Sequence Modeling: Recurrent and Recursive Nets

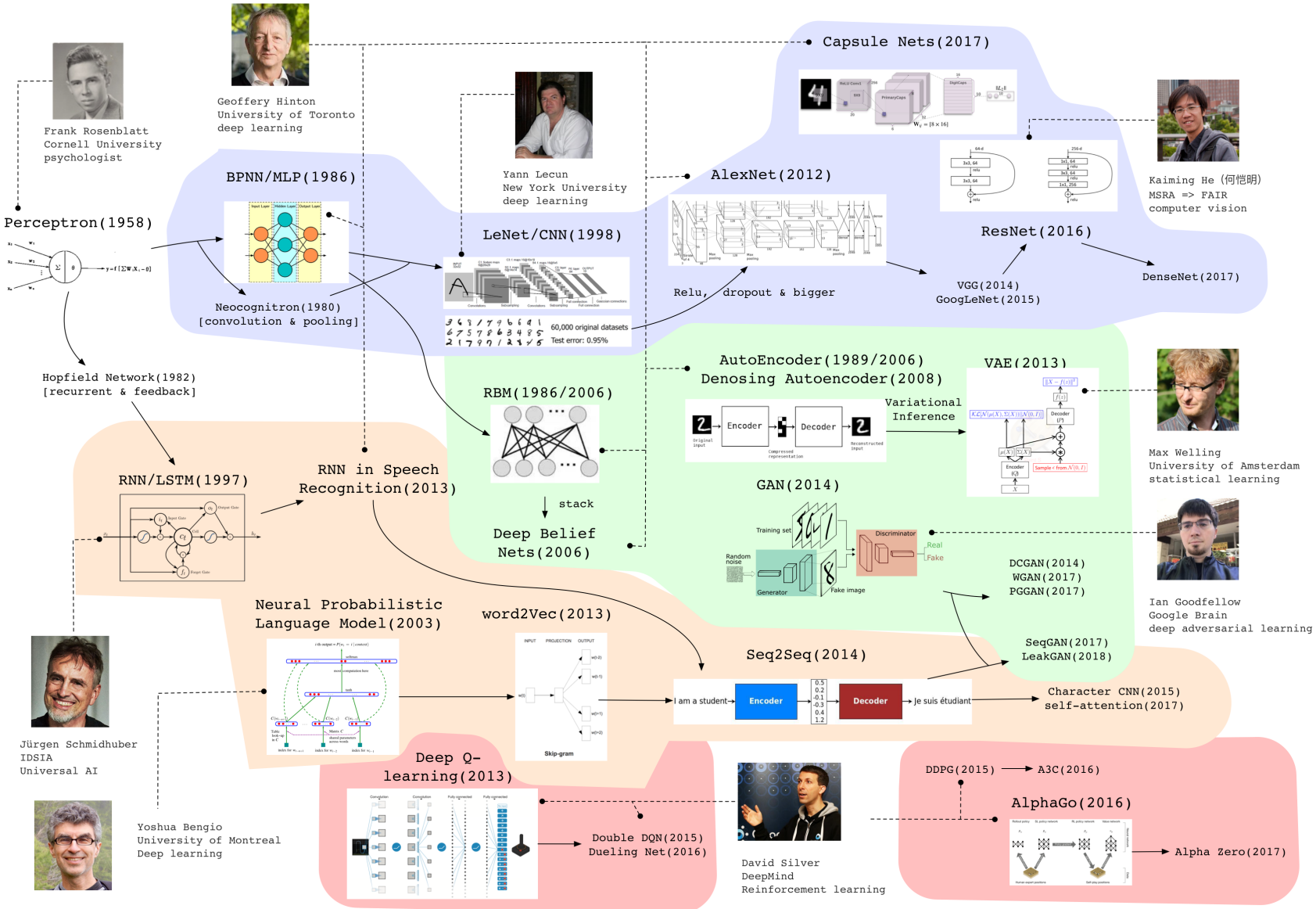
Jie Tang

Tsinghua University

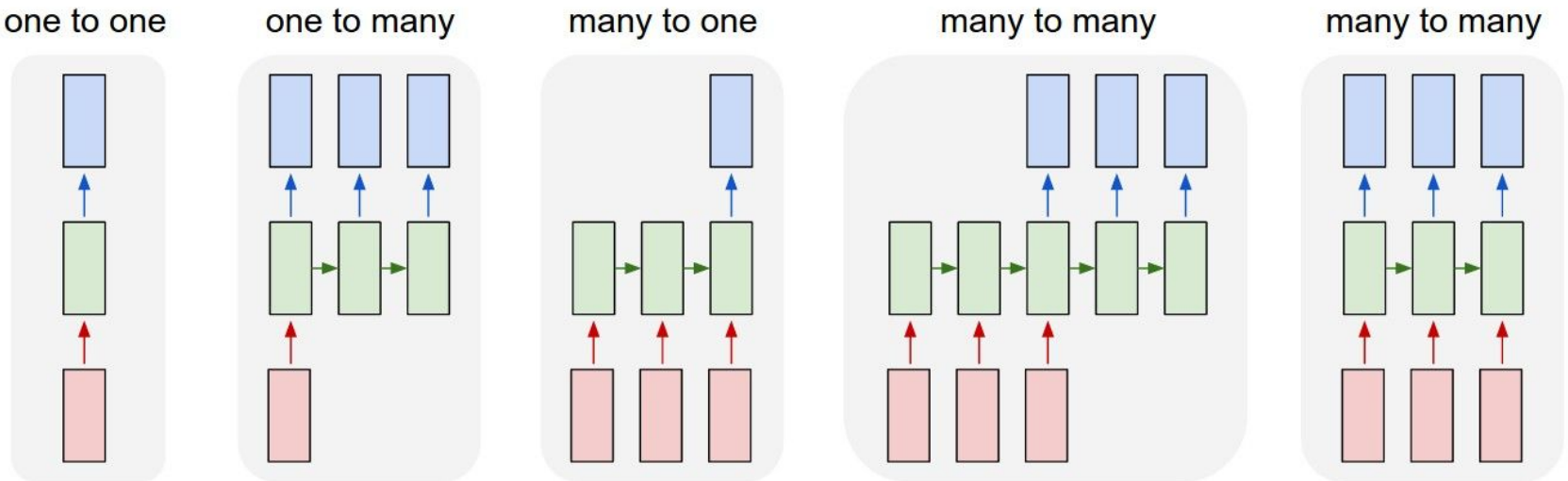
Overview

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.


Capsule Nets(2017)
Geoffery Hinton
University of Toronto
deep learning
Frank Rosenblatt
Cornell University
psychologist
Yann Lecun
New York University
deep learning
Kaiming He (何恺明)
MSRA => FAIR
computer vision
BPNN/MLP(1986)
AlexNet(2012)
Perceptron(1958)
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)
Encoder
Decoder
Variational
Inference
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)
Seq2Seq(2014)
SeqGAN(2017)
LeakGAN(2018)
I am a student
Encoder
Decoder
Je suis étudiant
Character CNN(2015)
self-attention(2017)
Skip-gram
Jürgen Schmidhuber
IDSIA
Universal AI
Deep Q-
learning(2013)
Yoshua Bengio
University of Montreal
Deep learning
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...

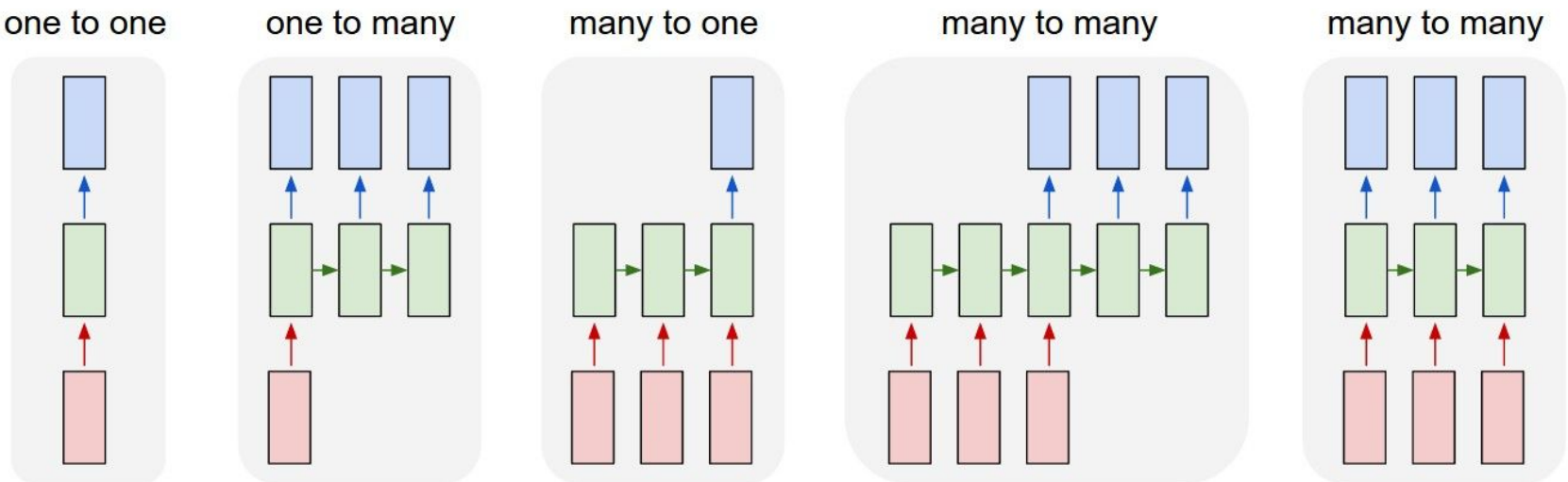


[1]

