

# TTIC 31230, Fundamentals of Deep Learning

David McAllester, Winter 2020

Language Modeling

Recurrent Neural Networks (RNNs)

Machine Translation

Attention

Beam Search

Statistical Machine Translation

# Natural Language Understanding












## GLUE: General Language Understanding Evaluation

ArXiv 1804.07461






Corpus	Train	Test	Task	Metrics	Domain
Single-Sentence Tasks					
CoLA	8.5k	<b>1k</b>	acceptability	Matthews corr.	misc.
SST-2	67k	1.8k	sentiment	acc.	movie reviews
Similarity and Paraphrase Tasks					
MRPC	3.7k	1.7k	paraphrase	acc./F1	news
STS-B	7k	1.4k	sentence similarity	Pearson/Spearman corr.	misc.
QQP	364k	<b>391k</b>	paraphrase	acc./F1	social QA questions
Inference Tasks					
MNLI	393k	<b>20k</b>	NLI	matched acc./mismatched acc.	misc.
QNLI	105k	5.4k	QA/NLI	acc.	Wikipedia
RTE	2.5k	3k	NLI	acc.	news, Wikipedia
WNLI	634	<b>146</b>	coreference/NLI	acc.	fiction books

Table 1: Task descriptions and statistics. All tasks are single sentence or sentence pair classification, except STS-B, which is a regression task. MNLI has three classes; all other classification tasks have two. Test sets shown in bold use labels that have never been made public in any form.

# BERT and GLUE

Rank	Name	Model	URL	Score
1	ERNIE Team - Baidu	ERNIE		90.1
2	Microsoft D365 AI & MSR AI & GATECH	MT-DNN-SMART		89.9
3	T5 Team - Google	T5		89.7
+	4 王玮	ALICE v2 large ensemble (Alibaba DAMO NLP)		89.5
5	XLNet Team	XLNet (ensemble)		89.5
6	ALBERT-Team Google Language	ALBERT (Ensemble)		89.4
7	Microsoft D365 AI & UMD	FreeLB-RoBERTa (ensemble)		88.8
8	Facebook AI	RoBERTa		88.5
9	Junjie Yang	HIRE-RoBERTa		88.3
+	10 Microsoft D365 AI & MSR AI	MT-DNN-ensemble		87.6
11	GLUE Human Baselines	GLUE Human Baselines		87.1

# BERT and SuperGLUE

Rank	Name	Model	URL	Score
1	SuperGLUE Human Baselines	SuperGLUE Human Baselines		89.8
2	T5 Team - Google	T5		88.9
3	Facebook AI	RoBERTa		84.6
4	IBM Research AI	BERT-ml		73.5
5	SuperGLUE Baselines	BERT++		71.5
		BERT		69.0

# Language Modeling

The recent progress on NLP benchmarks is due to pretraining on language modeling.

Language modeling is based on unconditional cross-entropy minimization.

$$\Phi^* = \operatorname{argmin}_{\Phi} E_{y \sim P_{\text{op}}} - \ln Q_{\Phi}(y)$$

In language modeling  $y$  is a sentence (or fixed length block of text).

## Autoregressive Models

A structured object, such as a sentence or an image, has an exponentially small probability.

An autoregressive model computes conditional probability for each part given “earlier” parts.

This can be done for the words of a sentence or the pixels of an image.

Warning: BERT is not autoregressive. But we will study autoregressive models first.

# Language Modeling

Let  $W$  be some finite vocabulary of tokens (words).

Let  $\text{Pop}$  be a population distribution over  $W^*$  (sentences).

We want to train a model  $Q_\Phi(y)$  for sentences  $y$

$$\Phi^* = \underset{\Phi}{\operatorname{argmin}} E_{y \sim \text{Pop}} - \ln Q_\Phi(y)$$

## The End of Sentence Token <EOS>

We want to define a probability distribution over sentence of different length.

For this we require that each sentence is “terminated” with an end of sentence token <EOS>.

We require  $w_T = \text{<EOS>}$  and  $w[t] \neq \text{<EOS>}$  for  $t < T$ .

