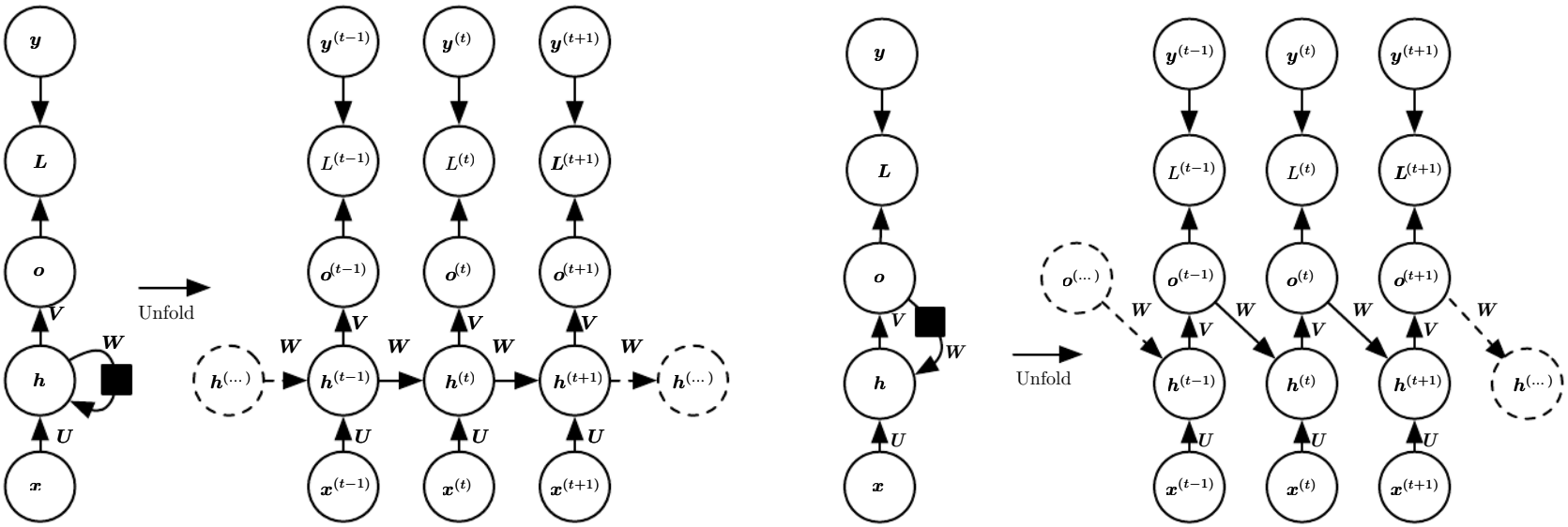
- randomly choose to use generated values or actual values as input

Gradient Computation in RNN

The gradient can be computed using a generalized back-propagation algorithm, specifically called back-propagation through time (BPTT):

$$
\begin{aligned}
(\nabla_{\boldsymbol{o}^{(t)}} L)_i &= \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \mathbf{1}_{i,y^{(t)}} \\
\nabla_{\boldsymbol{h}^{(t)}} L &= \boldsymbol{W}^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) \mathrm{diag}\left(1 - (\boldsymbol{h}^{(t+1)})^2\right) + \boldsymbol{V}^\top (\nabla_{\boldsymbol{o}^t} L) \\
\nabla_{\boldsymbol{c}} L &= \sum_t \nabla_{\boldsymbol{o}^{(t)}} L \\
\nabla_{\boldsymbol{b}} L &= \sum_t \mathrm{diag}\left(1 - (\boldsymbol{h}^{(t)})^2\right) \nabla_{\boldsymbol{h}^{(t)}} L \\
\nabla_{\boldsymbol{V}} L &= \sum_t (\nabla_{\boldsymbol{o}^{(t)}} L) \boldsymbol{h}^{(t)\top} \\
\nabla_{\boldsymbol{W}} L &= \sum_t \mathrm{diag}\left(1 - (\boldsymbol{h}^{(t)})^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L) \boldsymbol{h}^{(t-1)\top} \\
\nabla_{\boldsymbol{U}} L &= \sum_t \mathrm{diag}\left(1 - (\boldsymbol{h}^{(t)})^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L) \boldsymbol{x}^{(t)\top}
\end{aligned}
\tag{1}
$$

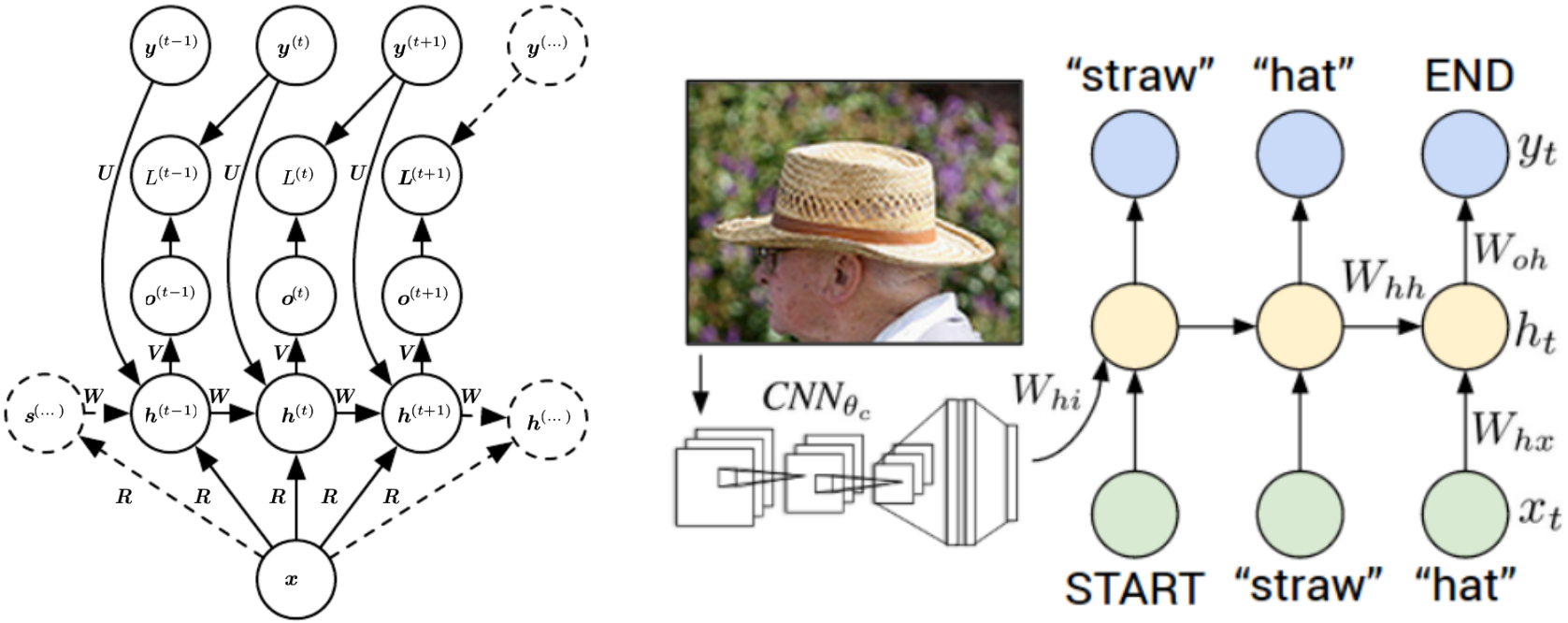
RNN as Directed Graphical Models



(a) Hidden-hidden Recurrence (b) Output-hidden Recurrence

Figure 2: Two Types of RNN we have discussed. Left: maximize the log-likelihood $\log P(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(t)})$; Right: maximize the log-likelihood $\log P(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(t)}, \boldsymbol{y}^{(1)}, \cdots, \boldsymbol{y}^{(t-1)})$

Modeling Sequences Conditioned on Context with RNNs



(a) (b)

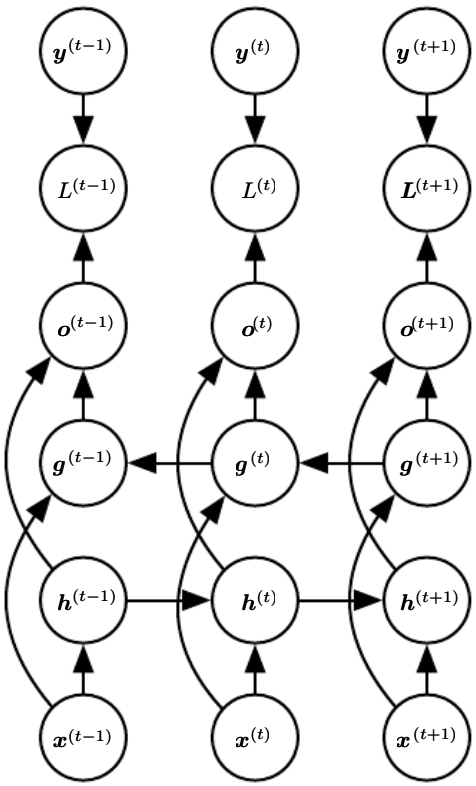
Figure 3: RNN for tasks like image captioning Left: a single image as input, the model produces a sequence of words describing the image. Right: a real example from Feifei Li's CVPR'15 paper. However, the RNN is only conditioned on the image information at the first time step

Overview

1. Introduction
2. Bidirectional RNN
3. Encoder-Decoder Architectures
4. Long-Term Dependencies
5. Attention as RNN
6. Pre-training of sequence model
7. Summary

Bidirectional RNNs

For classification you want to incorporate information from both preceding and following.

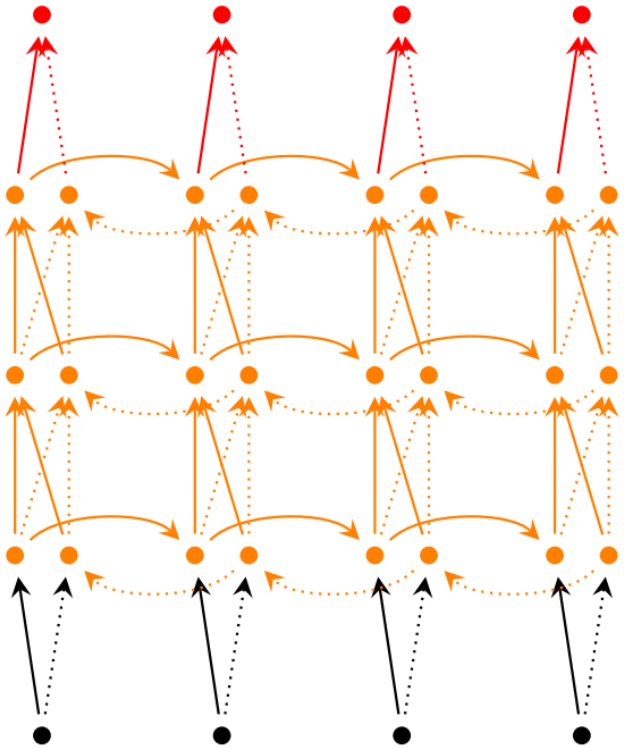


Bidirectional RNNs

- Two hidden layers, one for the left-to-right propagation and another for the right-to-left propagation

$$\vec{h}_t = f(\vec{W}x_t + \vec{V}\vec{h}_{t-1} + \vec{b})$$
$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t-1} + \overleftarrow{b})$$
$$\hat{y} = g(Uh_t + c) = g(U[\vec{h}_t, \overleftarrow{h}_t] + c) \quad (2)$$

Deep Bidirectional RNN



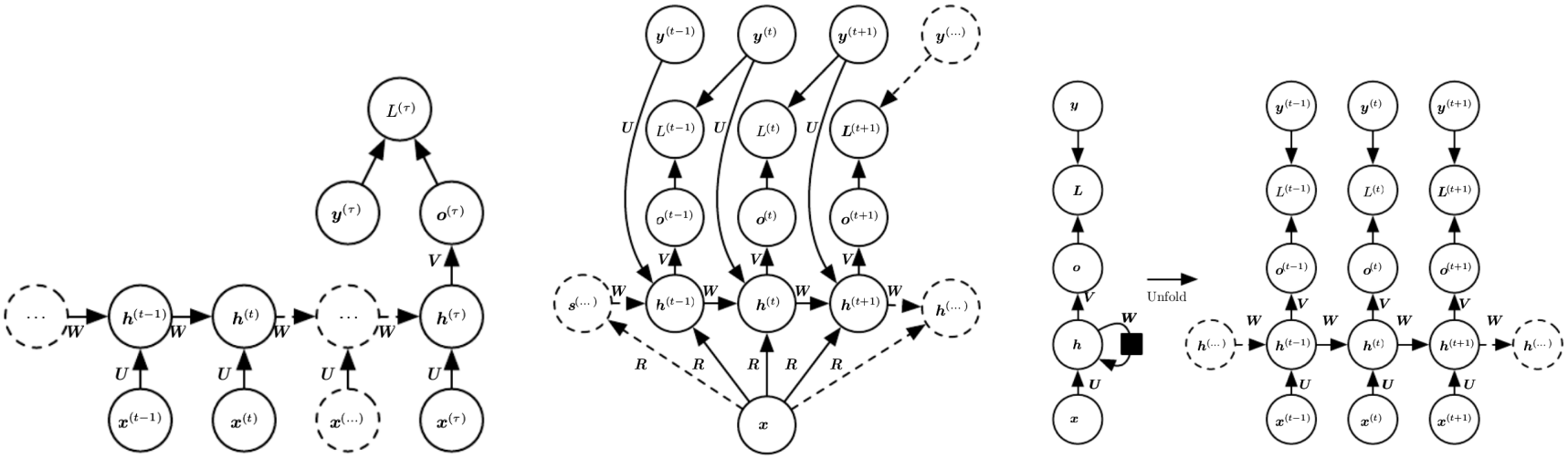
A deep bidirectional RNN with three RNN layers.

$$
\begin{aligned}
\vec{h}_t^i &= f(\vec{W}^i h_t^{i-1} + \vec{V}^i \vec{h}_{t-1}^i + \vec{b}^i) \\
\overleftarrow{h}_t^i &= f(\overleftarrow{W}^i h_t^{i-1} + \overleftarrow{V}^i \overleftarrow{h}_{t-1}^i + \overleftarrow{b}^i) \\
\hat{y} &= g(U h_t + c) = g(U[\vec{h}_t^L, \overleftarrow{h}_t^L] + c)
\end{aligned} \tag{3}
$$

Overview

1. Introduction
2. Bidirectional RNN
3. Encoder-Decoder Architectures
4. Long-Term Dependencies
5. Attention as RNN
6. Pre-training of sequence model
7. Summary

Encoder-Decoder Sequence-to-Sequence Architectures



(a) seq to fixed-size vec (b) fixed-size vec a seq (c) seq to seq of the same length

Figure 4: RNNs that we have discussed so far.