Simple neural network

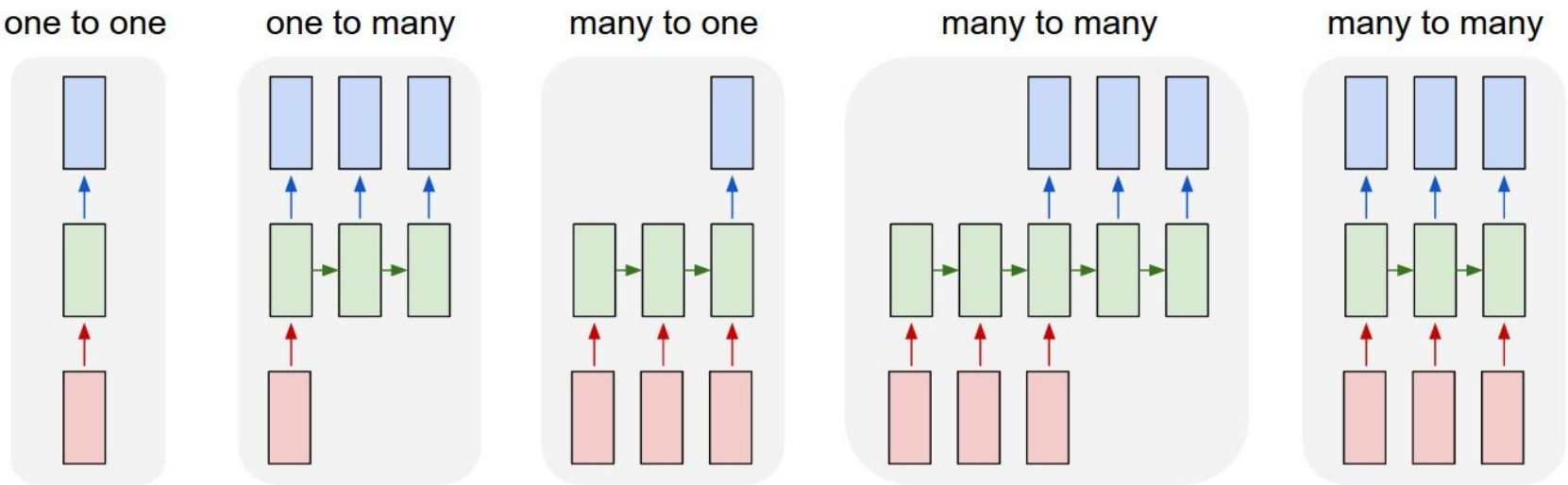
[1] The example is from Fei-Fei Li's slides.

Recurrent Networks offer a lot of flexibility...



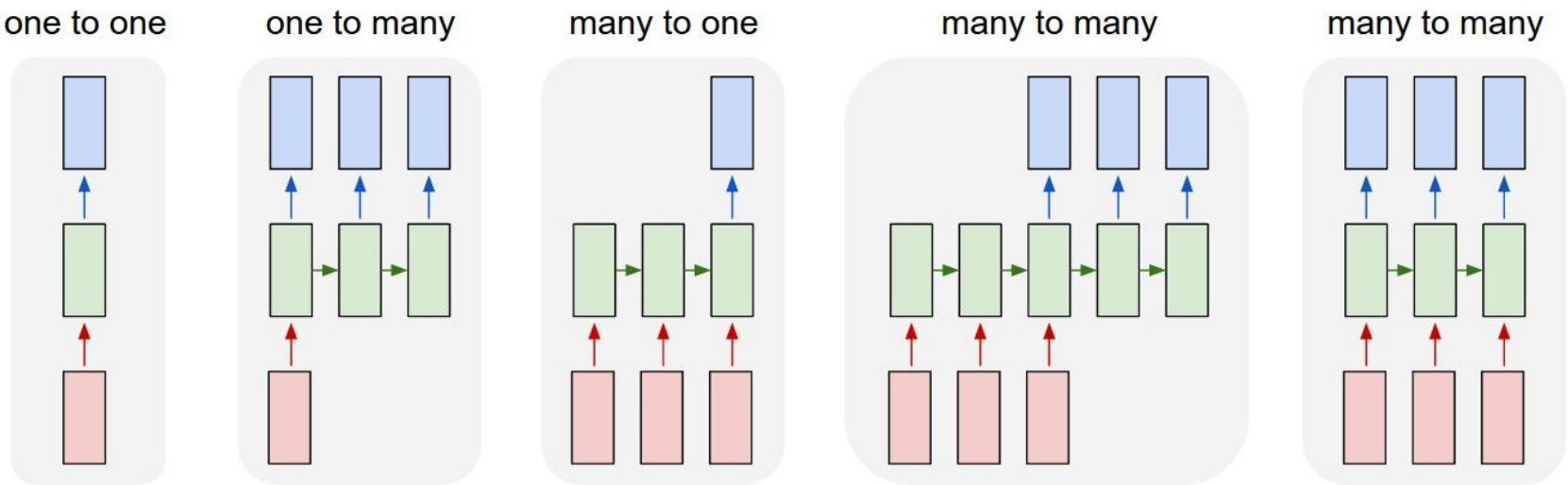
e.g., Image captioning: image$\rightarrow$ sequence-of-words

Recurrent Networks offer a lot of flexibility...



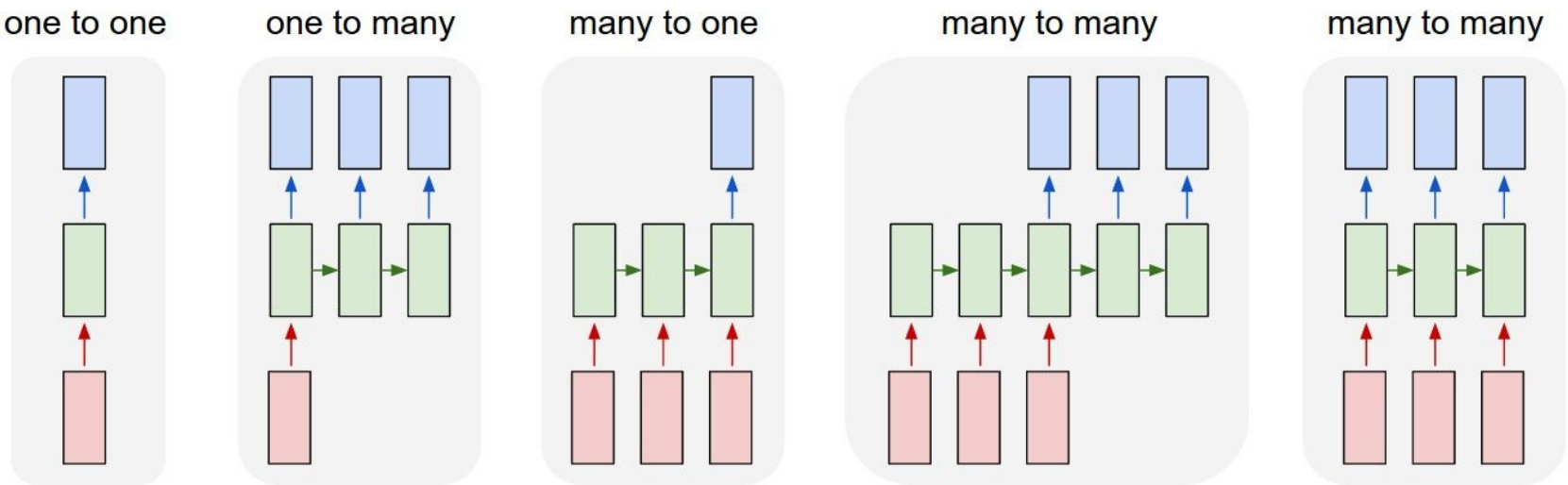
e.g., Sentiment analysis: sequence-of-words $\rightarrow$ sentiment

Recurrent Networks offer a lot of flexibility...