This allows

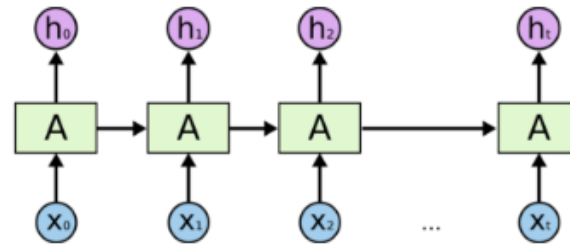
$$P(w_0, w_1, \dots, w_T) = \prod_{t=0}^T P(w_t \mid w_1, \dots, w_{t-1})$$

To handle sentences of different length.



want to know the probability of language

## Recurrent Neural Network (RNN) Language Modeling



→ hidden state

! autoregressive :

[Christopher Olah]

A typical neural language model has the form

predict the next words →

$$Q_{\Phi}(w_t \mid w_1, \dots, w_{t-1}) = \underset{w_t}{\text{softmax}} e(w_t)^{\top} h_{t-1}$$

softmax over vocabulary

**Word Embeddings:** Here each word  $w$  is associated with a vector  $e(w)$  called the embedding of word  $w$ .

## Standard Measures of Performance

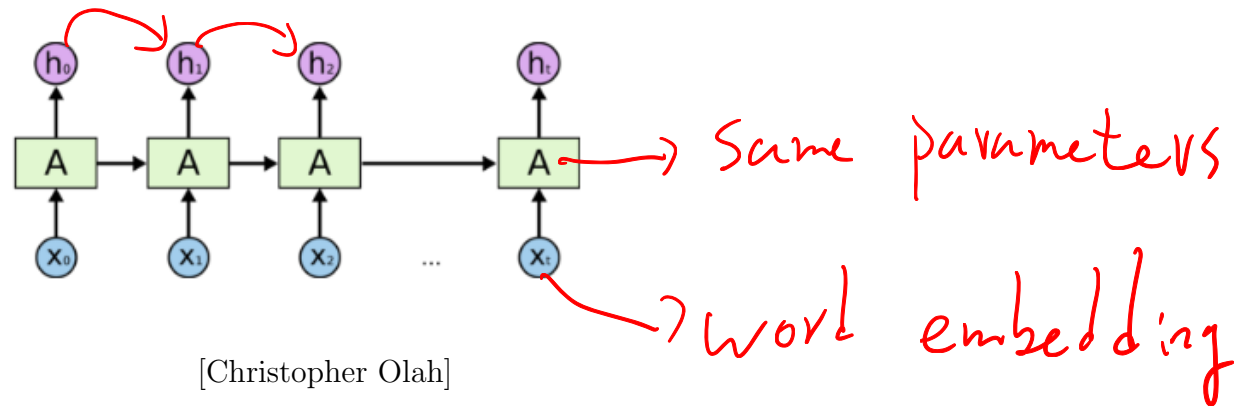
**Bits per Character:** For character language models performance is measured in bits per character. Typical numbers are slightly over one bit per character.

**Perplexity:** It would be natural to measure word language models in bits per word. However, it is traditional to measure them in perplexity which is defined to be  $2^b$  where  $b$  is bits per word. Perplexities of about 60 were typical until the past couple years. Today's large language models (ELMO, GPT-2) give perplexities of about 30.

According to Quora there are 4.79 letters per word. 1 bit per character (including space characters) gives a perplexity of  $2^{5.79}$  or 55.3.

# Recurrent Neural Networks (RNNs)

## Vanilla RNNs



[Christopher Olah]

A Vanilla RNN uses two-input linear threshold units.

$$h[b, t, j] = \sigma \left( \left( \sum_i W^{h,h}[j, i] h[b, t-1, i] \right) + \left( \sum_k W^{x,h}[j, k] x[b, t, k] \right) - B[j] \right)$$

Parameter  $\Phi = (W^{h,h}[J, J], W^{x,h}[J, K], B[J])$

Problem here:

## Exploding and Vanishing Gradients

If we avoid saturation of the activation functions then we get exponentially growing or shrinking eigenvectors of the weight matrix.

Note that if the forward values are bounded by sigmoids or tanh then they cannot explode.

However the gradients can still explode.

