Neural Models for Sequence Prediction --- Recurrent Neural Networks

Sunita Sarawagi
IIT Bombay
sunita@iitb.ac.in

Sequence Modeling taks

- Classify sequences $x_1, \ldots, x_n \rightarrow y$. E.g. sentiment classification, normal versus abnormal traffic
- Next word in a sequence $x_1, \ldots, x_n \rightarrow x_{n+1}$. E.g. language modeling
- Label per token in sequence $x_1, \ldots, x_n \rightarrow y_1, \ldots, y_n$. E.g. POS, speech recognition.
- Sequence prediction tasks $\mathbf{x} \rightarrow y_1, \ldots, y_m$ E.g. Translation, conversation assistant.

More examples

- Forecasting (Time Series)

$y_1, y_2 \ldots y_t$ outputs eg: demand for an item

$x_1, x_2 \ldots x_t$ inputs

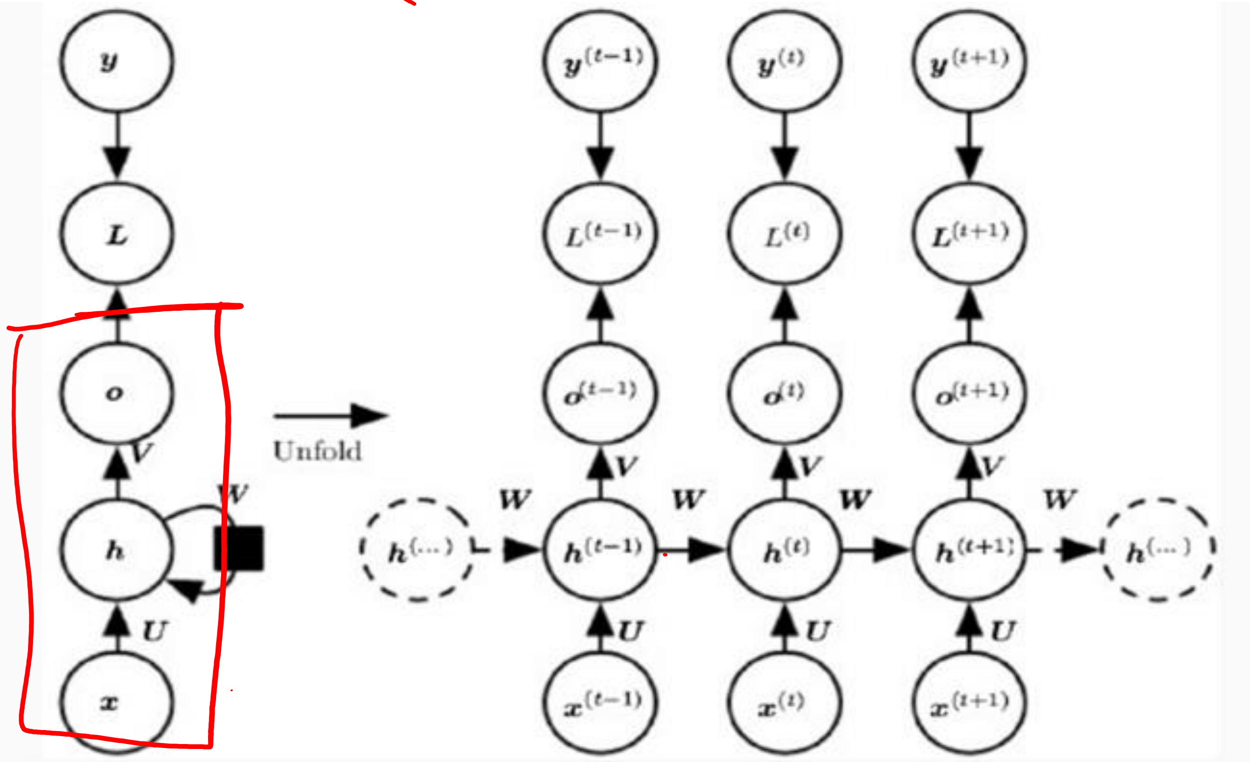
x_t : [day of the week, holiday?, temperature, month]

RNN: Recurrent Neural Network

- A model to process variable length 1-D input
- In CNN, each hidden output is a function of corresponding input and some immediate neighbors.
- In RNN, each output is a function of a 'state' summarizing all previous inputs and current input. State summary computed recursively.
- RNN allows deeper, longer range interaction among parameters than CNNs for the same cost.