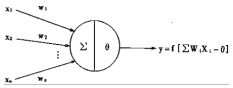
e.g., Machine translation: sequence $\rightarrow$ sequence

Recurrent Networks offer a lot of flexibility...



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

Frank Rosenblatt
Cornell University
psychologist

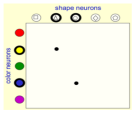


Perceptron(1958)

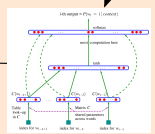
Hopfield Network(1982)
[recurrent & feedback]

Yoshua Bengio
University of Montreal
Deep learning

Geoffery Hinton
University of Toronto
deep learning

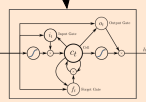


Distributed
representation(1986)

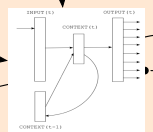


Neural Probabilistic
Language Model(2003)

Jürgen Schmidhuber
IDSIA
Universal AI

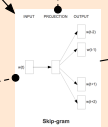


LSTM(1997)

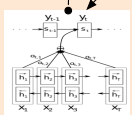


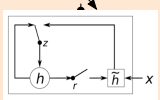
Tomas Mikolov
Google -> Facebook
Deep learning
NLP

RNN based language
model(2010)



Deepwalk (2014)
Network embedding





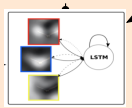
Seq2Seq
(2014.9.10)

attention seq2seq
(2014.9.1)

GRU & seq2seq
(2014,6.3)

word2Vec(2013)

Glove (2014)
count-based word
embedding



"show attend and
tell"(2015)

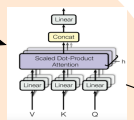
ByteNet (2016)
[convolution]

Nal Kalchbrenner
deepmind
deeplearning

Ilya Sutskever
Google -> openAI
Deep learning

context2vec(2016)
contextualized embedding

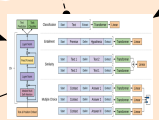
ResNet (2016)



Conv Seq2seq (2017)
Facebook

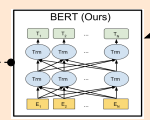
Transformer
(2017)
Google AI

ELMo(2018)
dynamic embedding



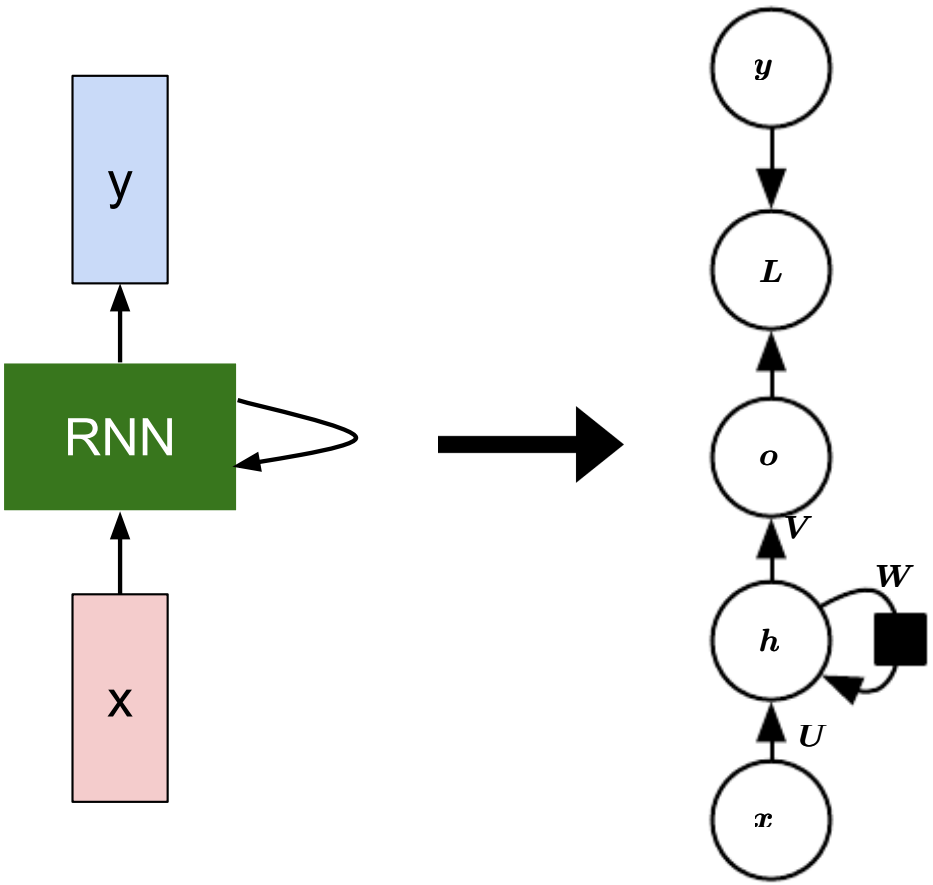
OpenAI GPT(2018)
pretraining

Jacob Devlin
microsoft -> google AI
Machine Translation



BERT(2018)