Can we train a RNN model to map an input sequence to an output sequence which is **not necessarily of the same length**?
This is important in many applications: speech recognition, machine translation, or question answering.

Encoder-Decoder Sequence-to-Sequence Architectures

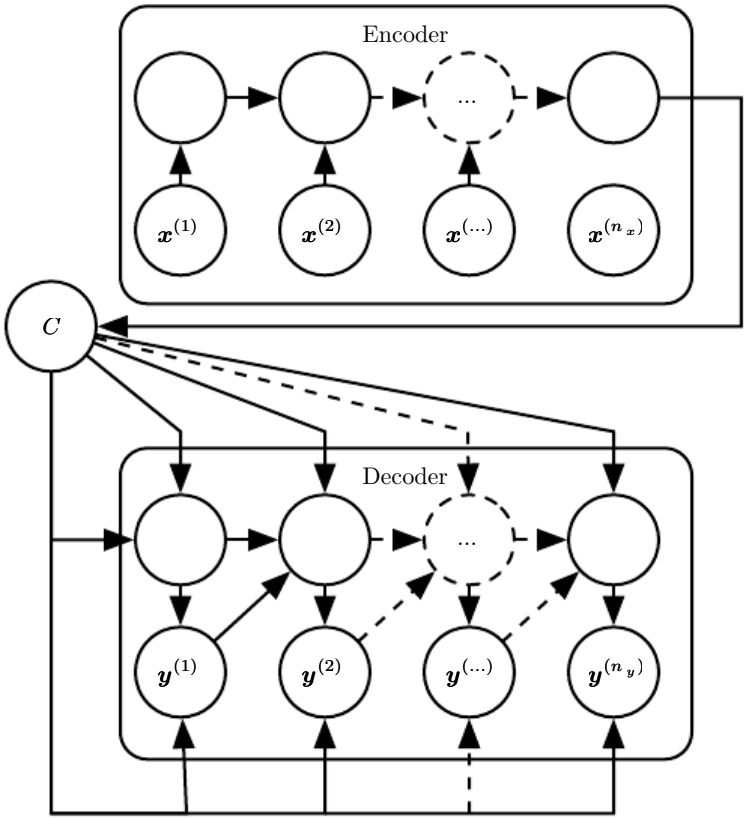


Figure 5: Encoder-decoder or sequence-to-sequence RNN architecture.

When C is a fixed-size vector, this architecture could be reviewed as a combination of a sequence to fixed-size vector RNN and a fixed-size vector to sequence RNN.

Encoder-Decoder Sequence-to-Sequence Architectures

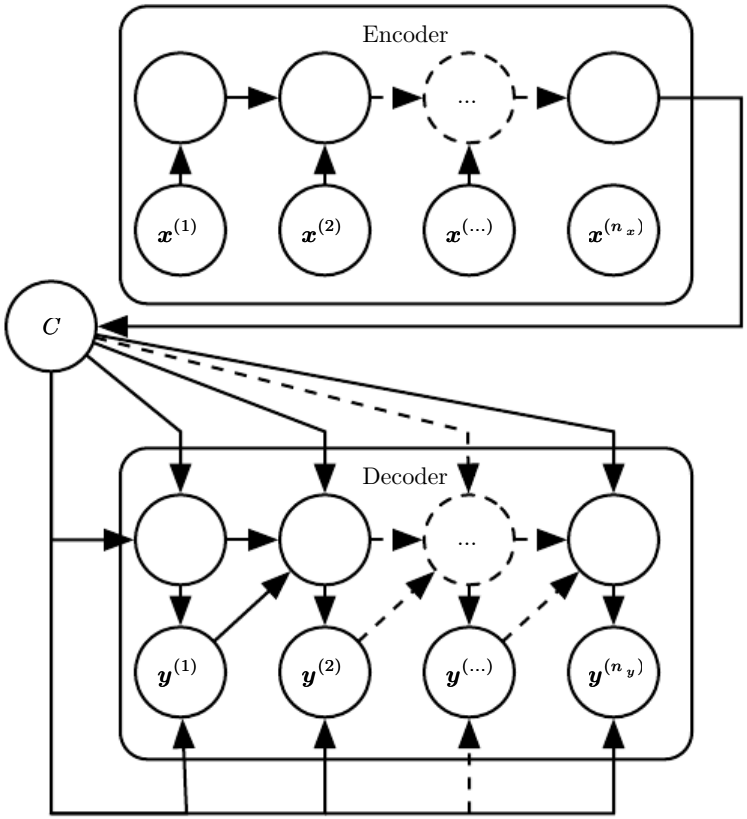


Figure 5: Encoder-decoder or sequence-to-sequence RNN architecture.

When C is a fixed-size vector, this architecture could be reviewed as a combination of a sequence to fixed-size vector RNN and a fixed-size vector to sequence RNN.

Limitation: C may be too small to summarize a long sequence (Bahdanau et al. (2015)).

Encoder-Decoder Seq-to-Seq Architectures

C may be too small to summarize a long sequence (Bahdanau et al. (2015)). They proposed to make C a variable-length sequence. Additionally, they introduced the attention mechanism.

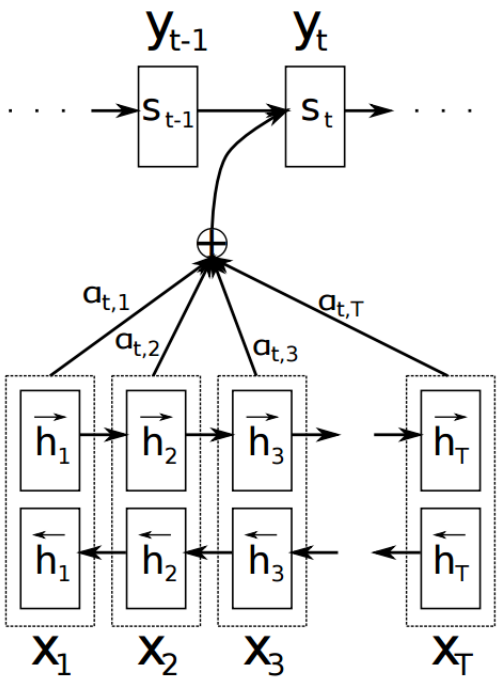
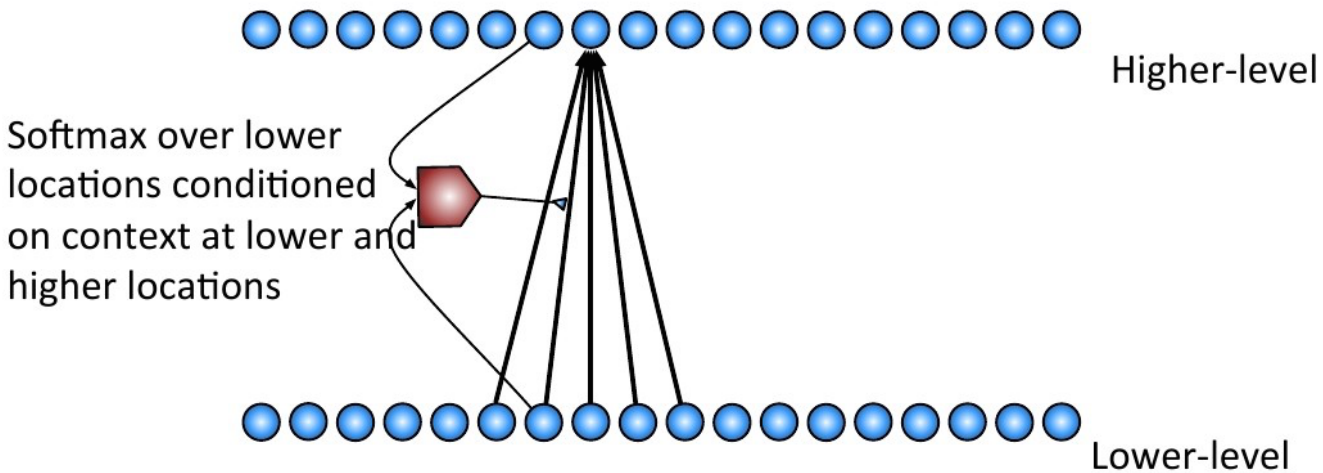


Figure 6: Formulatin: $a(\cdot)$ is an alignment model which scores how well the inputs around position j and the output at position i match, which is parameterized as a feedforward neural network

Attention Mechanism with Seq-to-Seq Architectures

- Consider an input (or intermediate) sequence
- Consider an upper level representation, which can choose "where to look" when producing some task (e.g., machine translation) at each position



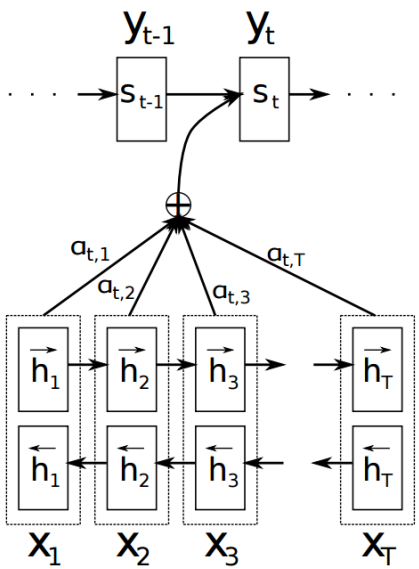
Attention Mechanism with Seq-to-Seq Architectures

- Decoder: is often trained to predict the next word y_t given the context vector c and all the previous words $\{y_1, y_2, ..., y_{t-1}\}$

$$p(y_t|y_1, ..., y_{t-1}, \mathbf{x}) = g(y_{t-1}, s_t, c_t) \tag{4}$$

where s_t is an RNN hidden state for position t, computed by

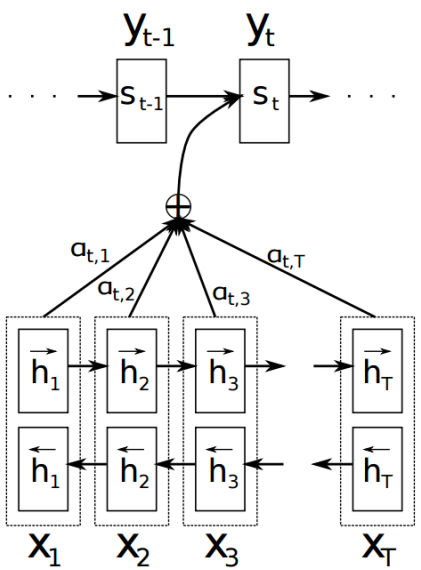
$$s_i = f(s_{i-1}, y_{i-1}, c_i) \tag{5}$$



Attention Mechanism with Seq-to-Seq Architectures

- The context vector c_t depends on a sequence of *annotations* $(h_1, ..., h_T)$ to which an encoder maps the input sentence, computed by

$$c_t = \sum_{j=1}^{T} \alpha_{tj} h_j \tag{6}$$



Attention Mechanism with Seq-to-Seq Architectures

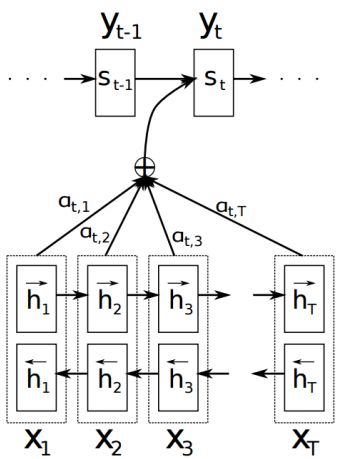
- The weight α is so called "attention matrix" (used to choose "where to look") and is computed by

$$\alpha_{tj} = \frac{\exp\{e_{tj}\}}{\sum_{k=1}^{T} e_{tk}} \tag{7}$$

where

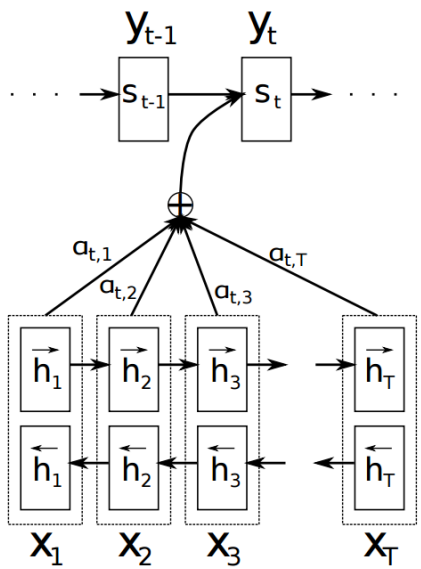
$$e_{tj} = h(s_{t-1}, h_j) \tag{8}$$

is an attention model which scores how well the inputs around position j and the output at position t match.



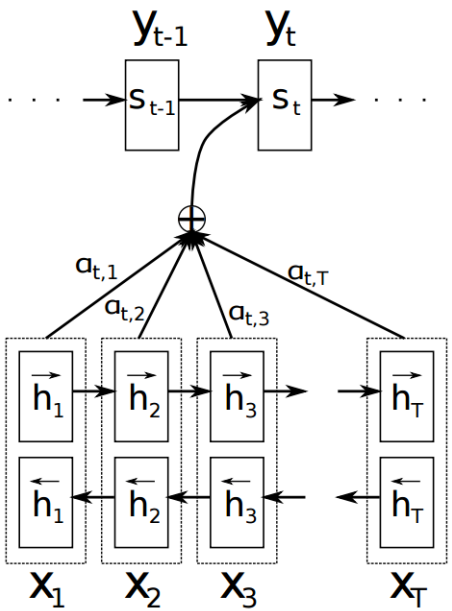
Attention Mechanism with Seq-to-Seq Architectures

- The attention model can be parametrized as a feedforward neural network (or multilayer perceptron, etc.) which is jointly trained with all the other components of the system.



Attention Mechanism with Seq-to-Seq Architectures

To summarize, the attention mechanism with sequence-to-sequence architectures have the following formalization:



$$
\begin{aligned}
p(y_i | y_1, \cdots, y_{i-1}, \mathbf{x}) &= g(y_{i-1}, s_i, c_i) \\
s_i &= f(s_{i-1}, y_{i-1}, c_i) \\
c_i &= \sum_{j=1}^{T_x} \alpha_{ij} h_j \\
\alpha_{ij} &= \frac{\exp(e_{ij})}{\sum_{j=1}^{T_x} \exp(e_{ij})} \\
e_{ij} &= a(s_{i-1}, h_j)
\end{aligned}
$$

Figure 7: Formulatin: $a(\cdot)$ is an alignment model which scores how well the inputs around position j and the output at position i match, which is parameterized as a feedforward neural network

Overview

1. Introduction
2. Bidirectional RNN
3. Encoder-Decoder Architectures
4. Long-Term Dependencies
5. Attention as RNN
6. Pre-training of sequence model
7. Summary

Let us review the vanishing and exploding gradient problem

Let's first review what is long-term dependencies mentioned in Lecture "Optimization for Training Deep Models".