RNNs: Basic type

- Notation:
  - ht to denote state instead of zt
  - Input to RNN is xt, instead of yt



$$o^t = c + Vh^t, \quad h^t = \sigma(b + Wh^{t-1} + Ux^t)$$

RNN: forward computation example.

x_1 , y_1

x_2 , y_2

$\vdots$

$x_t \quad y_t \qquad h^0 = 0$

$x_i \in \mathbb{R}$

$y_i \in \mathbb{R}$

$U = \mathbb{R}^{4\times 1}$ $\quad W$

$h = \mathbb{R}^4$

$W = \mathbb{R}^{4\times 4}$

$V = \mathbb{R}^{1\times 4}$

$b = \mathbb{R}^{4\times 1}$

$c = \mathbb{R}^{1\times 1}$

$\underline{t=1}$

$\sigma(W\underline{h^0} + U\underline{x_1} + b) = h^{(1)}$

$o^{(1)} = Vh^{(1)} + c$

$\text{Loss} = L_1 = (y_1 - o^1)^2$

$\underline{t=2}$

$\sigma(Wh^1 + Ux_2 + b) = h^{(2)}$

$o^{(2)} = Vh^{(2)} + c$

$L_2 = (y_2 - o^{(2)})^2$

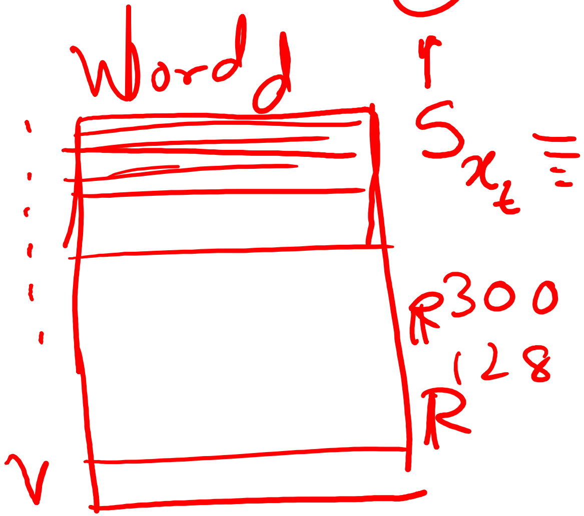
RNN for text (Predict next word) – word embeddings

x_t = word $\in [1 \cdots V]$

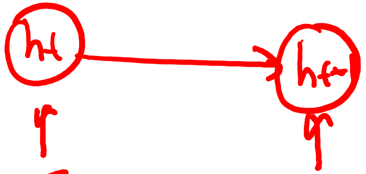
vocabulary size.

$V = 30k$

$x_t = \mathbb{R}^V$



Embeddings



$S_{x_t} = x_t S$

Learned parameters → $S = V \times d$

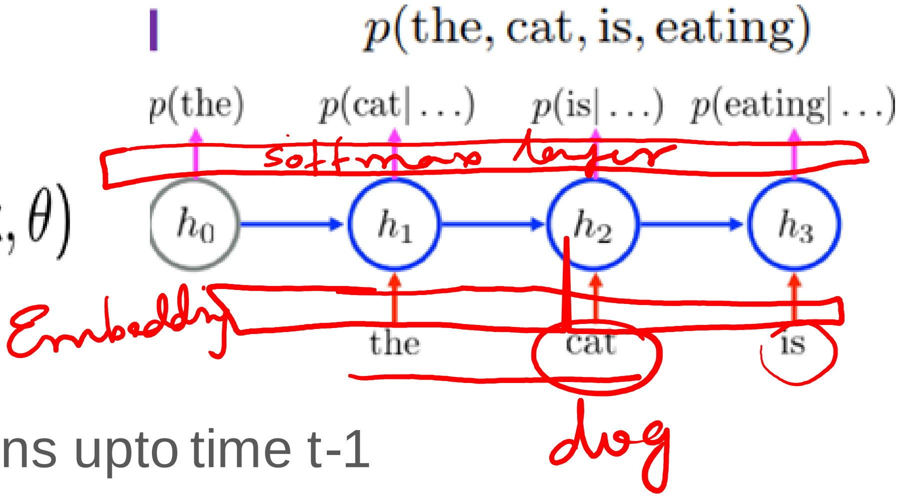
Typically $d \approx 300$

Training a sequence model

- Maximum Likelihood

$$P(\mathbf{y}|\mathbf{x},\theta) = \prod_{t=1}^{n} P(y_t|y_1,\ldots,y_{t-1},\mathbf{x},\theta)$$