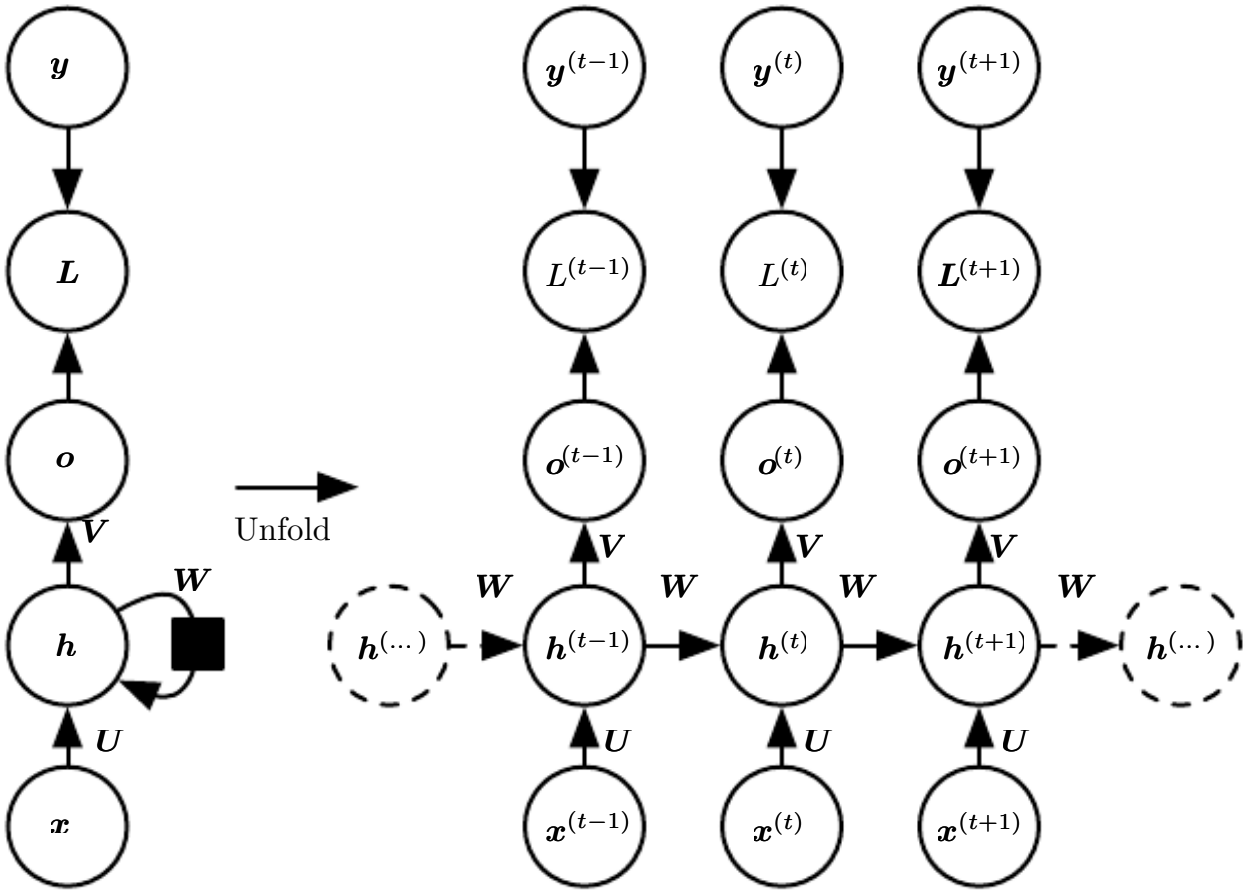
Recurrent Neural Networks — An Example



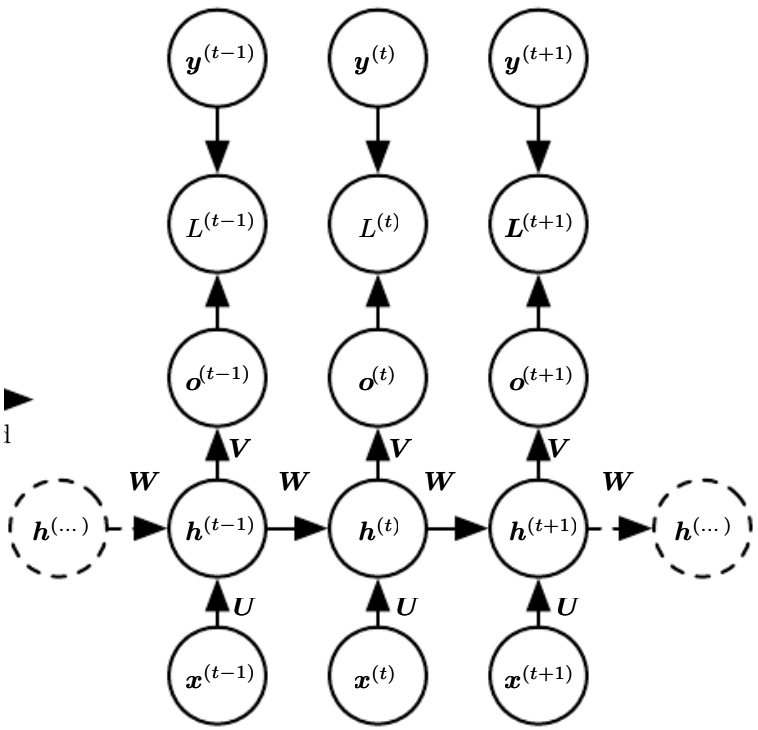
- Input x and output y are from the training data;
- o is the estimated output value;
- L is to evaluate the difference (loss) between the estimated output $\hat{y} = softmax(o)$ and the true output y;
- U, W, V are three weigh matrices.

Recurrent Neural Networks — An Example

- Let us unfold the graphical representation.



Recurrent Neural Networks — An Example



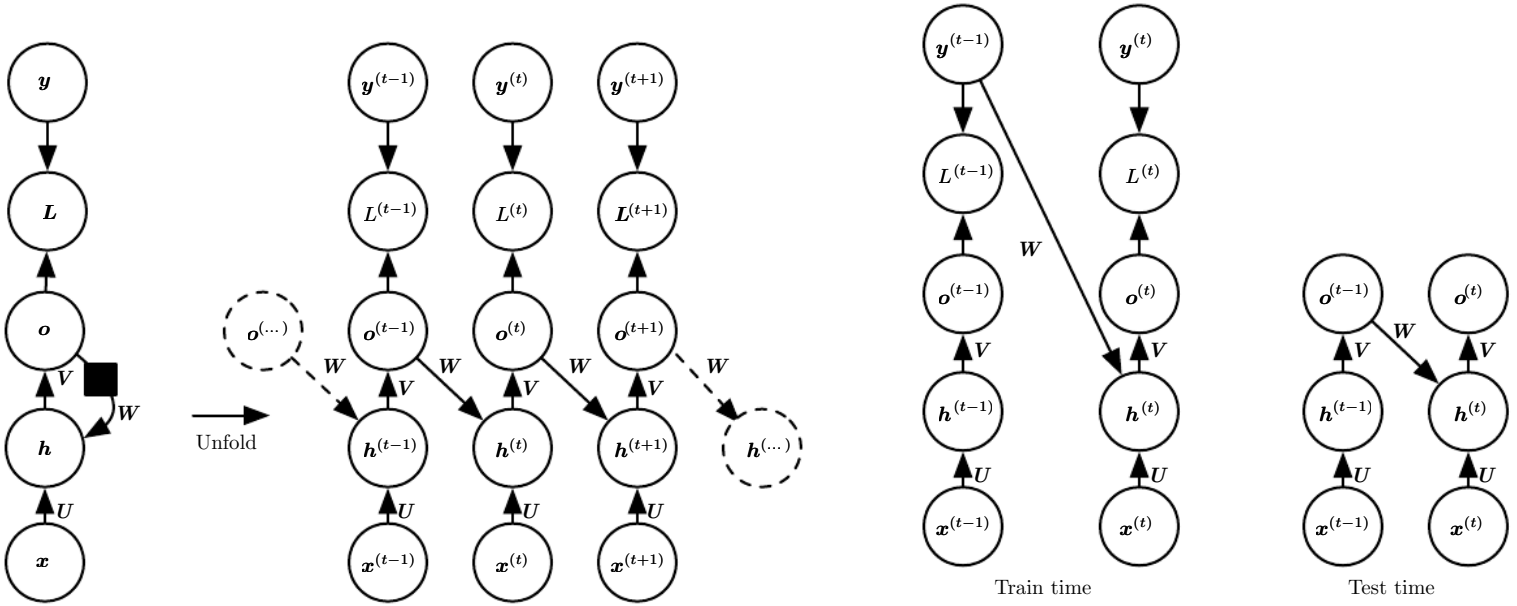
Update Equations

- $\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)})$
- $\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}$
- $\hat{\boldsymbol{y}}^{(t)} = \text{softmax}(\boldsymbol{o}^{(t)})$
- 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

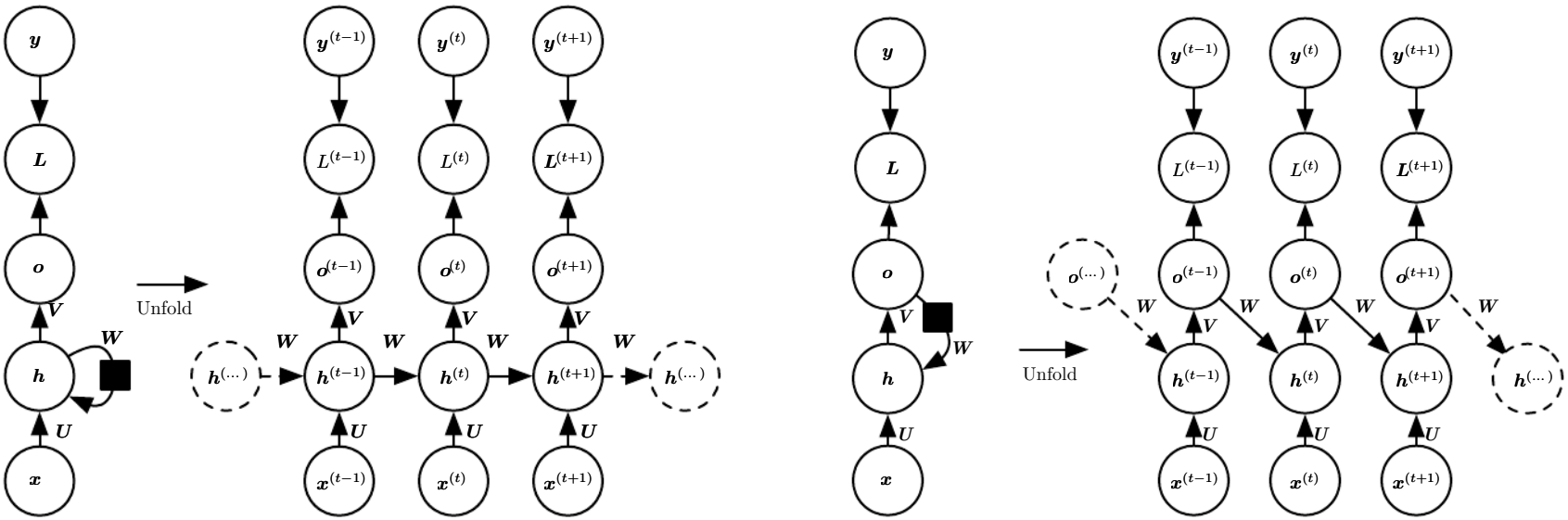
- randomly choose to use generated values or actual values as input