Pure Linear Case

- Consider a RNN with no nolinearity

$$h^{(t)} = Wh^{(t-1)} + Vx^{(t)} + b$$

$$\frac{\partial h^{(t)}}{\partial h^{(k)}} = \prod_{j=k+1}^{t} \frac{\partial h^{(j)}}{\partial h^{(j-1)}} = \prod_{j=k+1}^{t} W^{\top} = (W^{\top})^{t-k} \tag{9}$$

- Suppose that W has an eigendecomposition $W^{\top} = V\text{diag}(\lambda)V^{-1}$, then

$$(W^{\top})^{t-k} = V\text{diag}(\lambda)^{t-k}V^{-1} \tag{10}$$

- Vanishing and exploding gradient problem: Any eigenvalues λ_i that are not near an absolute value of 1 will either explode (if $|\lambda_i| > 1$) or vanish (if $|\lambda_i| < 1$).

Vanishing and exploding gradient problem

Non-linear Case

Let's consider the Jacobi Matrix and its norm:

$$h^{(t)} = f(Wh^{(t-1)} + Vx^{(t)} + b)$$

$$\frac{\partial h^{(t)}}{\partial h^{(k)}} = \prod_{j=k+1}^{t} \frac{\partial h^{(j)}}{\partial h^{(j-1)}} = \prod_{j=k+1}^{t} W^{\top} \text{diag}\left(f'(h^{(j-1)})\right) \tag{11}$$

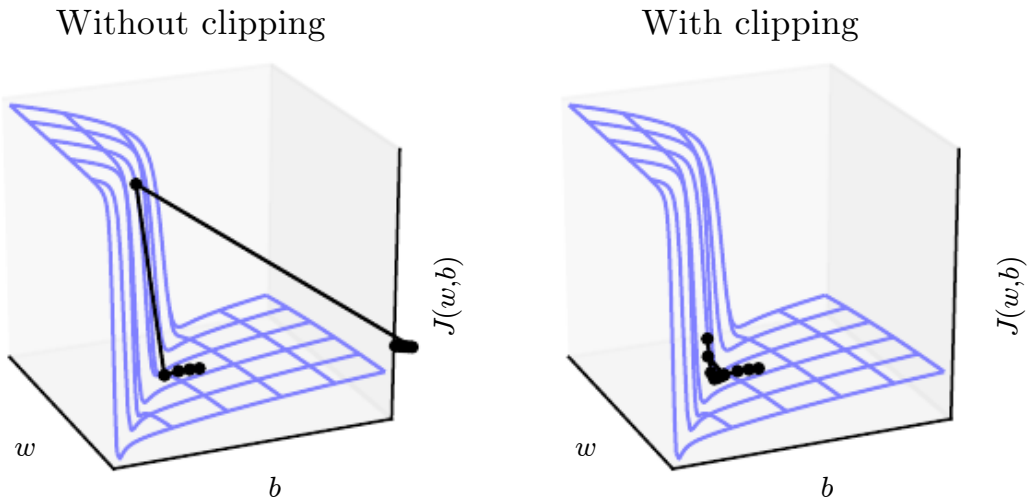
$$||\frac{\partial h^{(t)}}{\partial h^{(j-1)}}|| \leq ||W^{\top}|| \times ||\text{diag}\left(f'(h^{(j-1)})\right)||$$

Non-linear activation function may

- bound recurrent dynamics, alleviating the exploding gradient problem.
- make vanishing gradient problem severer.

The remaining of this slides discuss various approaches to reduce the difficulty of learning long-term dependencies.

Trick for exploding gradient: Clipping Gradients



The solution first introduced by Mikolov is to clip gradients...

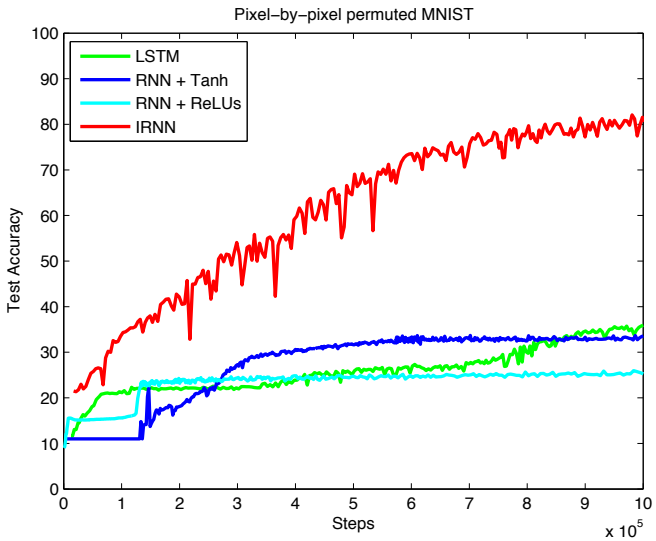
Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

It makes big difference in RNN!

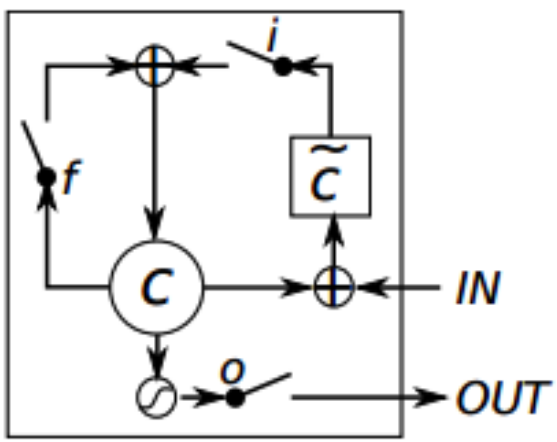
For vanishing gradients: Initialization+ReLU

- Initialize $W(\star)$'s to identity matrix I and ReLU $f(z) = \max(0, z)$
- It makes huge difference!
- Several ideas about initialization for DL Initialization idea first introduced in Parsing with Compositional Vector Grammars, Socher et al. 2013
  A Simple Way to Initialize Recurrent Networks of Rectified

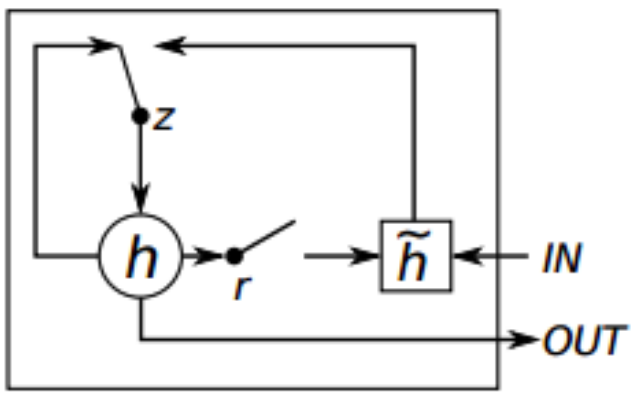


Gated RNNs and Long Short-Term Memory

- The Long-Short-Term-Memory (LSTM) algorithm was proposed in 1997 (Hochreiter and Schmidhuber, 1997).
- Recent work on gated RNNs, Gated Recurrent Units (GRU) was proposed in 2014
- The LSTM has been found extremely successful in many applications, such as unconstrained handwriting recognition, speech recognition, handwriting generation, machine translation, image captioning and parsing.



(a) Long Short-Term Memory



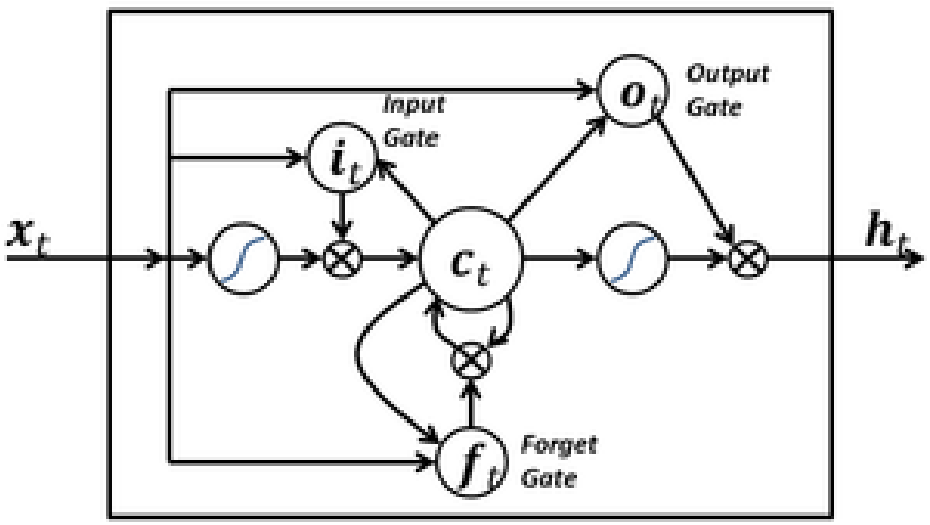
(b) Gated Recurrent Unit

LSTM

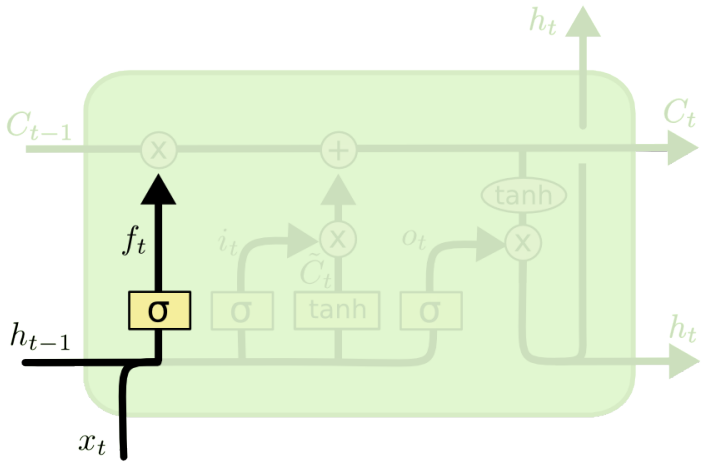
- Forget Gate: $\boldsymbol{f}^{(t)} = \text{sigmoid}(\boldsymbol{W}^f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}^f \boldsymbol{x}^{(t)} + \boldsymbol{b}^f)$
- Input Gate: $\boldsymbol{i}^{(t)} = \text{sigmoid}(\boldsymbol{W}^i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}^i \boldsymbol{x}^{(t)} + \boldsymbol{b}^i)$
- Output Gate: $\boldsymbol{o}^{(t)} = \text{sigmoid}(\boldsymbol{W}^o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}^o \boldsymbol{x}^{(t)} + \boldsymbol{b}^o)$
- Cell State Vector:

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tanh(\boldsymbol{W}^c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}^c \boldsymbol{x}^{(t)} + \boldsymbol{b}^c)$$

- Final Hidden State: $\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \circ \tanh(\boldsymbol{c}^{(t)})$



LSTM: Forget Gate

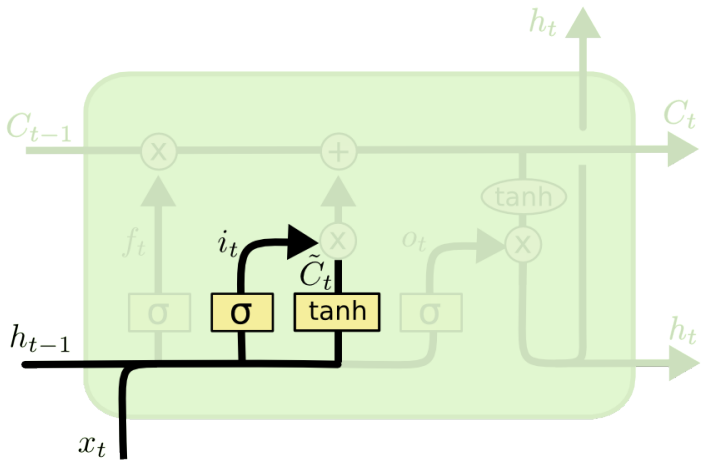


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Forget gate looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}.

- A one represents "completely keep this".
- A zero represents "completely get rid of this.".

LSTM: Input Gate

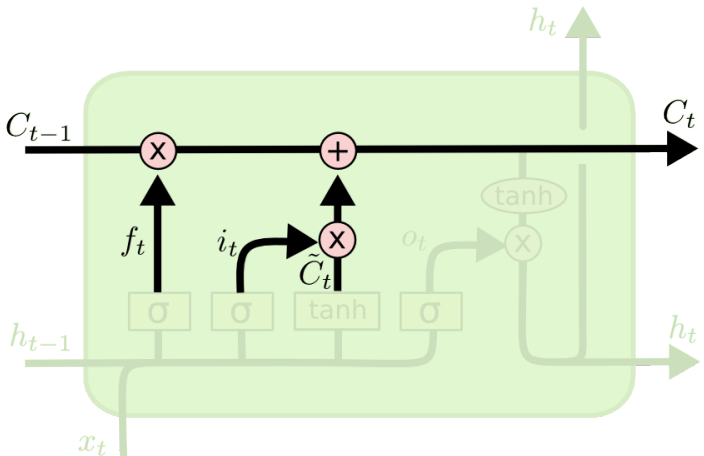


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Decide what new information we're going to store in the cell state.

- a sigmoid layer called the "input gate layer" decides which values we'll update.
- a tanh layer creates a vector of new candidate values of cell state.

LSTM: Update Cell State

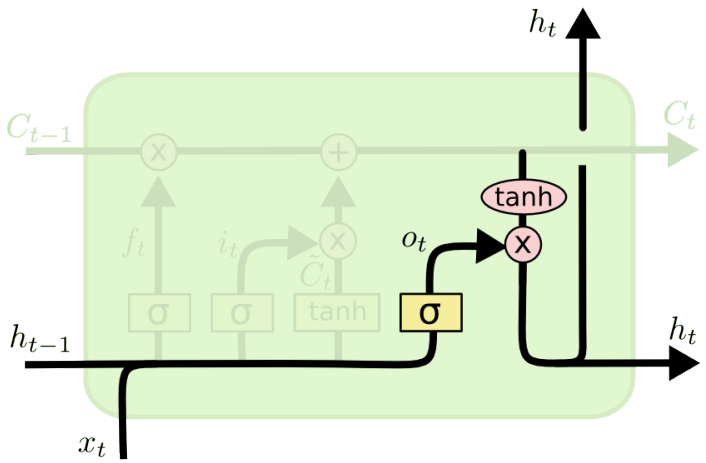


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Update the old cell state

- multiply the old state by f_t (results from forget gate), forgetting the things we decided to forget.
- add the new candidate values, scaled by i_t — how much we decided to update each state value.

LSTM: Output Gate



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

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

Output final results.