## Exploding Gradients: Gradient Clipping

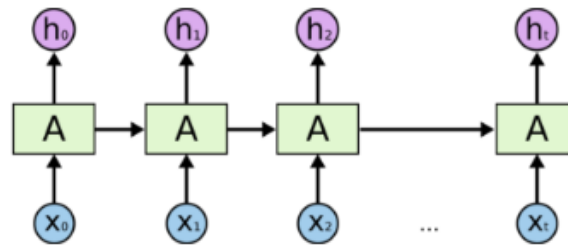
*brute force to make gradient small*

We can dampen the effect of exploding gradients by clipping them before applying SGD.

$$W.\text{grad}' = \begin{cases} W.\text{grad} & \text{if } ||W.\text{grad}|| \leq n_{\max} \\ n_{\max} W.\text{grad}/||W.\text{grad}|| & \text{otherwise} \end{cases}$$

See `torch.nn.utils.clip_grad_norm`

## Time as Depth



[Christopher Olah]

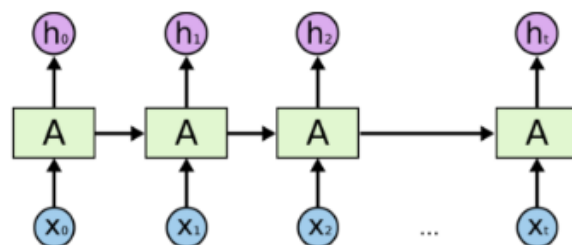
We would like the RNN to **remember and use** information from much earlier inputs.

All the issues with depth now occur through time.

However, for RNNs **at each time step we use the same model parameters.**

In CNNs **at each layer uses its own model parameters.**

# Skip Connections Through Time



[Christopher Olah]

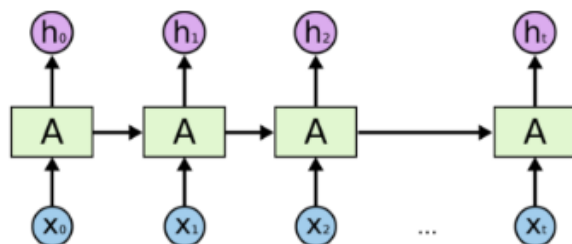
We would like to add **skip connections through time**.

However, We have to handle the fact that the same model parameters are used at every time step.



LSTM

## Gated RNNs



[Christopher Olah]

previous hidden state

previous layer

newly computed state

$$h[b, t, j] = G_t[b, t, j]h[b, t-1, j] + (1 - G[b, t, j])R[b, t, j]$$

Rather than add the diversion  $R[b, t, j]$  to  $h[b, t-1, j]$  we take a convex combination determined by a computed “gate”  $G[b, t, j] \in [0, 1]$ .

For  $G[b, t, j] = 1$  we pass feature  $j$  through unchanged.

For  $G[b, t, j] = 0$  we completely replace feature  $j$ .

## Update Gate RNN (UGRNN)

$$R[b, t, j] = \tanh \left( \left( \sum_i W^{h,R}[j, i] h[b, t-1, i] \right) + \left( \sum_k W^{x,R}[j, k] x[b, t, k] \right) - B^R[j] \right)$$

$$G[b, t, j] = \sigma \left( \left( \sum_i W^{h,G}[j, i] h[b, t-1, i] \right) + \left( \sum_k W^{x,G}[j, k] x[b, t, k] \right) - B^G[j] \right)$$