Gradient Computation in RNN

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

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

RNN as Directed Graphical Models

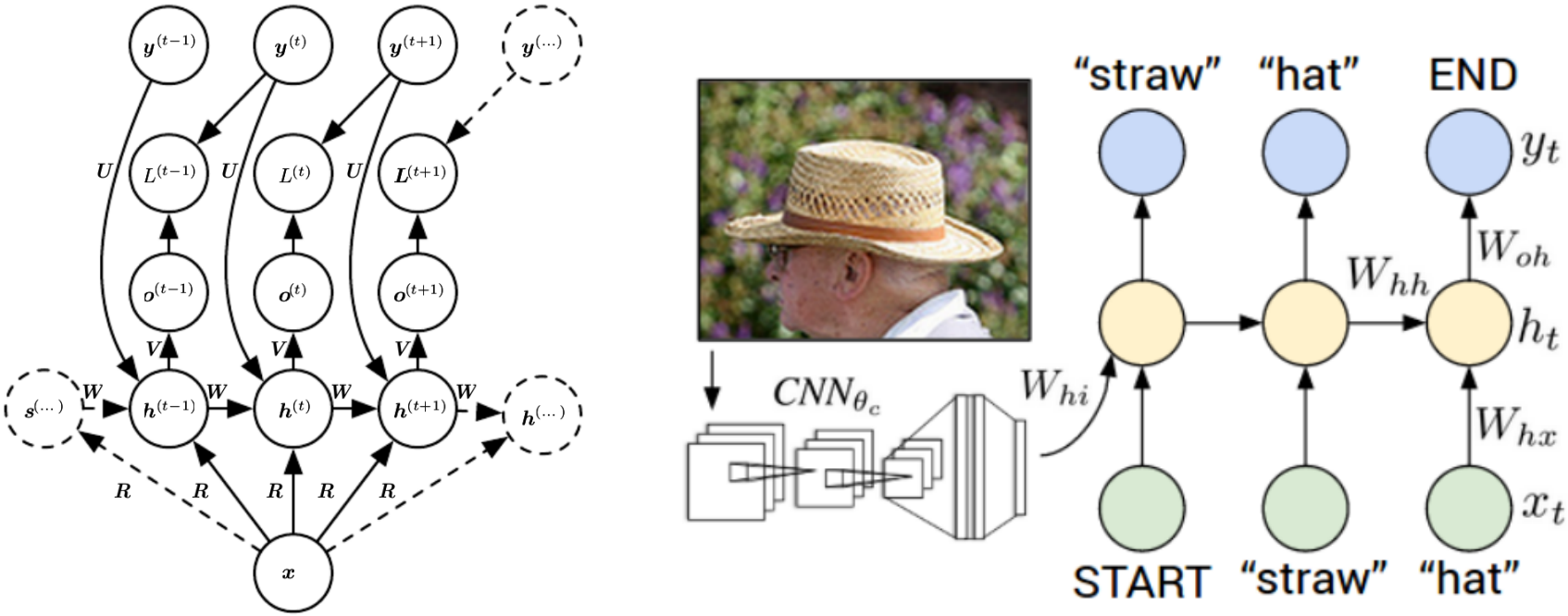


(a) Hidden-hidden Recurrence

(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



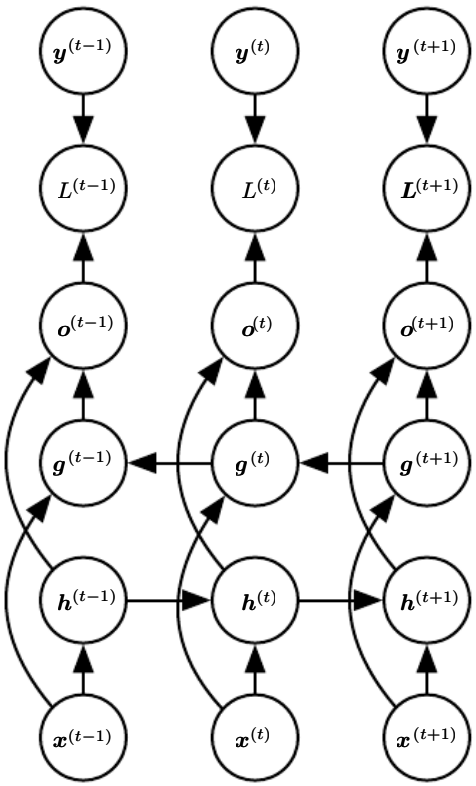
(a) (b)

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

Bidirectional RNNs

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

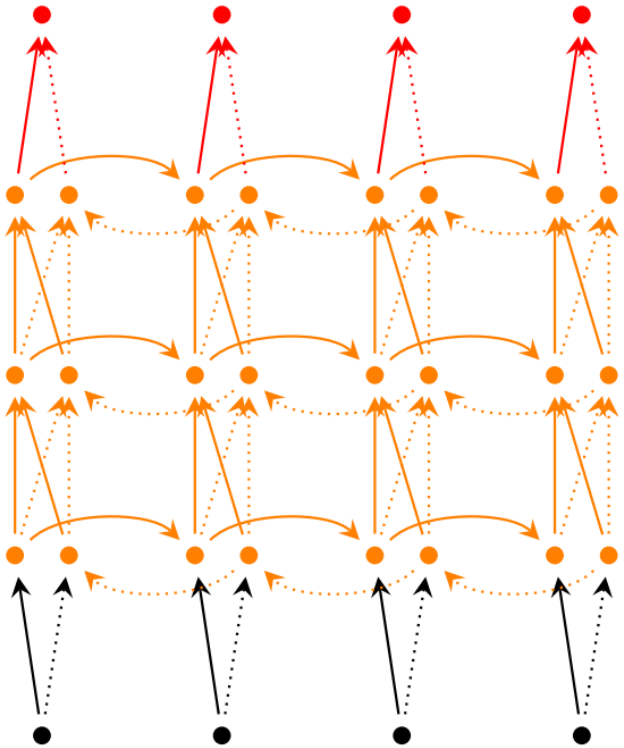


Bidirectional RNNs

- 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

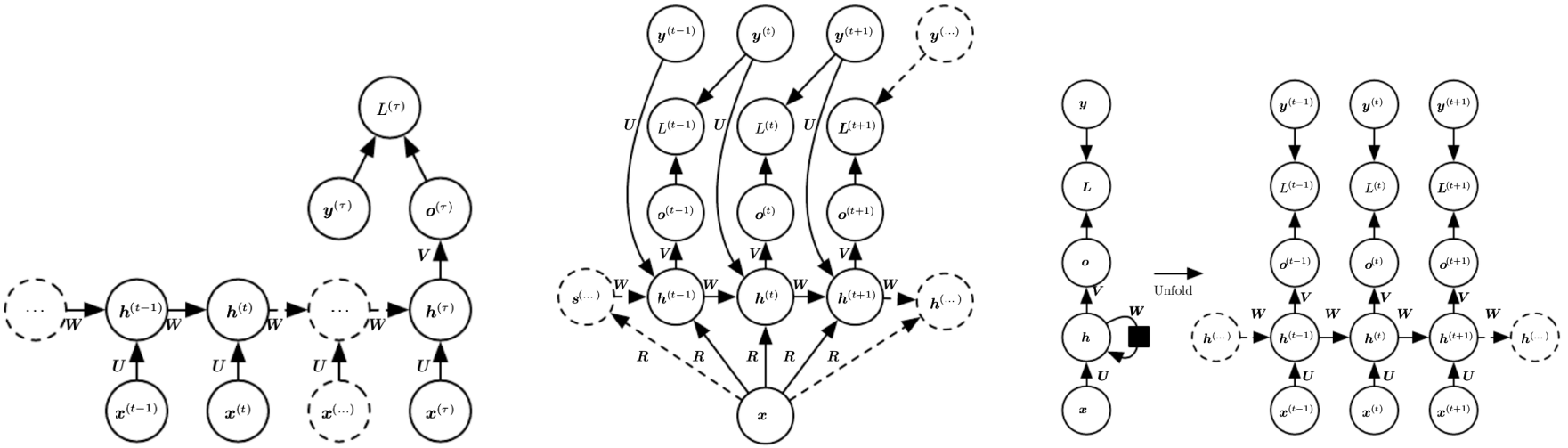


A deep bidirectional RNN with three RNN layers.

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

Overview

Encoder-Decoder Sequence-to-Sequence Architectures



(a) seq to fixed-size vec (b) fixed-size vec to 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

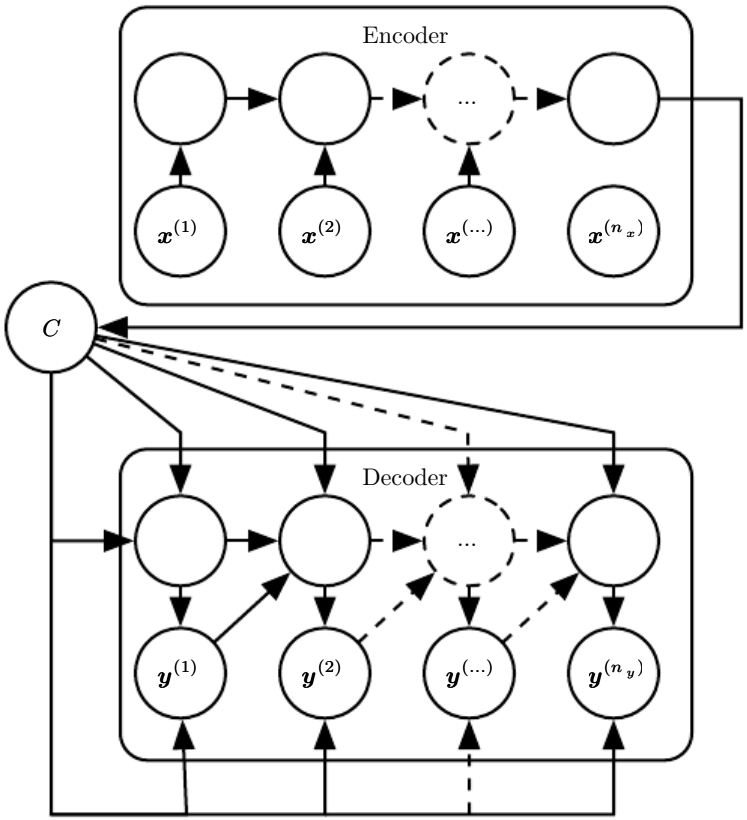


Figure 5: Encoder-decoder or sequence-to-sequence RNN architecture.