- a sigmoid layer which decides what parts of the cell state we're going to output.
- put the cell state through tanh and multiply it by the output of the sigmoid gate.

Gated Recurrent Units (GRU, 2014)

Let's denote element-wise product (Hadamard product) as $\circ$

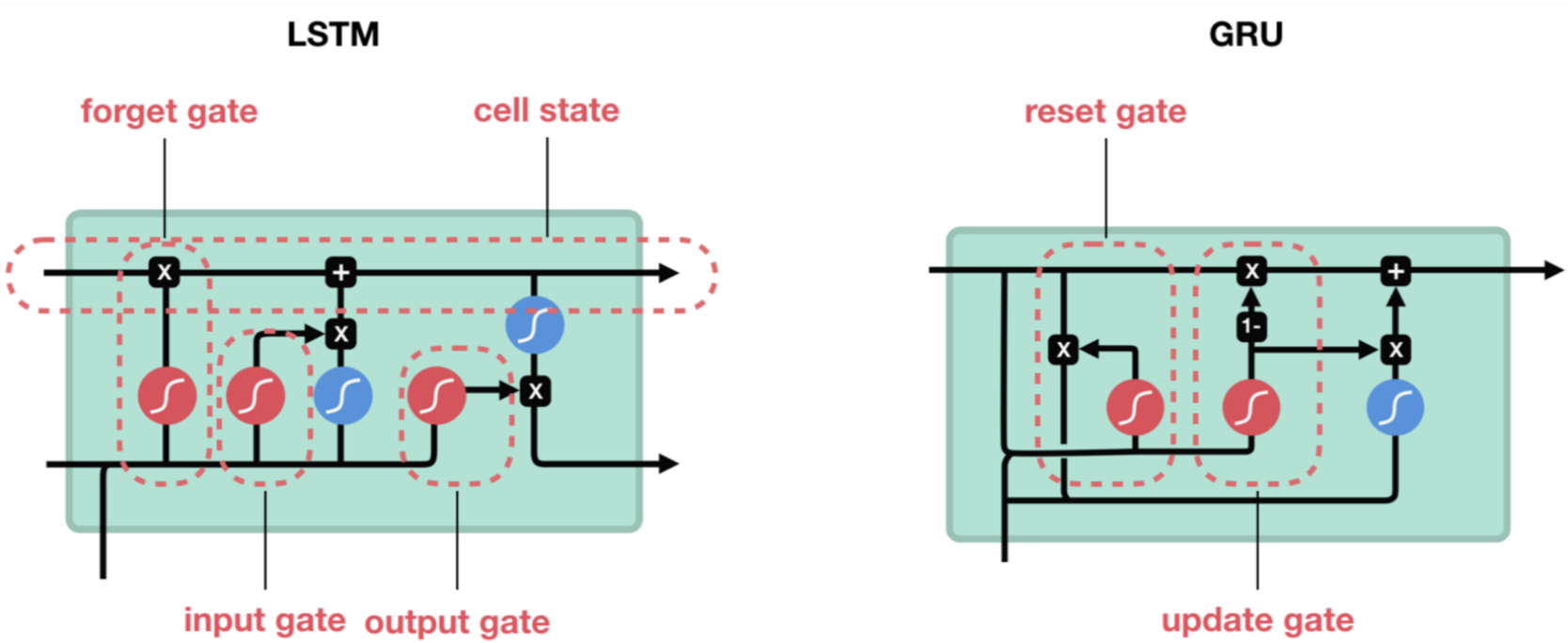
- Update Gate: $\boldsymbol{z}^{(t)} = \text{sigmoid}(\boldsymbol{W}^z \boldsymbol{h}^{(t-1)} + \boldsymbol{U}^z \boldsymbol{x}^{(t)} + \boldsymbol{b}^z)$
- Reset Gate: $\boldsymbol{r}^{(t)} = \text{sigmoid}(\boldsymbol{W}^r \boldsymbol{h}^{(t-1)} + \boldsymbol{U}^r \boldsymbol{x}^{(t)} + \boldsymbol{b}^r)$
- New Memory Content:
  $\bar{\boldsymbol{h}}^{(t)} = \tanh\left(\boldsymbol{W}(\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}) + \boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{b}\right)$
- Final Memory: $\boldsymbol{h}^{(t)} = z^{(t)} \circ \boldsymbol{h}^{(t-1)} + (1 - z^{(t)}) \circ \bar{\boldsymbol{h}}^{(t)}$

Remark

- The reset gate $\boldsymbol{r}^{(t)}$ close to 0 makes RNN ignore previous hidden state — allow model to drop information that is irrelevant
- The Update gate $\boldsymbol{z}^{(t)}$ control how much of past state should matter now.

LSTM vs GRU

- GRUs achieve similar performance in a variety of tasks compared to LSTM.
- GRUs train faster than LSTMs on less training data.
- LSTM outperform GRUs in tasks requiring modeling long-distance relations.



Overview

1. Introduction
2. Bidirectional RNN
3. Encoder-Decoder Architectures
4. Long-Term Dependencies
5. Attention as RNN
6. Pre-training of sequence model
7. Summary

Problem with RNNs

- Hard to parallelization efficiently
- Back propagation through sequence
- Transmitting local and global information through one bottleneck (hidden state)

Convolutional Models

- Trying to solve the problems with sequence models
- Notable work:
  - Neural GPU (Kaiser et al., 2016)
  - ByteNet (Kalchbrenner et al., 2017)
  - ConvS2S (Gehring et al., 2017)
- Limited by the size of convolution

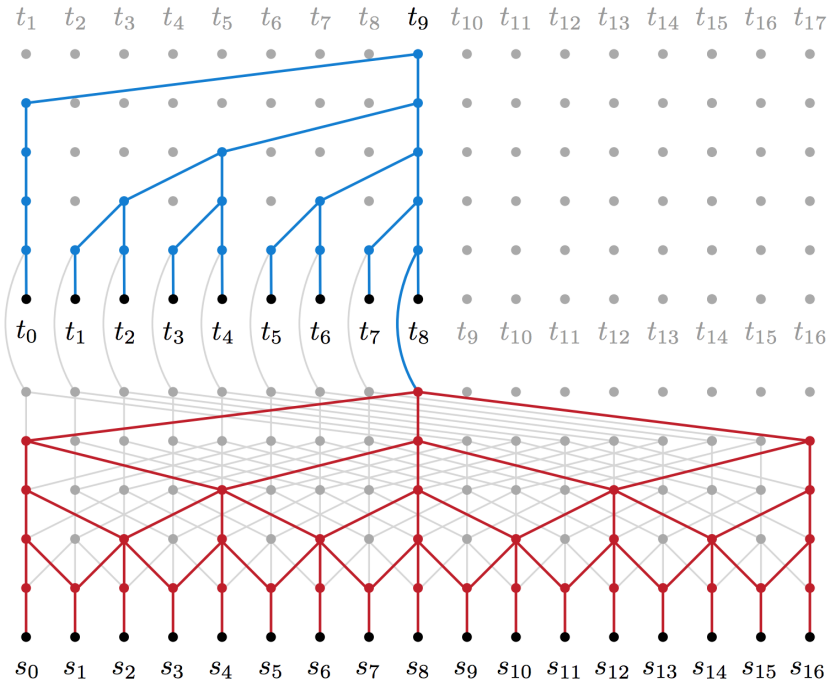


Figure 8: The architecture of the ByteNet.

Why Self-Attention?

- Constant path length between any two positions.
- Trivial to parallelize (per layer).

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

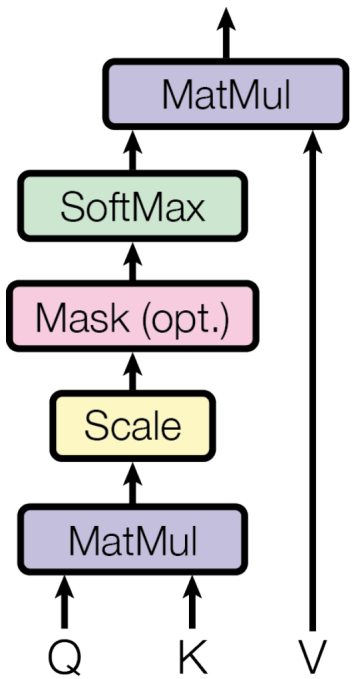
Scaled Dot-Product Attention

- Problem of dot-product attention: scale of dot product increases as dimensions get larger
- Fix: scaled by the size of the vector.

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

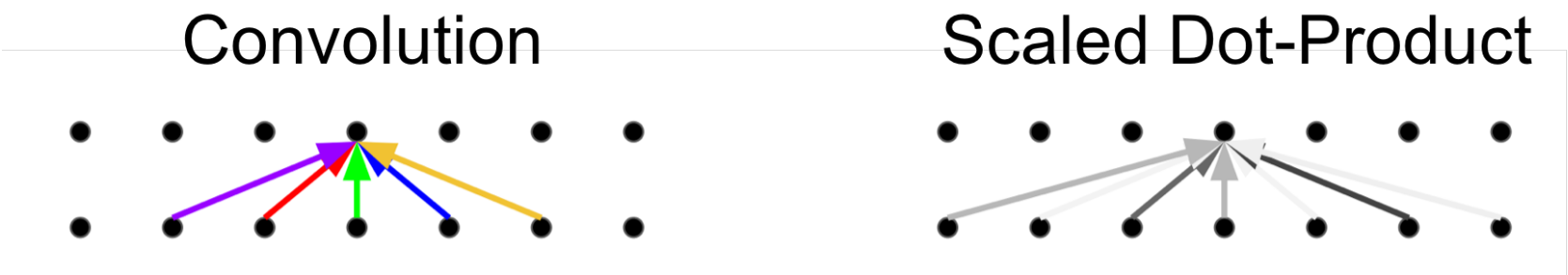
d_k: the dimensionality of Q and K.

Scaled Dot-Product Attention



What's missing from Scaled Dot-Product Attention?

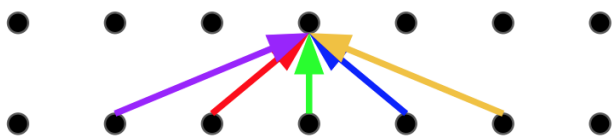
- Convolution: a different linear transformation for each relative position.
- Scaled Dot-Product: a weighted average :(



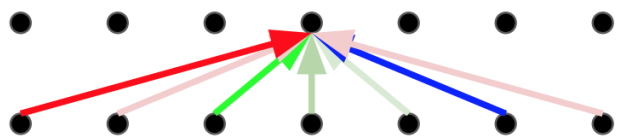
The Fix: Multi-Head Attention

- Multiple attention layers (heads) in parallel (shown by different colors).
- Each head uses different linear transformations.
- Different heads can learn different relationships.

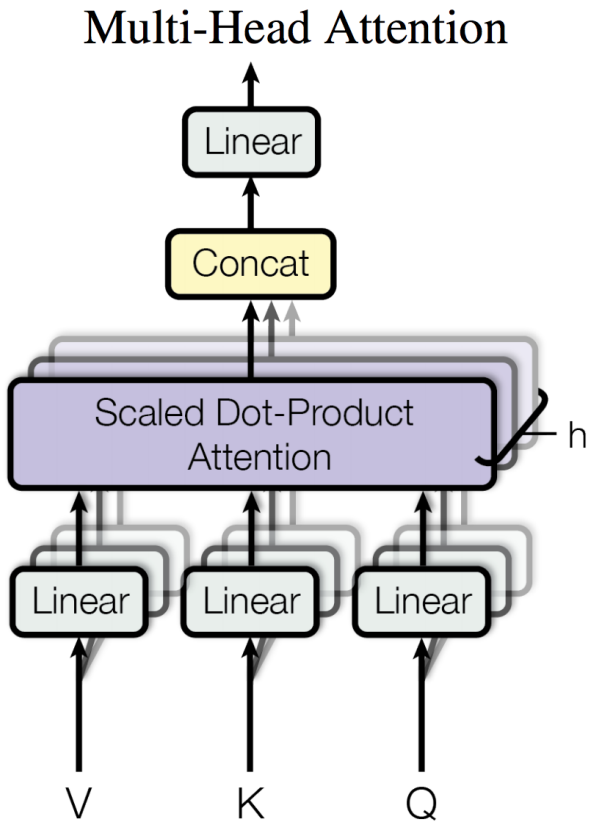
Convolution



Multi-Head Attention



Multi-Head Attention

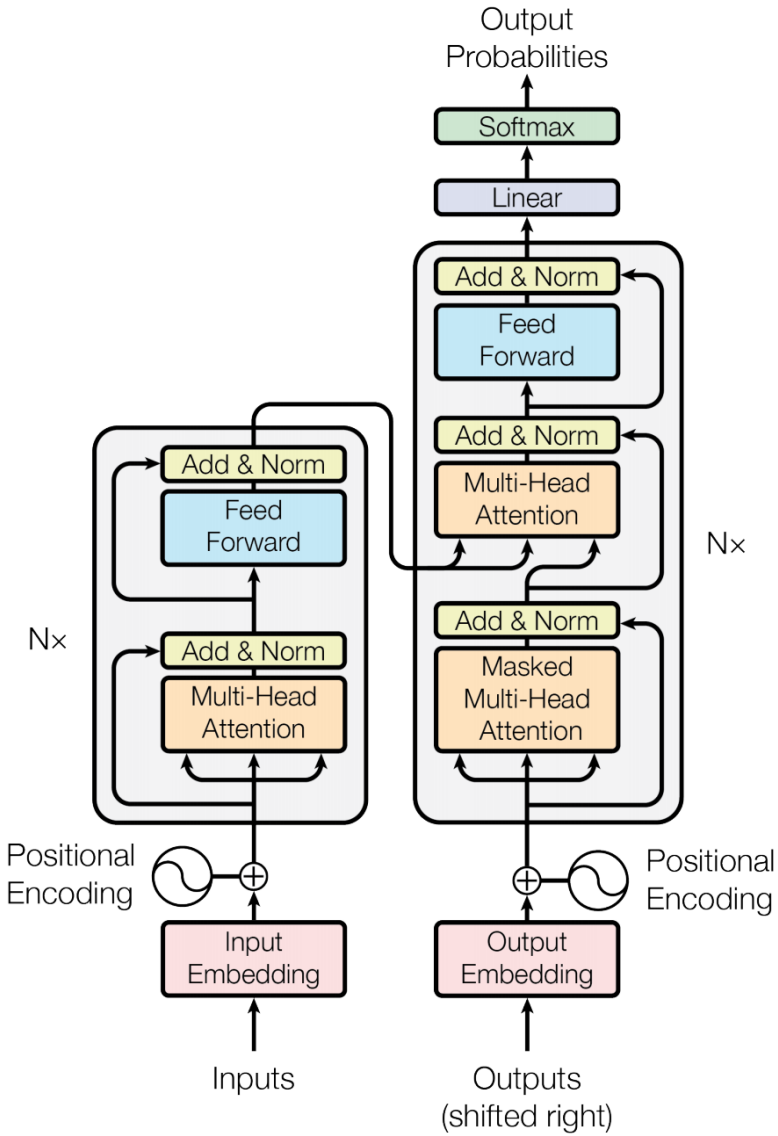


$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

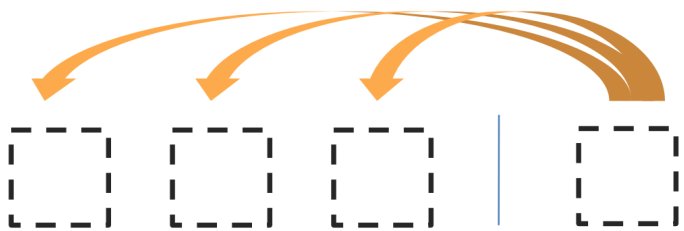
Transformer (Vaswani et al., 2017)

- Transformer is based solely on attention mechanisms, dispensing with recurrence and convolutions entirely.
- Encoder: 6 layers of self-attention + feedforward network
- Decoder: 6 layers of masked self-attention and output of encoder + feed-forward.