$$h[b, t, j] = G[b, t, j] h[b, t-1, j] + (1 - G[b, t, j]) R[b, t, j]$$

$$\Phi = (W^{h,R}, W^{x,R}, B^R, W^{h,G}, W^{x,G}, B^G)$$

## Hadamard product

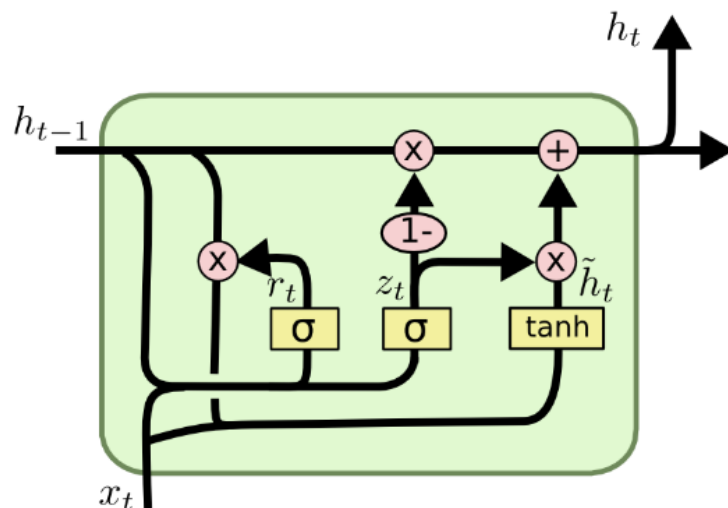
$$h[b, t, j] = G[b, t, j]h[b, t-1, j] + (1 - G[b, t, j])R[b, t, j]$$

is sometimes written as

$$h[b, t, J] = G[b, t, J] \odot h[b, t-1, J] + (1 - G[b, t, J]) \odot R[b, t, J]$$

$\odot$  is the Hadamard product (componentwise product) on vectors.

## Gated Recurrent Unity (GRU) by Cho et al. 2014

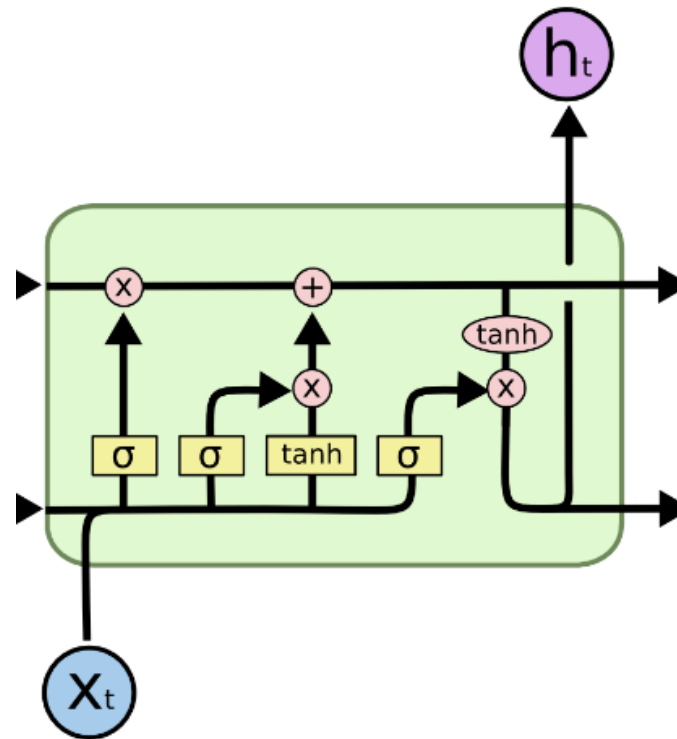


[Christopher Olah]

The right half is a UGRNN.

The GRU adds a gating on  $h_{t-1}$  before the tanh.

# Long Short Term Memory (LSTM)



[figure: Christopher Olah]

[LSTM: Hochreiter&Shmidhuber, 1997]

## UGRNN vs. GRUs vs. LSTMs

In class projects from previous years, GRUs consistently outperformed LSTMs.

A systematic study [Collins, Dickstein and Sussulo 2016] states:

Our results point to the GRU as being the most learnable of gated RNNs for shallow architectures, followed by the UGRNN.

## RNN for a generic CELL Procedure

As usual, we use capital letter indices to denote whole tensors or slices and lower case letters to denote particular index values.