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

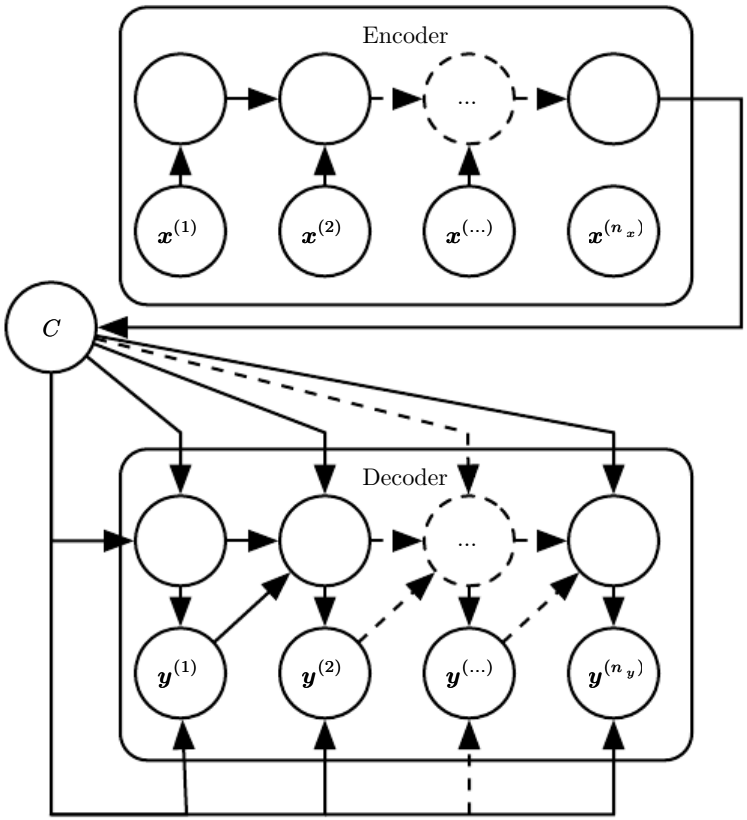


Figure 5: Encoder-decoder or sequence-to-sequence RNN architecture.

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 Architectures with Attention Mechanism

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.

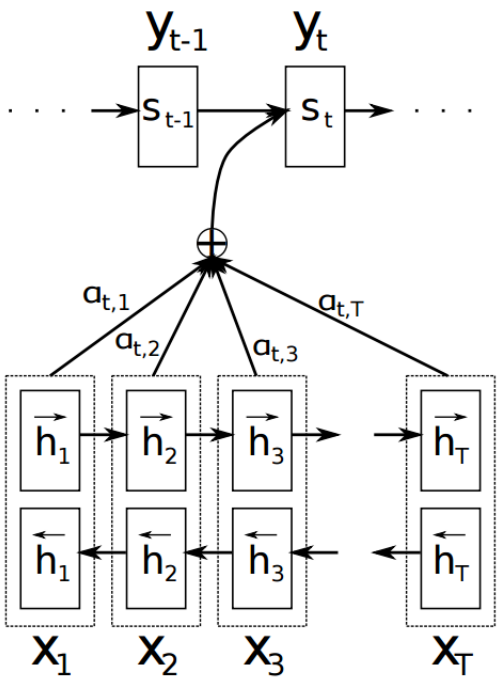
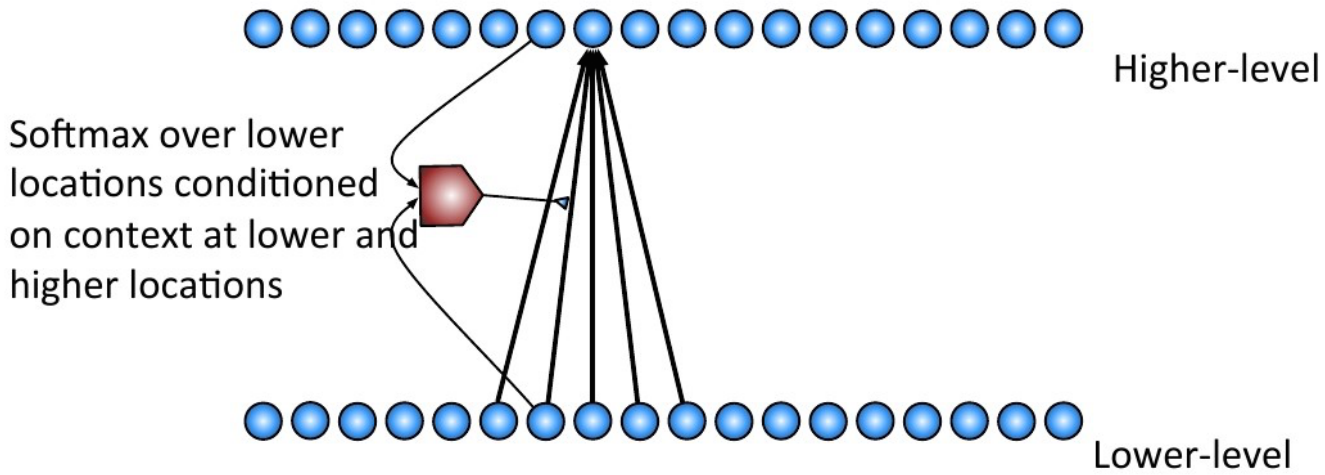


Figure 6: Formulatin: $a(\cdot)$ is an alignment model which scores how well the inputs around position j and the output at position i match, which is parameterized as a feedforward neural network

Encoder-Decoder Architectures with Attention Mechanism

- 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



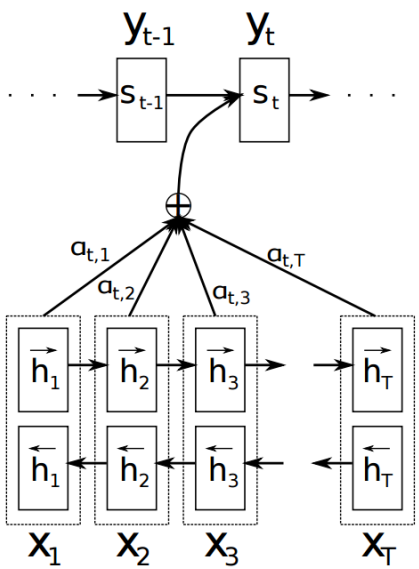
Encoder-Decoder Architectures with Attention Mechanism

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

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

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

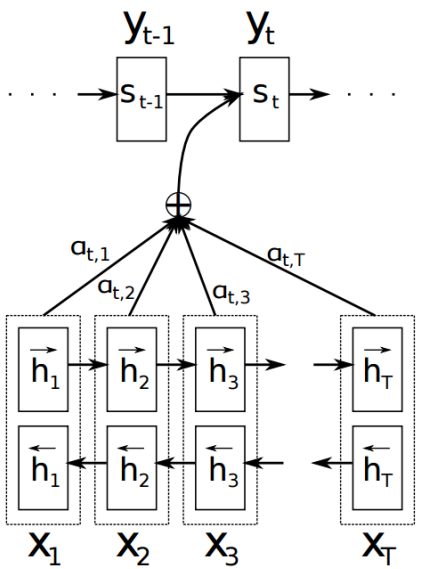
$$s_i = f(s_{i-1}, y_{i-1}, c_i) \tag{5}$$



Attention Mechanism

- 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

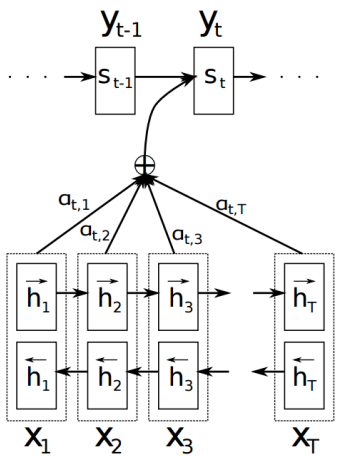
- The weight α is so called "attention matrix" (used to choose "where to look") and is computed by