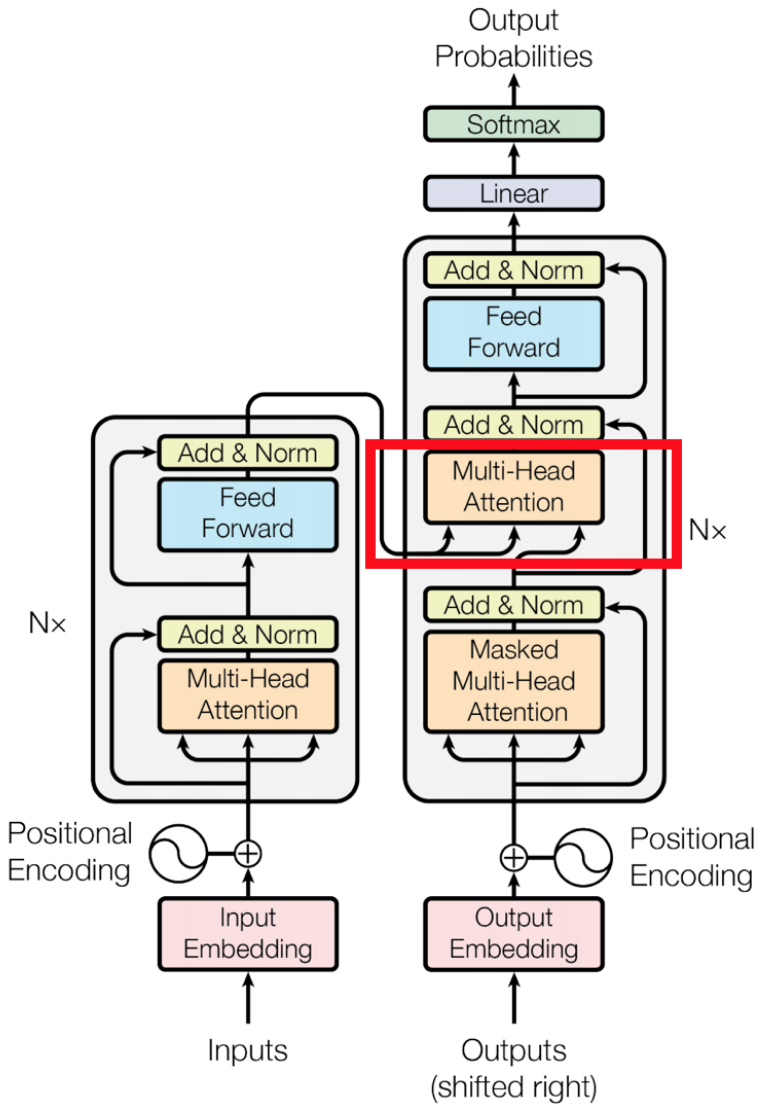


Three Ways of Attention

- Encoder-Decoder Attention: the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder.



Encoder-Decoder Attention

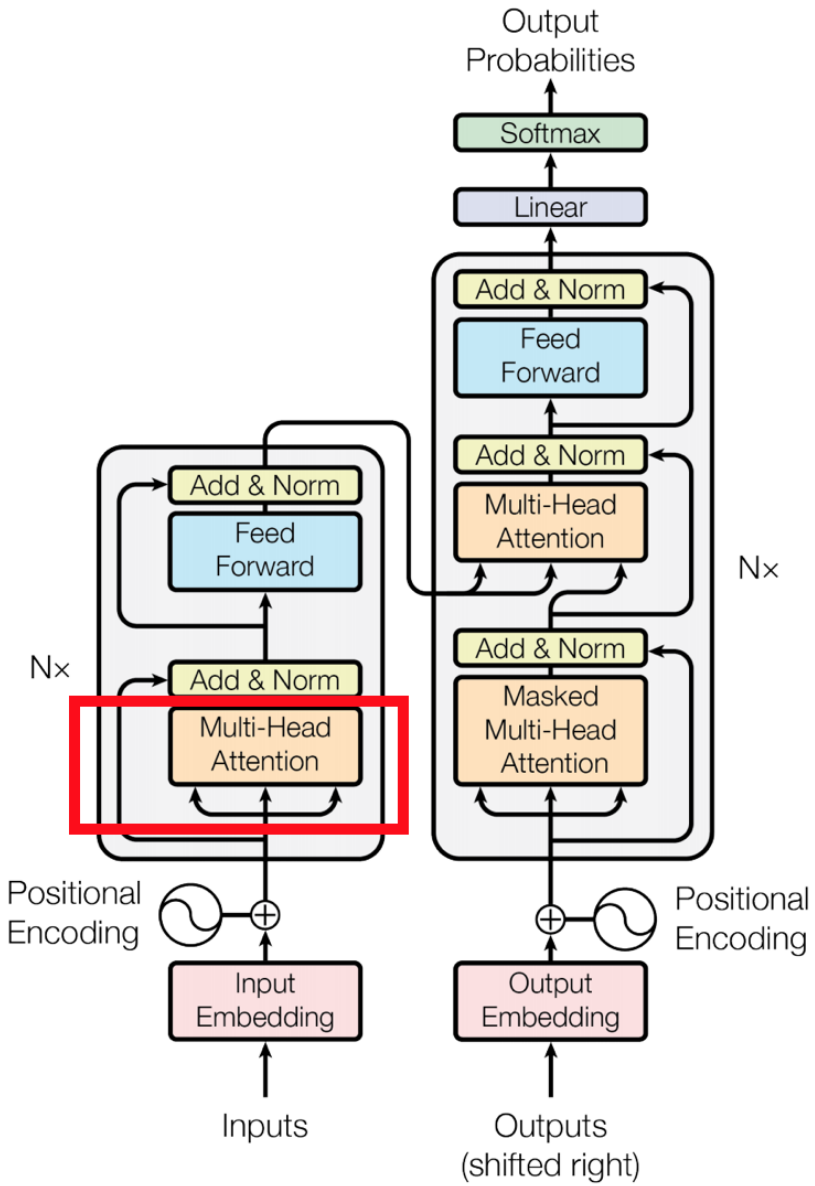


Three Ways of Attention

- Encoder Self-Attention: all of the keys, values and queries come from the same place, in this case, the output of the previous layer.

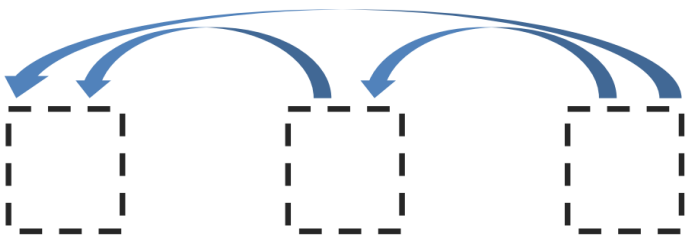


Encoder Self-Attention

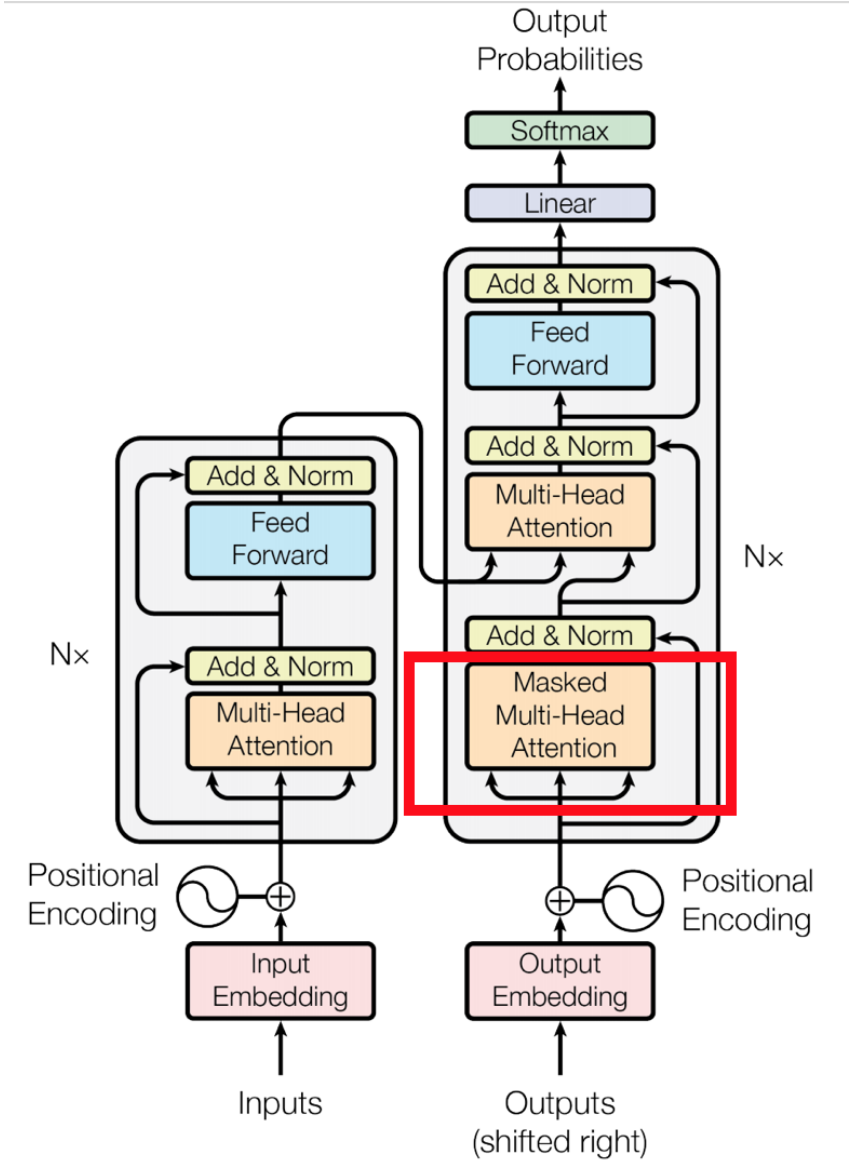


Three Ways of Attention

- Masked Decoder Self-Attention: each position in the decoder to attend to all positions in the decoder up to and including that position.



MaskedDecoder Self-Attention



Positional Encoding

- To make use of the order of the sequence, positional encoding provides relative or absolute position of given token.
- Many options to select positional encoding[2]. In Transformer,

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

  where d_{model} is the dimensionality of input. The authors hypothesized the function would allow the model to easily learn to attend by relative positions, since for any fixed k, PE_{pos+k} can be represented as a linear function of PE_{pos}.
- Alternative, to learn positional embeddings.

[2] Gehring et al., 2017. Convolutional sequence to sequence learning.

Results

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	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**	**41.8**	$2.3 \cdot 10^{19}$	