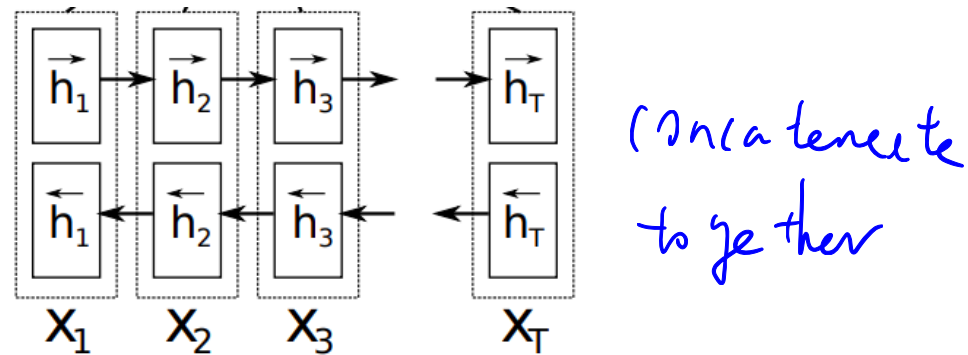
*solution of quiz!*

Procedure  $\text{RNN}_{\Phi}(x(T, I))$

$$\begin{aligned} h[0, J] &= \text{CELL}_{\Phi.\text{cell}}(\Phi.\text{init}[J], x[0, I]) \\ \text{for } t > 0 \quad h[t, J] &= \text{CELL}_{\Phi.\text{cell}}(h[t-1, J], x[t, I]) \end{aligned}$$

Return  $h[T, J]$

## bi-directional RNNs



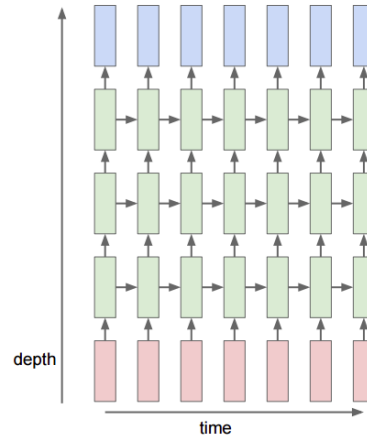
$$\vec{h}[T, J] = \text{RNN}_{\Phi.LR}(x[T, I])$$

$$\overleftarrow{h}[T, J] = \text{RNN}_{\Phi.RL}(x[T, I])$$

for  $t$   $h[t, 2J] = \vec{h}[t, J]; \overleftarrow{h}[t, J]$  where  $x; y$  is vector concatenation



# Multi-Layer RNNs



[Figure by Leonardo Araujo dos Santos]

$$h[0, T, J] = \text{RNN}_{\Phi[0]}(x[T, I])$$

$$\text{for } \ell > 0 \quad h[\ell, T, J] = \text{RNN}_{\Phi[\ell]}(h[\ell - 1, T, J])$$

Each layer can be bidirectional.

## Residual Multi-Layer RNNs

$$h[0, T, J] = \text{RNN}_{\Phi[0]}(x[T, I])$$

$$\text{for } \ell > 0 \quad h[\ell, T, J] = \textcolor{red}{h[\ell - 1, T, J]} + \text{RNN}_{\Phi[\ell]}(h[\ell - 1, T, J])$$

This is used in Google translation.

# Neural Machine Translation

# Machine Translation

$$w_1^{\text{in}}, \dots, w_{t_{\text{in}}}^{\text{in}} \Rightarrow w_1^{\text{out}}, \dots, w_{t_{\text{out}}}^{\text{out}}$$

$$w_{\text{in}}[T_{\text{in}}] \Rightarrow w_{\text{out}}[T_{\text{out}}]$$

Translation is a **sequence to sequence** (seq2seq) task.



**Sequence to Sequence Learning with Neural Networks**, Sutskever, Vinyals and Le, NIPS 2014, arXiv Sept 10, 2014.

We describe a simplification of the paper.

## Machine Translation

$$w_{\text{in}}[T_{\text{in}}] \Rightarrow w_{\text{out}}[T_{\text{out}}]$$

We define a model

$$P_{\Phi}(w_{\text{out}}[T_{\text{out}}] \mid w_{\text{in}}[T_{\text{in}}])$$

$$\begin{aligned}\Phi^* &= \operatorname{argmin}_{\Phi} E_{\text{Pop}} - \ln P_{\Phi}(w_{\text{out}}[T_{\text{out}}] \mid w_{\text{in}}[T_{\text{in}}]) \\ &= \operatorname{argmin}_{\Phi} E_{\text{Pop}} - \ln P_{\Phi}(y|x)\end{aligned}$$

## A Simple RNN Translation Model

The final state of a right-to-left RNN is used as the initial hidden state of a left-to-right RNN language model for the output.