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

where

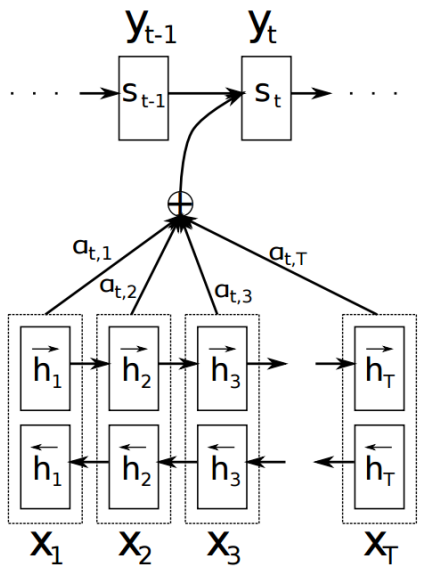
$$e_{tj} = h(s_{t-1}, h_j) \tag{8}$$

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



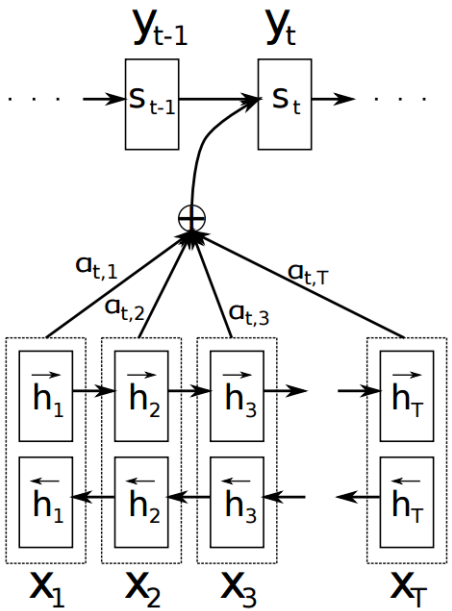
Attention Mechanism

- 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

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:

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

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

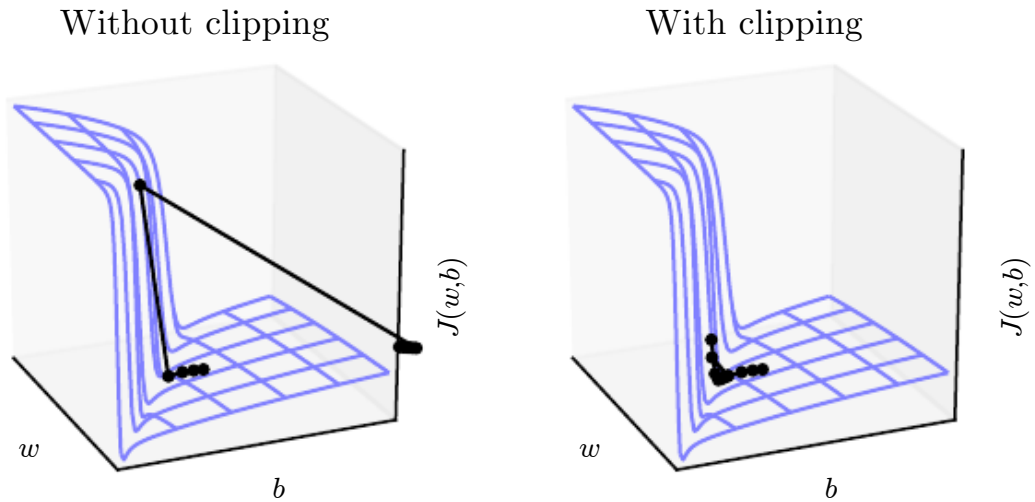
$$||\frac{\partial h^{(t)}}{\partial h^{(j-1)}}|| \leq ||W^{\top}|| \times ||\text{diag}\left(f'(h^{(j-1)})\right)||$$

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...

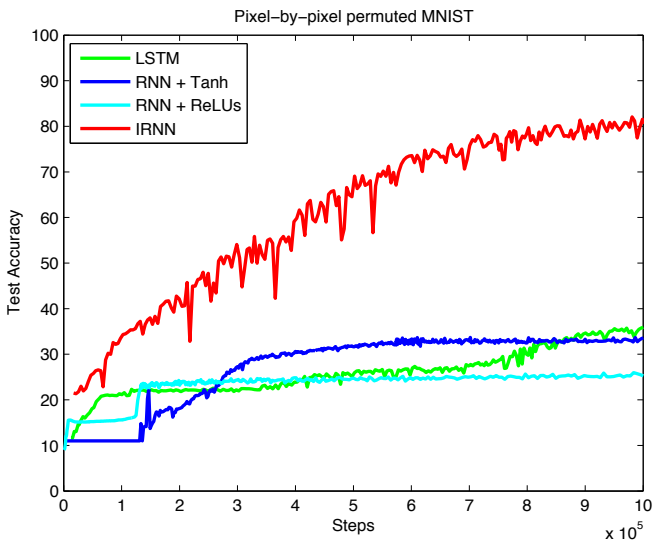
Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
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!

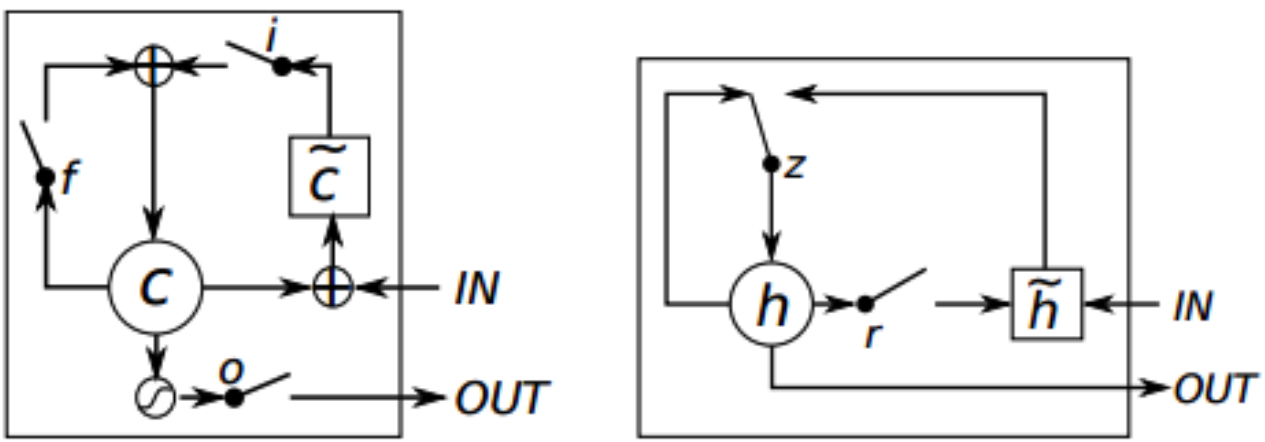
For vanishing gradients: Initialization+ReLU

- 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.



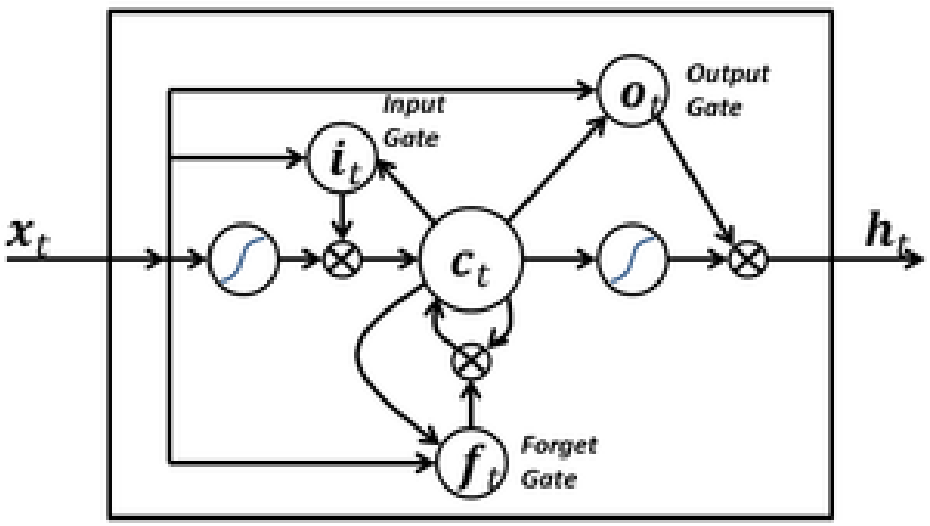
(a) Long Short-Term Memory (b) Gated Recurrent Unit