Model Variations

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

Attention Visualizations

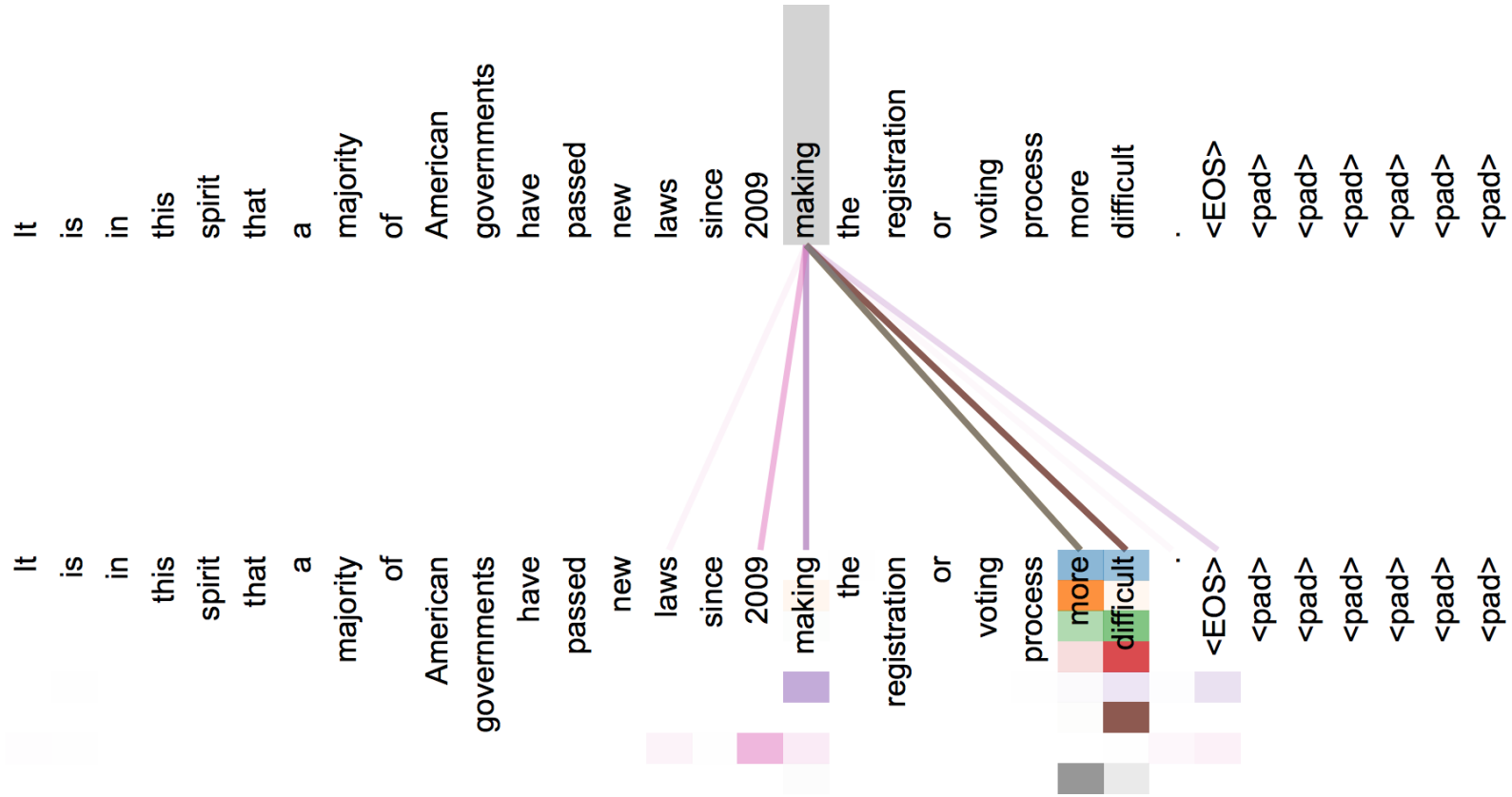
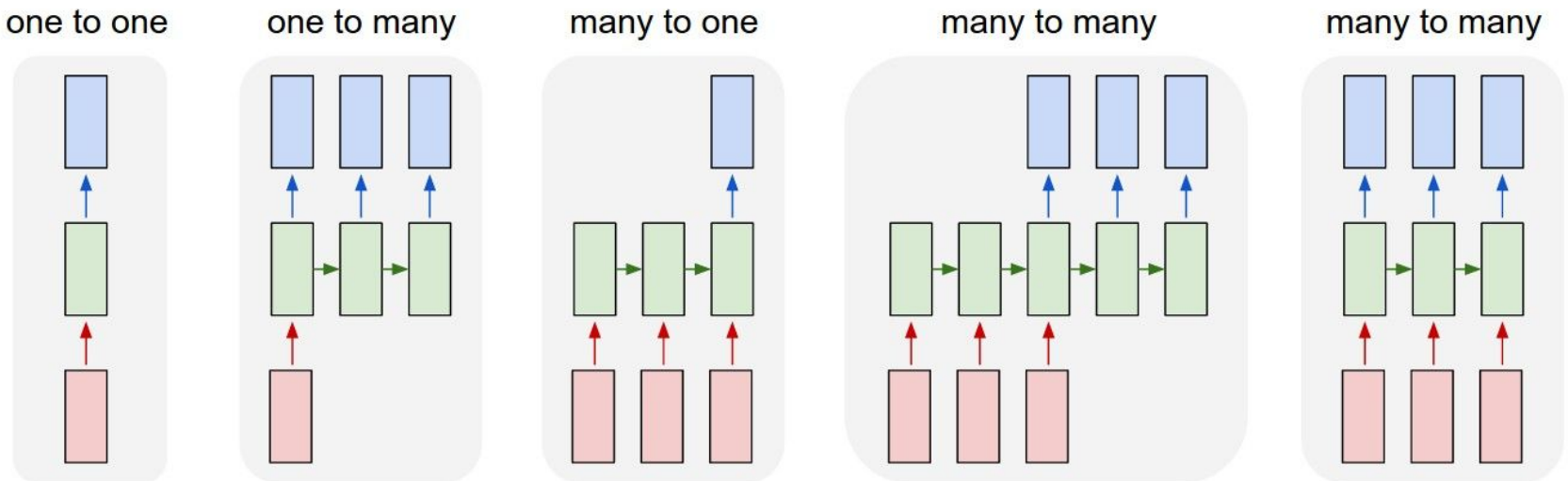


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colors represent different heads. Best viewed in color.

Overview

1. Introduction
2. Bidirectional RNN
3. Encoder-Decoder Architectures
4. Long-Term Dependencies
5. Attention as RNN
6. Pre-training of sequence model
7. Summary

Motivation



3

- Can we improve performances of diverse specific tasks via pre-training for sequence model?

[3] The example is from Fei-Fei Li's slides.

Word Representation Learning

- Learning a low-dimensional vector for each word (Word embedding).
- Using pre-trained word embedding as the input of sequence model.
- Notable work:
  - Word2vec (Mikolov et al.)
  - Glove (Pennington et al.)
- Problem: Can not solve word ambiguity.

ELMo: Embeddings from Language Models[4]

- Joint training the foward language and backward language model via LSTM.
- Using the concatenation of left-to-right and right-to-left LSTM to generate features for downstream tasks.

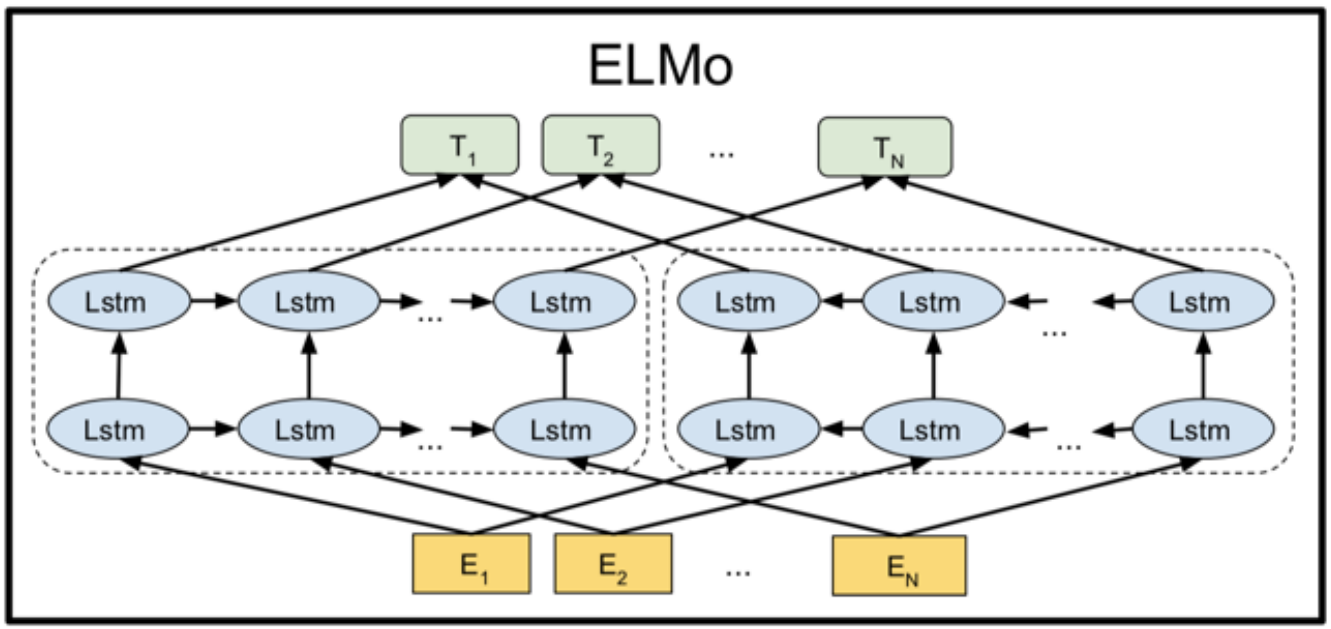


Figure 9: The architecture of ELMo.

[4]Peters M E, Neumann M, Iyyer M, et al. Deep contextualized word representations[J].

OpenAI GPT: Generative Pre-trained Transformer[5]

- Step 1: Pretraining a left-to-right transformer for the forward language model.
- Step 2: Fine-tuning the pre-trained transformer with both text prediction objective and task-specific objective.

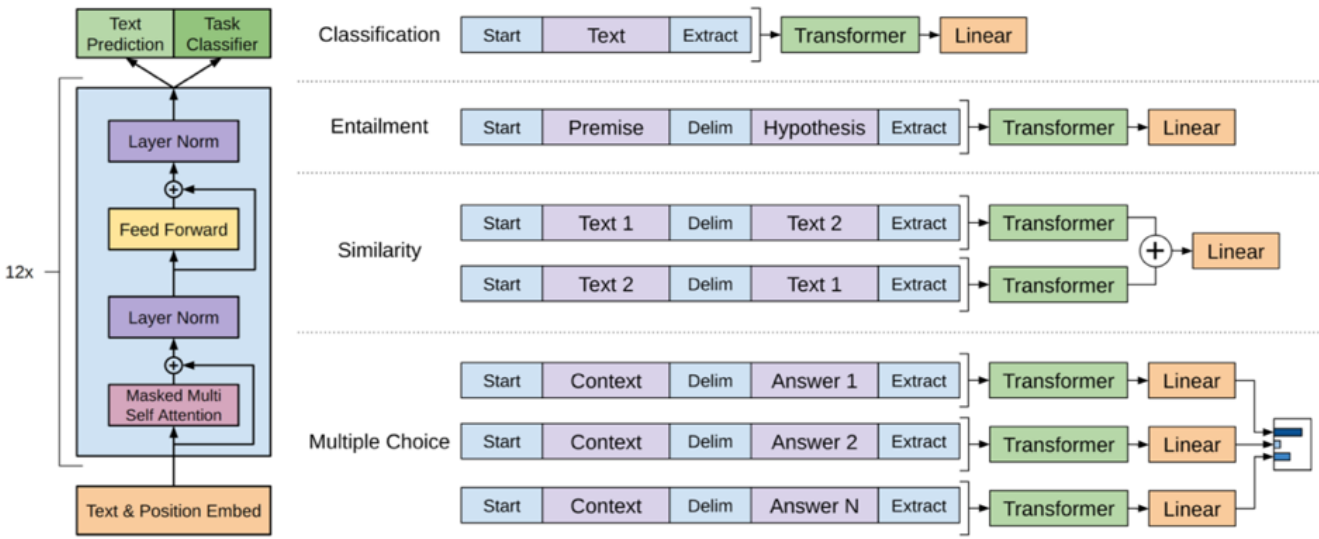


Figure 10: The architecture of OpenAI GPT.

[5]Radford A, Narasimhan K, Salimans T, et al. Improving language understanding by generative pre-training[J].

BERT: Pre-training of Deep Bidirectional Transformers[6]

- Limits of previous pre-training methods
  - Language models in pre-training are unidirectional, witch restrict the power of the pre-trained representations.
    - ELMo concatenates forward and backward language models.
    - OpenAI GPT used left-to-right architecture
- BERT:
  - masked language model
  - token prediction + sentence prediction

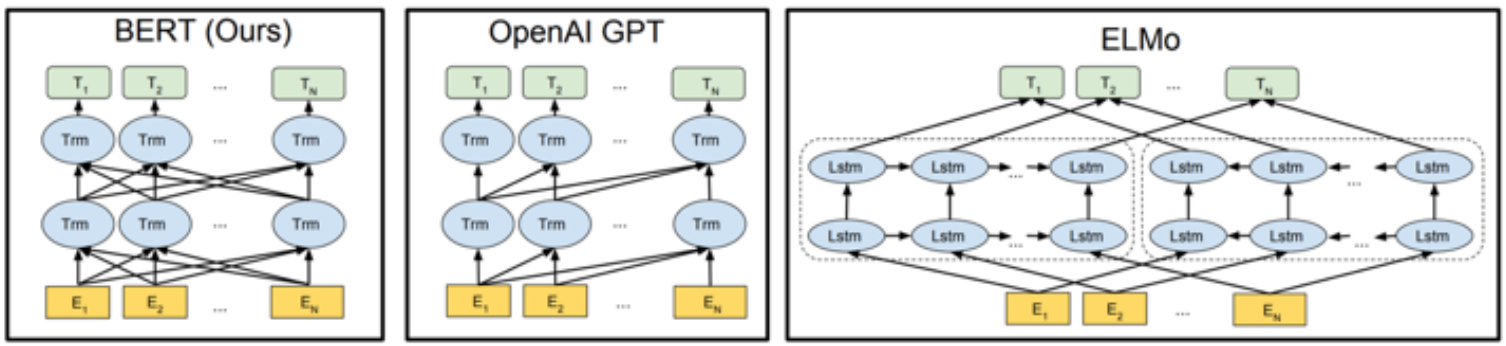


Figure 11: Differences in pre-training model architectures.

[6]Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J].

Input Representation of BERT

- Token + Segment + Position
  - Represent a single sentence or a pair of sentences (e.g. [Question, Answer]) in one token sequence

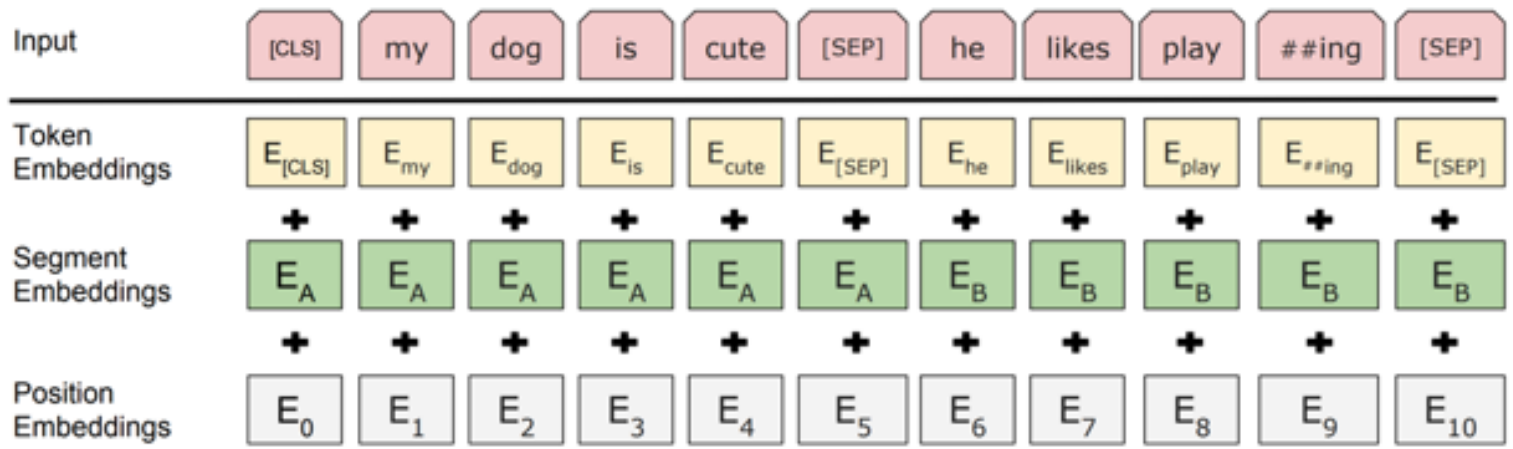


Figure 12: BERT input representation.