The final right-to-left state  $\vec{\text{RNN}}_{\Phi.\text{in}}(w_{\text{in}}[T_{\text{in}}]) [0, J]$  is a “thought vector” representation of the input sentence.

Taking a right-to-left thought vector from the beginning of the input sentence facilitates getting a good start in left-to-right modeling of the output.

## Machine Translation Decoding

We can sample from  $P_{\Phi.\text{out}}(w[T] \mid H[J])$ .

But we might prefer

$$w_{\text{out}}[T_{\text{out}}] = \operatorname{argmax}_{w_{\text{out}}[T_{\text{out}}]} P_{\Phi} (w_{\text{out}}[T_{\text{out}}] \mid w_{\text{in}}[T_{\text{in}}])$$

This is typically approximated with greedy decoding:

$$w_{\text{out}}[t+1] = \operatorname{argmax}_w p_{\text{out}}[t+1, w]$$

These are not the same.

# Attention-Based Translation

**Neural Machine Translation by Jointly Learning to  
Align and Translate** Dzmitry Bahdanau, Kyunghyun Cho,  
Yoshua Bengio, ICLR 2015 (arXiv Sept. 1, 2014)

take the sequence of vectors from RNN and print

We describe a simplification of the paper.



## Representing Sentences by Vector Sequences

The input sentence is now represented by a sequence of vectors.

$$\begin{aligned} &P_{\Phi}(w_{\text{out}}[T_{\text{out}}] \mid w_{\text{in}}[T_{\text{in}}]) \\ &= P_{\Phi.\text{out}}(w_{\text{out}}[T_{\text{out}}] \mid \overleftarrow{\text{RNN}}_{\Phi.\text{RNN}}(w_{\text{in}}[T_{\text{in}}])) \end{aligned}$$

We still use the final state of an RNN run on the input as the initial state of a decoding RNN.

But the decoding RNN now has access to the entire sequence of hidden states for the input.

# Attention-Based Translation

## Memory-Based Language Modeling

Procedure  $P_{\Phi}(w[T] \mid \textcolor{red}{M}[T_M, J]) \;;; \textcolor{red}{w}[T]$  given

$$x[t, I] = (\Phi.\text{embed})[w[t], I]$$

$$h[T, J] = \text{RNN}_{\Phi.\text{RNN}}(x[T, I] \mid \textcolor{red}{M}[T_M, J]) \quad J \geq I$$

$$s[0, w] = (\Phi.\text{embed}[w, I]) \cdot (\Phi.\text{init}[I])$$

$$\text{for } t > 0 \quad s[t, w] = (\Phi.\text{embed}[w, I]) \cdot h[t-1, I] \quad h \text{ truncated to } I$$

$$p[t, w] = \underset{w}{\text{softmax}} \ s[t, w]$$


Return  $\prod_t p[t, w[t]]$

## Attention-Based (Memory-Based) Conditional RNN

Procedure  $\text{RNN}_\Phi(x[T_x, I] \mid M[T_M, J])$

$$h[0, J] = \text{CELL}_\Phi(M[0, J]; x[0, I]; 0[J])$$

for  $t > 0$

$$h[t, J] = \text{CELL}_\Phi(h[t-1, J]; x[t, I]; \text{Lookup}(h[t-1, J], M[T, J]))$$


Return  $h[T, I]$

Memory, sequence of input sentence

## Attention as a Key-Value Memory Mechanism

Procedure Lookup(key[J], M[T, J])