LSTM

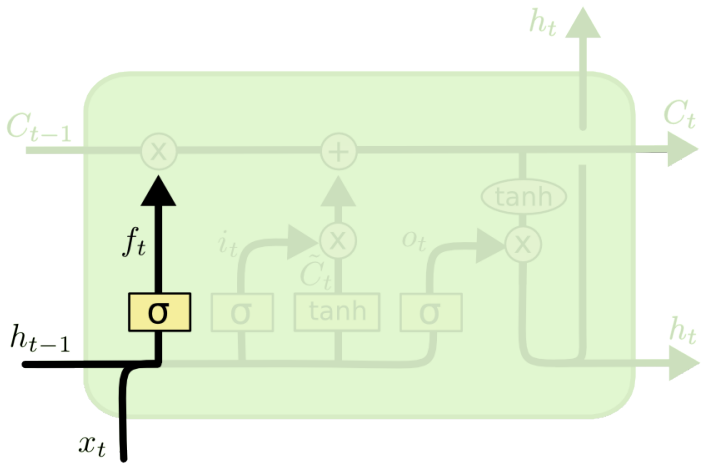
- Forget Gate: $\boldsymbol{f}^{(t)} = \text{sigmoid}(\boldsymbol{W}^f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}^f \boldsymbol{x}^{(t)} + \boldsymbol{b}^f)$
- Input Gate: $\boldsymbol{i}^{(t)} = \text{sigmoid}(\boldsymbol{W}^i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}^i \boldsymbol{x}^{(t)} + \boldsymbol{b}^i)$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

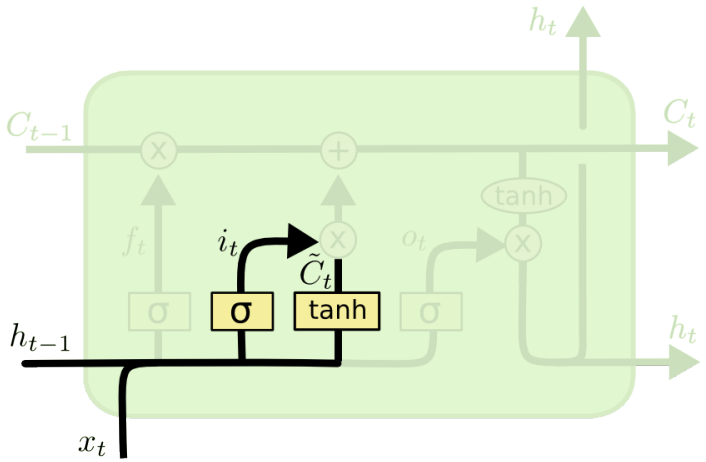


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

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



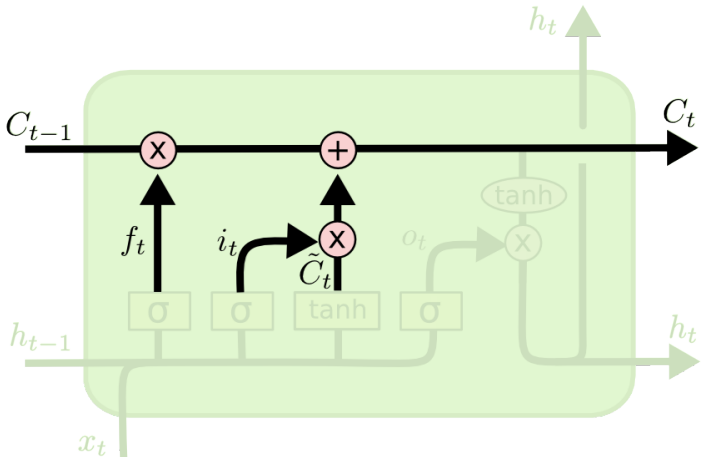
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

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

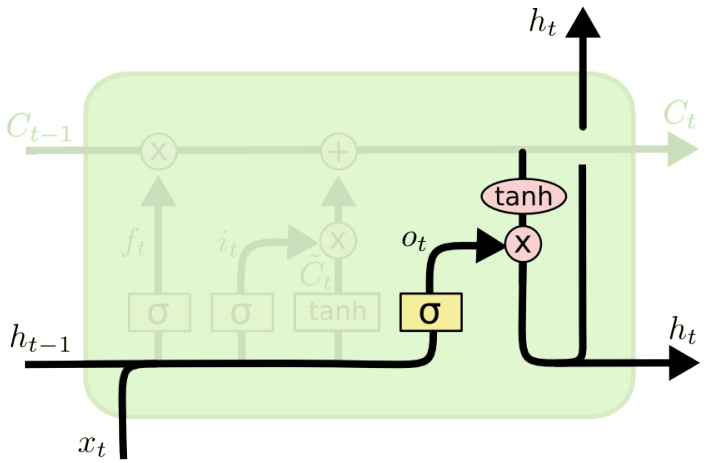


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

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 $\circ$

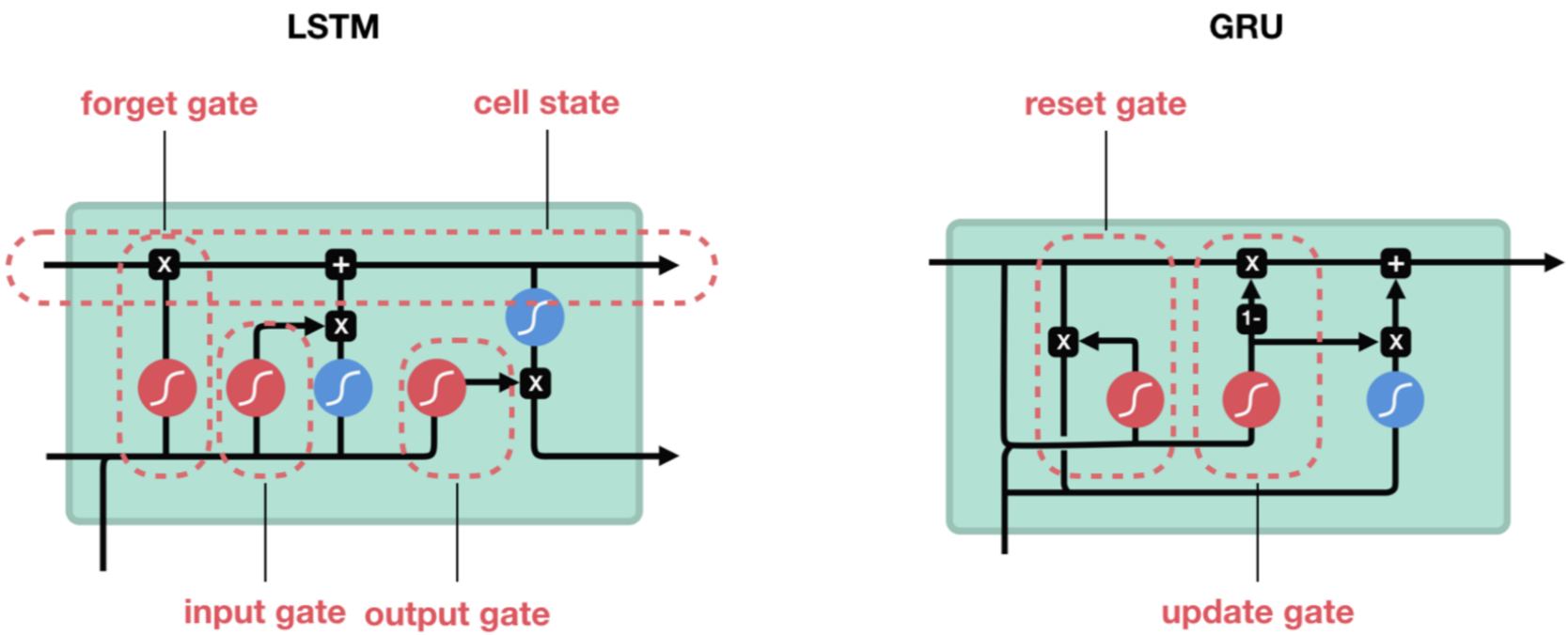
- Update Gate: $\boldsymbol{z}^{(t)} = \text{sigmoid}(\boldsymbol{W}^z \boldsymbol{h}^{(t-1)} + \boldsymbol{U}^z \boldsymbol{x}^{(t)} + \boldsymbol{b}^z)$
- Reset Gate: $\boldsymbol{r}^{(t)} = \text{sigmoid}(\boldsymbol{W}^r \boldsymbol{h}^{(t-1)} + \boldsymbol{U}^r \boldsymbol{x}^{(t)} + \boldsymbol{b}^r)$
- 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

Summary

- RNN is able to model sequential data with lots of flexibilities
- RNN can be also extended to model bidirectional dependencies and work with attention mechanism.
- LSTM is a special RNN and can be used to model long-term dependencies. GRU is a simplification of LSTM that runs faster and requires less training data.

Thanks.

HP: http://keg.cs.tsinghua.edu.cn/jietang/
Email: jietang@tsinghua.edu.cn