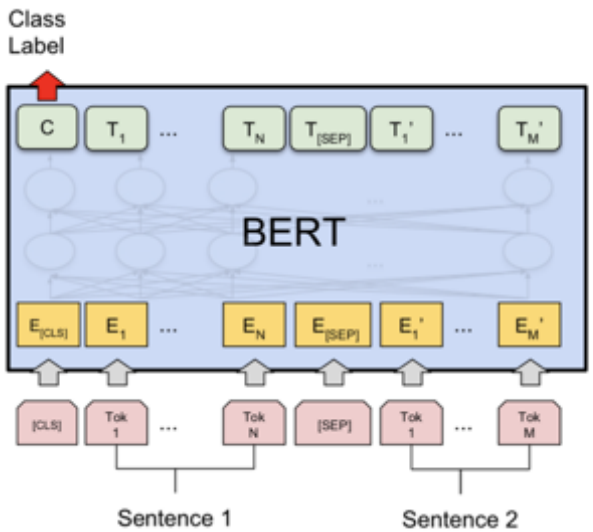
Pre-training Methods

- Task #1: Masked Language Model (Cloze Task)
- Mask some percentage (i.e. 15%) of the input tokens at random and then predict only those masked tokens
  - Downsides #1: [MASK] token is never seen during fine-tuning for specific tasks
    - Solution:
    - 80% of the time: Replace the word with the [MASK] token, e.g., my dog is hairy $\rightarrow$ my dog is [MASK]
    - 10% of the time: Replace the word with a random word e.g. my dog is hairy $\rightarrow$ my dog is apple
    - 10% of the time: Keep the word unchanged, e.g., my dog is hairy $\rightarrow$ my dog is hairy
  - Downsides #2: only 15% of tokens are predicted in each batch, which suggests that more pre-training steps may be required for converge

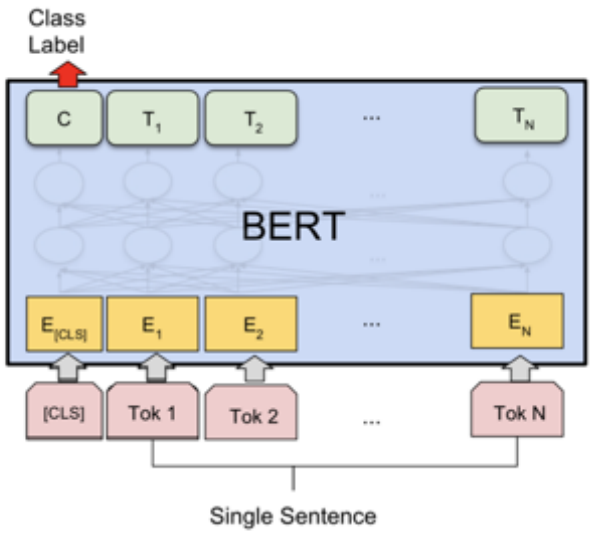
- Task #2: Next Sentence Prediction
  - Understanding the relationship between two sentences
    - When choosing sentences A and B, 50% of time B is the actual next sentence, 50% of time B is random sentence from the corpus
    - Beneficial to downstream tasks such as Question Answering (QA) and Natrual Language Inference (NLI)
  - Examples:
    - Input: [CLS] the man went to [MASK] store [SEP] he bought a gallon [mask] milk [SEP]
    - Label: IsNext
    - Input: [CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight ##less birds [SEP]
    - Label: NotNext

Fine-tuning Procedure

- Sequence-level classification
  - Take the final hidden state for the first token ([CLS]) the pooled representation of the input sequence
  - Add a classification layer
  - All of the parameters of BERT and the classification layer are fine-tuned jointly



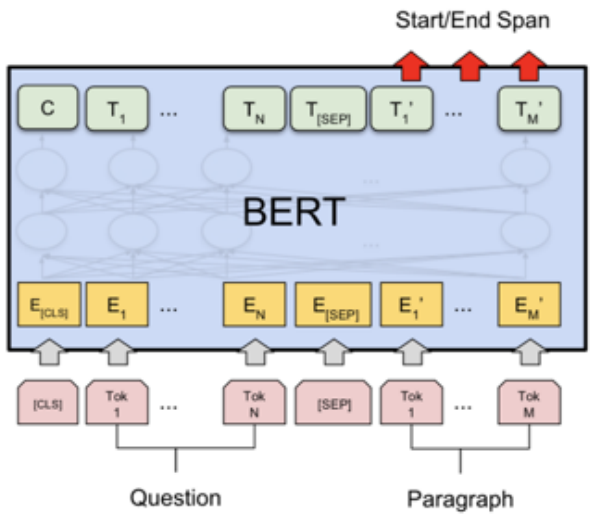
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



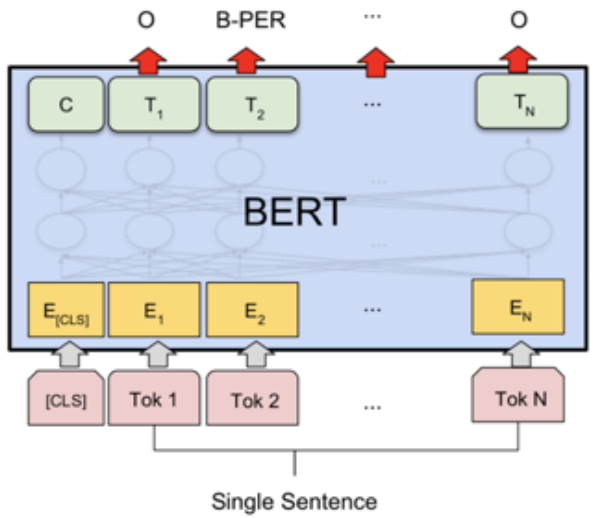
(b) Single Sentence Classification Tasks: SST-2, CoLA

Fine-tuning Procedure

- Sequence-level classification
  - Take the final hidden state for the first token ([CLS]) the pooled representation of the input sequence
  - Add a classification layer
  - All of the parameters of BERT and the classification layer are fine-tuned jointly



(c) Question Answering Tasks: SQuAD v1.1



(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Experiments-GLUE

- GLUE (General Language Understanding Evaluation) benchmark
- Results:

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**

Table 1: GLUE Test results, scored by the GLUE evaluation server. The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. OpenAI GPT = (L=12, H=768, A=12); $\text{BERT}_{\text{BASE}}$ = (L=12, H=768, A=12); $\text{BERT}_{\text{LARGE}}$ = (L=24, H=1024, A=16). BERT and OpenAI GPT are single-model, single task. All results obtained from https://gluebenchmark.com/leaderboard and https://blog.openai.com/language-unsupervised/.

Experiments-SQuAD

- Given a question and a paragraph from Wikipedia containing the answer, the task is to predict the answer text span in the paragraph.

- Input Question:

  `Where do water droplets collide with ice crystals to form precipitation?`

- Input Paragraph:

  `... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals within a cloud. ...`

- Output Answer:

  `within a cloud`

- Results:

System	Dev EM	Dev F1	Test EM	Test F1
Leaderboard (Oct 8th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
#1 Single - nlnet	-	-	83.5	90.1
#2 Single - QANet	-	-	82.5	89.3
Published				
BiDAF+ELMo (Single)	-	85.8	-	-
R.M. Reader (Single)	78.9	86.3	79.5	86.6
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
$\text{BERT}_{\text{BASE}}$ (Single)	80.8	88.5	-	-
$\text{BERT}_{\text{LARGE}}$ (Single)	84.1	90.9	-	-
$\text{BERT}_{\text{LARGE}}$ (Ensemble)	85.8	91.8	-	-
$\text{BERT}_{\text{LARGE}}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
$\text{BERT}_{\text{LARGE}}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

Table 2: SQuAD results. The BERT ensemble is 7x systems which use different pre-training checkpoints and fine-tuning seeds.

Experiments—NER, SWAG

- Named Entity Recognition (NER)
  - Recognize named entities (i.e. Person, Organization, Location) from a given sentence
- Results:

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**

Table 3: CoNLL-2003 Named Entity Recognition results. The hyperparameters were selected using the Dev set, and the reported Dev and Test scores are averaged over 5 random restarts using those hyperparameters.

- Situations With Adversarial Generations (SWAG)
  - Given a sentence from a sentence from a video captioning dataset, the task is to decide among four choices the most plausible continuation.
- Results:

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 4: SWAG Dev and Test accuracies. Test resul were scored against the hidden labels by the SWAG a thors. †Human performance is measure with 100 sar ples, as reported in the SWAG paper.

Ablation Study — NSP and MLM

- Main findings:
  - Next Sentence prediction (NSP) pre-training is important for sentence-pair classification task
  - Masked Language Model (MLM) contributes significantly to the performance improvement

Tasks	Dev Set MNLI-m (Acc)	QNLI (Acc)	MRPC (Acc)	SST-2 (Acc)	SQuAD (F1)
$\text{BERT}_{\text{BASE}}$	84.4	88.4	86.7	92.7	88.5
No NSP	83.9	84.9	86.5	92.6	87.9
LTR & No NSP	82.1	84.3	77.5	92.1	77.8
+ BiLSTM	82.1	84.1	75.7	91.6	84.9

Table 5: Ablation over the pre-training tasks using the $\text{BERT}_{\text{BASE}}$ architecture. "No NSP" is trained without the next sentence prediction task. "LTR & No NSP" is trained as a left-to-right LM without the next sentence prediction, like OpenAI GPT. "+ BiLSTM" adds a randomly initialized BiLSTM on top of the "LTR + No NSP" model during fine-tuning.

Ablation Study — Model size

- Main findings:
  - Bigger model sizes are better even for small-scale tasks

Hyperparams				Dev Set Accuracy		
#L	#H	#A	LM (ppl)	MNLI-m	MRPC	SST-2
3	768	12	5.84	77.9	79.8	88.4
6	768	3	5.24	80.6	82.2	90.7
6	768	12	4.68	81.9	84.8	91.3
12	768	12	3.99	84.4	86.7	92.9
12	1024	16	3.54	85.7	86.9	93.3
24	1024	16	3.23	86.6	87.8	93.7

Table 6: Ablation over BERT model size. #L = the number of layers; #H = hidden size; #A = number of attention heads. "LM (ppl)" is the masked LM perplexity of held-out training data.

Recent advances—GPT-2[7]

- Applying left-to-right transformer on more larger dataset(1.5 billion parameters, 8 million web pages).
- GPT-2 achieves state-of-the-art scores on a variety of domain-specific language modeling tasks with "zero-shot setting".

Language Models are Unsupervised Multitask Learners

	LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)	PTB (PPL)	enwik8 (BPB)	text8 (BPC)	WikiText103 (PPL)	1BW (PPL)
SOTA	99.8	59.23	85.7	82.3	39.14	46.54	0.99	1.08	18.3	**21.8**
117M	**35.13**	45.99	**87.65**	**83.4**	**29.41**	65.85	1.16	1.17	37.50	75.20
345M	**15.60**	55.48	**92.35**	**87.1**	**22.76**	47.33	1.01	**1.06**	26.37	55.72
762M	**10.87**	**60.12**	**93.45**	**88.0**	**19.93**	**40.31**	**0.97**	**1.02**	22.05	44.575
1542M	**8.63**	**63.24**	**93.30**	**89.05**	**18.34**	**35.76**	**0.93**	**0.98**	**17.48**	42.16

Table 3. Zero-shot results on many datasets. No training or fine-tuning was performed for any of these results. PTB and WikiText-2 results are from (Gong et al., 2018). CBT results are from (Bajgar et al., 2016). LAMBADA accuracy result is from (Hoang et al., 2018) and LAMBADA perplexity result is from (Grave et al., 2016). Other results are from (Dai et al., 2019).

Figure 13: Results of GPT-2

[7] https://openai.com/blog/better-language-models/

Recent advances—ERNIE[8]

- BERT processes Chinese text in character level. It is difficult to learn complete semantic representation for concept.
- ERNIE learns the semantic representation of complete concepts by masking semantic units such as words and entities.

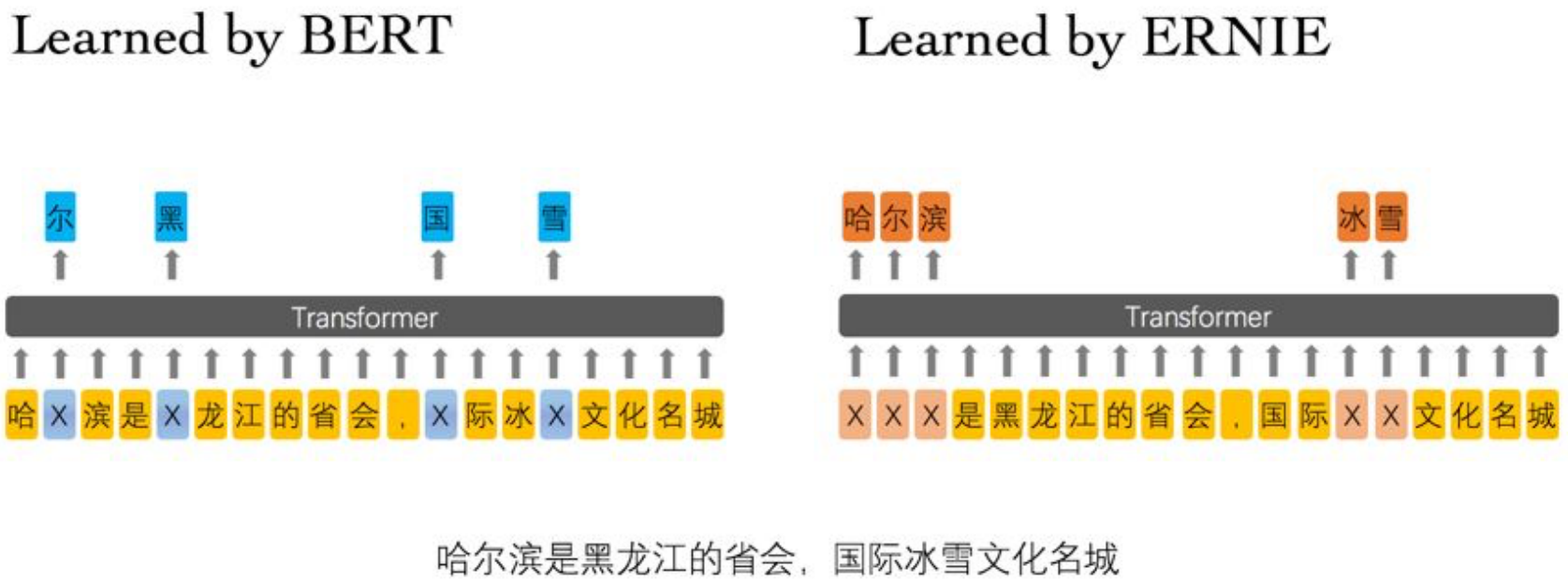
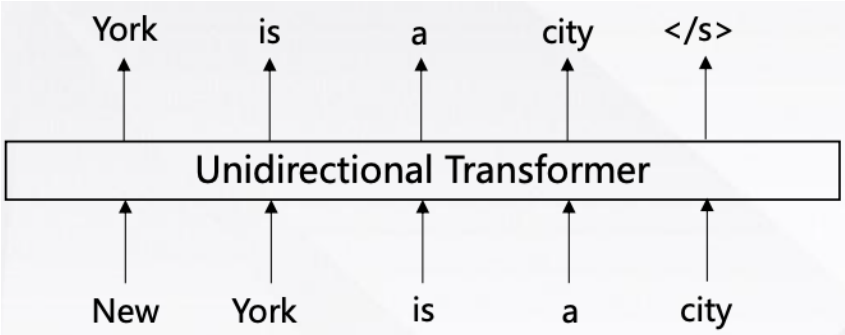


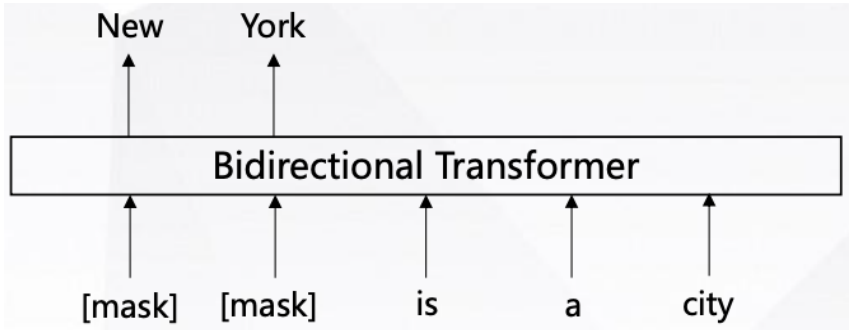
Figure 14: The differences between BERT and ERNIE.