Score for  
each place  
of input

inner product

$$s[t] = \text{key}[J]^T M[t, J]$$

hidden state h

memory in your input

index for memory

$$\alpha[t] = \underset{t}{\text{softmax}} s[t]; \text{ the attention}$$

pdf of position

$$V[J] = \sum_t \alpha[t] M[t, J] \Rightarrow \text{Weighted sum of pdf}$$

M: memory

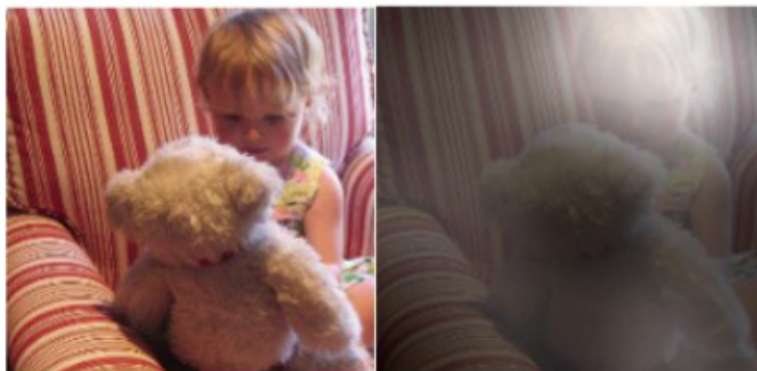
T: position

Return V[J]

## Attention in Image Captioning



A woman is throwing a frisbee in a park.



A little girl sitting on a bed with a teddy bear.

Xu et al. ICML 2015

## Greedy Decoding vs. Beam Search

We would like

$$W_{\text{out}}[T_{\text{out}}]^* = \operatorname{argmax}_{W_{\text{out}}[T_{\text{out}}]} P_{\Phi}(W_{\text{out}}[T_{\text{out}}] \mid W_{\text{in}}[T_{\text{in}}])$$

But a greedy algorithm may do well

$$w_t = \operatorname{argmax}_w P_{\Phi}(w \mid W_{\text{in}}[T_{\text{in}}], w_1, \dots, w_{t-1})$$

But these are not the same.

## Example

“Those apples are good” vs. “Apples are good”

$$P_{\Phi}(\text{Apples are Good } \langle \text{eos} \rangle) > P_{\Phi}(\text{Those apples are good } \langle \text{eos} \rangle)$$

$$P_{\Phi}(\text{Those}|\varepsilon) > P_{\Phi}(\text{Apples}|\varepsilon)$$

## Beam Search

At each time step we maintain a list the  $K$  best words and their associated hidden vectors.

This can be used to produce a list of  $k$  “best” decodings which can then be compared to select the most likely one.



# Phrase Based Statistical Machine Translation (SMT)

Phrase based SMT dominated machine translation before deep Seq2Seq models.

SMT is still used for low resource languages, such as regional African languages, and in unsupervised machine translation.

## Phrase Based Statistical Machine Translation (SMT)

Step I: Learn a phrase table — a set of triples  $(p, q, s)$  where

- $p$  is a (short) sequence of source words.
- $q$  is a (short) sequence of target words.
- $s$  is a score.

(“au”, “to the”, .5)                      (“au banque”, “for the bank”, .01)

For a phrase triple  $P$  we will write  $P$ .source for the source phrase,  $P$ .target for the target phrase, and  $P$ .score for the score.

## Derivations

Consider an input sentence  $x$  of length  $T$ .

We will write  $x[s : t]$  for the substring  $x[s], \dots, x[t - 1]$ .

A derivation  $d$  from  $x$  is a sequence  $(P_1, s_1, t_1, ), \dots, (P_K, s_K, t_K)$  where  $P_k.\text{source} = x[s_k : t_k]$ .

The substrings  $x[s_k : t_k]$  should be disjoint and “cover”  $x$ .

For  $d = [(P_1, s_1, t_1, ), \dots, (P_L, s_K, t_K)]$  we define

$$y(d) \equiv P_1.\text{target} \cdots P_K.\text{target}$$

We let  $D(x)$  be the set of derivations from  $x$ .

## Scoring

For  $d \in D(x)$  we define a score  $s(d)$

$$s(d) = \alpha \ln P_{\text{LM}}(y(d)) + \beta \sum_k P_k.\text{score} + \gamma \text{distortion}(d)$$

where  $P_{\text{LM}}(y)$  is the probability assigned to string  $y$  under a language model for the target language

and  $\text{distortion}(d)$  is a measure of consistency of word ordering between source and target strings as defined by the indices  $(s_1, t_1), \dots, (s_K, t_K)$ .

# Translation

$$y(x) = y(d^*(x))$$

$$d^*(x) = \operatorname{argmax}_{d \in D(x)} s(d)$$

**END**