[8]https://github.com/PaddlePaddle/LARK/tree/develop/ERNIE

Recent advance—XLNet[9]



- Auto-regressive language modeling
- $\log p(\mathbf{x}) = \sum_{t=1}^{T} \log p\left(x_t | \mathbf{x}_{<t}\right)$
- Not able to model bidirectional context.



- (Denoising) Auto-encoding
- $\log p(\overline{\mathbf{x}} | \hat{\mathbf{x}}) = \sum_{t=1}^{T} \text{mask}_t \log p\left(x_t | \hat{\mathbf{x}}\right)$
- Predicted tokens are independent of each other.
- [mask] is not used during finetuning.

[9] https://arxiv.org/abs/1906.08237

XLNet: Permutation Language Modeling

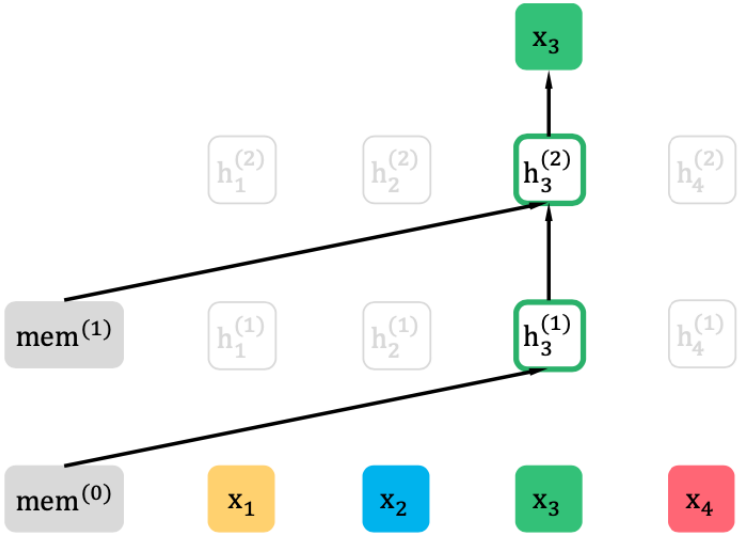
- Sample a factorization order
- Determine the attention masks based on the order
- Optimize a standard language modeling objective

$$\mathbb{E}_{\mathbf{z}\sim\mathcal{Z}_T}\left[\sum_{t=1}^{T}\log p\left(x_{z_t}|\mathbf{x}_{\mathbf{z}<t}\right)\right]$$

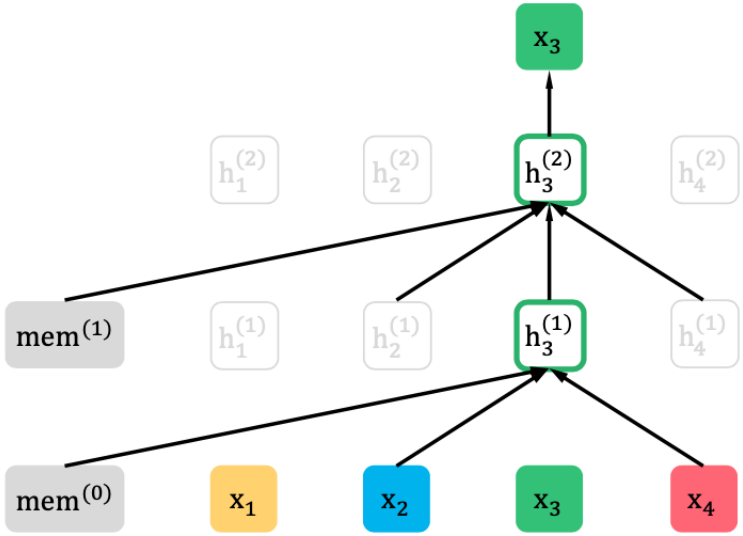
- Benefits:
  - Able to model bidirectional context
  - Autoregressive, avoiding disadvantages of AE

XLNet: Examples

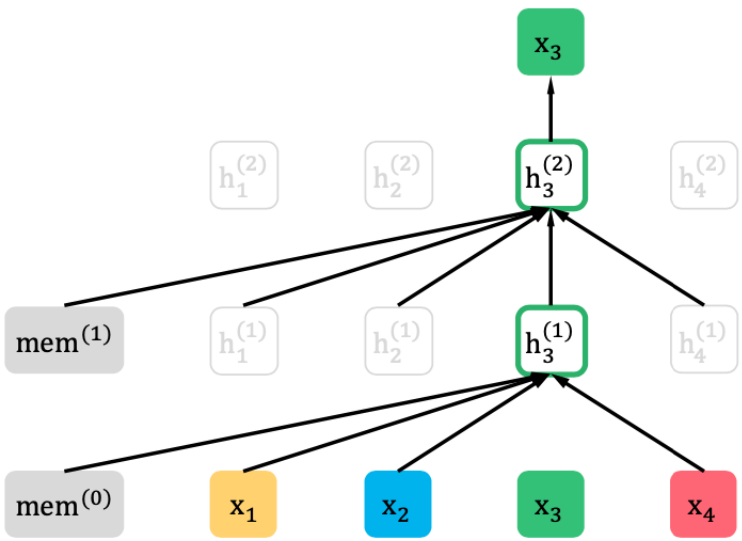
- Factorization order: New York is a city
  $P(\text{New York is a city}) = P(\text{New}) * P(\text{York}|\text{New}) *$
  $P(\text{is}|\text{New York}) * P(\text{a}|\text{New York is}) * P(\text{city}|\text{New York is a})$
- Factorization order: city a is New York
  $P(\text{New York is a city}) = P(\text{city}) * P(\text{a}|\text{city}) * P(\text{is}|\text{city a}) *$
  $P(\text{New}|\text{city a is}) * P(\text{York}|\text{city a is New})$
  - Sequence order is not shuffled.
  - Attention masks are changed to reflect factorization order.



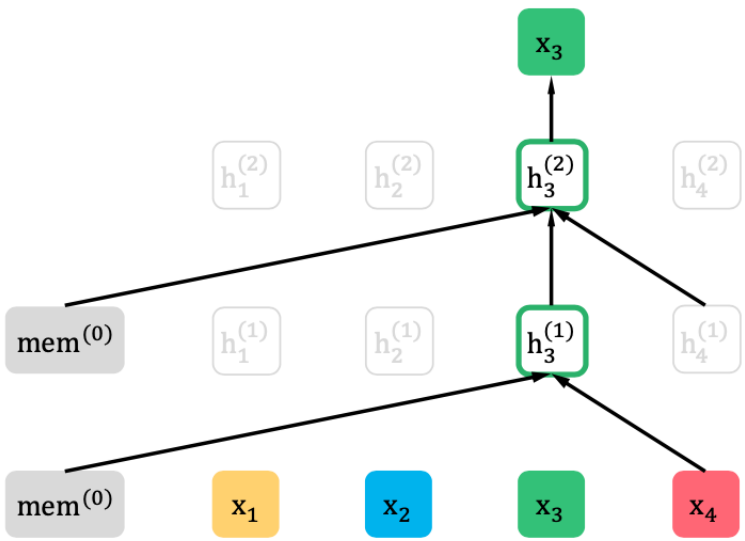
Factorization order: 3 → 2 → 4 → 1



Factorization order: 2 → 4 → 3 → 1



Factorization order: 1 → 4 → 2 → 3



Factorization order: 4 → 3 → 1 → 2

Comparing XLNet and BERT objectives

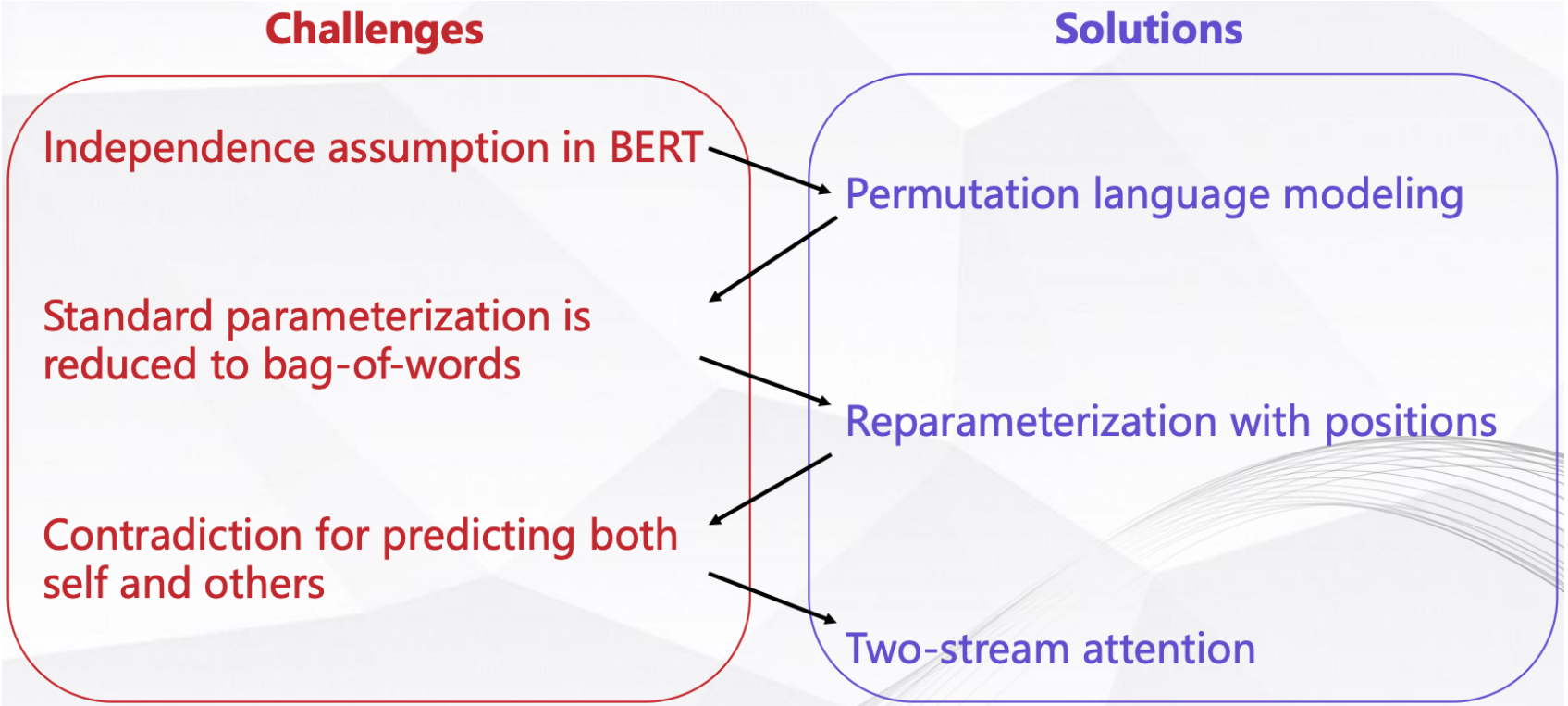
- BERT objective (auto-encoding)
  - $\mathcal{J}_{\mathrm{BERT}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{is a city})$
  - New and York are independent.
- XLNet objective (auto-regressive)
  - $\mathcal{J}_{\mathrm{XLNet}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{New, is a city})$
  - or

    $\mathcal{J}_{\mathrm{XLNet}} = \log p(\text{New} \mid \text{York, is a city}) + \log p(\text{York} \mid \text{is a city})$
  - Able to model dependency between New and York.
  - Able to model bidirectional context.
  - Factorize the joint probability using a product rule that holds universally.

XLNet: Reparameterization

- Standard Parameterization
  - $p_\theta\left(X_{z_t} = x | \mathbf{x}_{\mathbf{z}<t}\right) = \frac{e(x)^\top h_\theta(\mathbf{x}_{\mathbf{z}<t})}{\sum_{x'} e(x')^\top h_\theta(\mathbf{x}_{\mathbf{z}<t})}$
  - h does not contain the position of the target.
  - $p\left(X_3 = \text{ is} \mid \text{New York}\right) = p\left(X_4 = \text{ is} \mid \text{New York}\right) = p\left(X_5 = \text{ is} \mid \text{New York}\right)$
  - Reduced to predicting a bag of words.
- Solution: condition the distribution on the position.
  - $p_\theta\left(X_{z_t} = x | \mathbf{x}_{\mathbf{z}<t}\right) = \frac{e(x)^\top g_\theta(\mathbf{x}_{\mathbf{z}<t}, \mathbb{Z}_t)}{\sum_{x'} e(x')^\top g_\theta(\mathbf{x}_{\mathbf{z}<t}, \bar{z}_t)}$

Summarizing XLNet



Overview

1. Introduction
2. Bidirectional RNN
3. Encoder-Decoder Architectures
4. Long-Term Dependencies
5. Attention as RNN
6. Pre-training of sequence model
7. Summary

Summary

- RNN is able to model sequential data with lots of flexibilities
- RNN can be also extended to model bidirectional dependencies
- LSTM is a special RNN and can be used to model long-term dependencies
- Attention as RNN: Attention is All you Need (Vaswani et al., 2017).
- Pretraining of sequence model: BERT and XLNet take NLP to much higher accuracy

Thanks.

HP: http://keg.cs.tsinghua.edu.cn/jietang/
Course HP: https://www.aminer.cn/aml
Email: jietang@tsinghua.edu.cn