- Mechanism of training
  - Input to RNN is the true tokens upto time t-1
  - Output is the probability distribution over tokens
  - Maximize the probability of the correct token.
- Advantages
  - Easy. Generative --- token at a time. Sound-- full dependency!



Training data :

$D = \{(x^1, y^1), \ldots (x^N, y^N)\}$

LM: $y^i = x^i$ shifted by one to the left

Example x^i: The cat is sleeping

$x^i_1 \quad x^i_2 \quad x^i_3 \quad x^i_4$

y^i

cat is sleeping EOS

$y^i_1 \quad y^i_2 \quad y^i_3 \quad y^i_4$

Training RNN parameters

Backpropagation through time

- Unroll graph along time
- Compute gradient through back-propagation exactly as in feedforward networks
- Sum up the gradient from each layer since parameters are shared.

Backpropagation through time

$$L = L_{t-1} + L_t$$

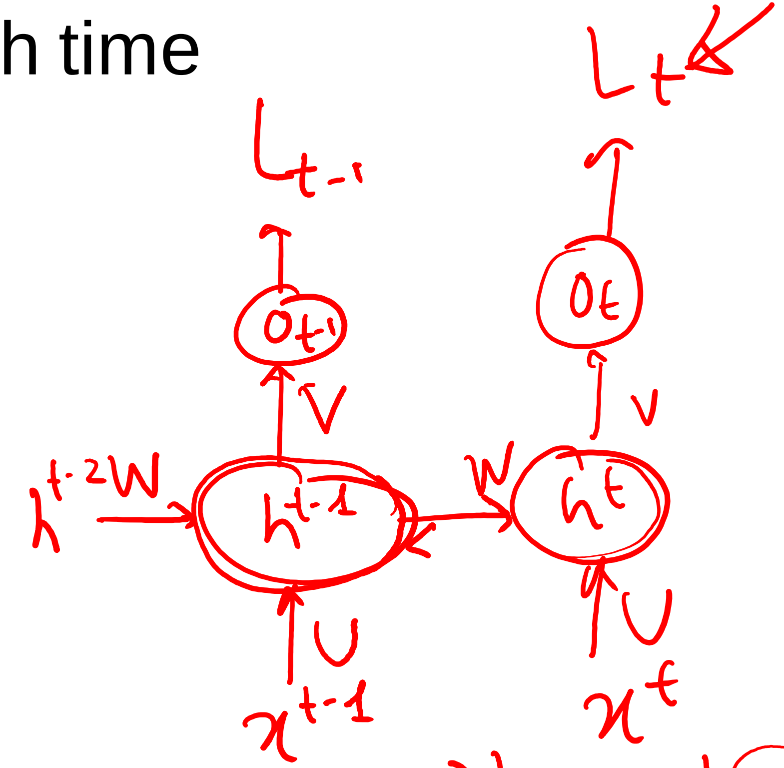
$$\frac{\partial L}{\partial V} = \frac{\partial L_{t-1}}{\partial V} + \frac{\partial L_t}{\partial V}$$

$$= \frac{\partial L_{t-1}}{\partial O_{t-1}} h^{t-1} + \frac{\partial L_t}{\partial O_t} \cdot h^t$$

$$\frac{\partial L}{\partial W} = \frac{\partial L_{t-1}}{\partial W} + \frac{\partial L_t}{\partial W}$$

$$= \frac{\partial L_{t-1}}{\partial O_{t-1}} \frac{\partial O_{t-1}}{\partial h^{t-1}} \frac{\partial h^{t-1}}{\partial W} + \frac{\partial L_t}{\partial O_t} \cdot V \cdot \sigma' h^{t-1} + \frac{\partial L_t}{\partial O_t} V \cdot \sigma' W \frac{\partial h^{t-1}}{\partial W}$$

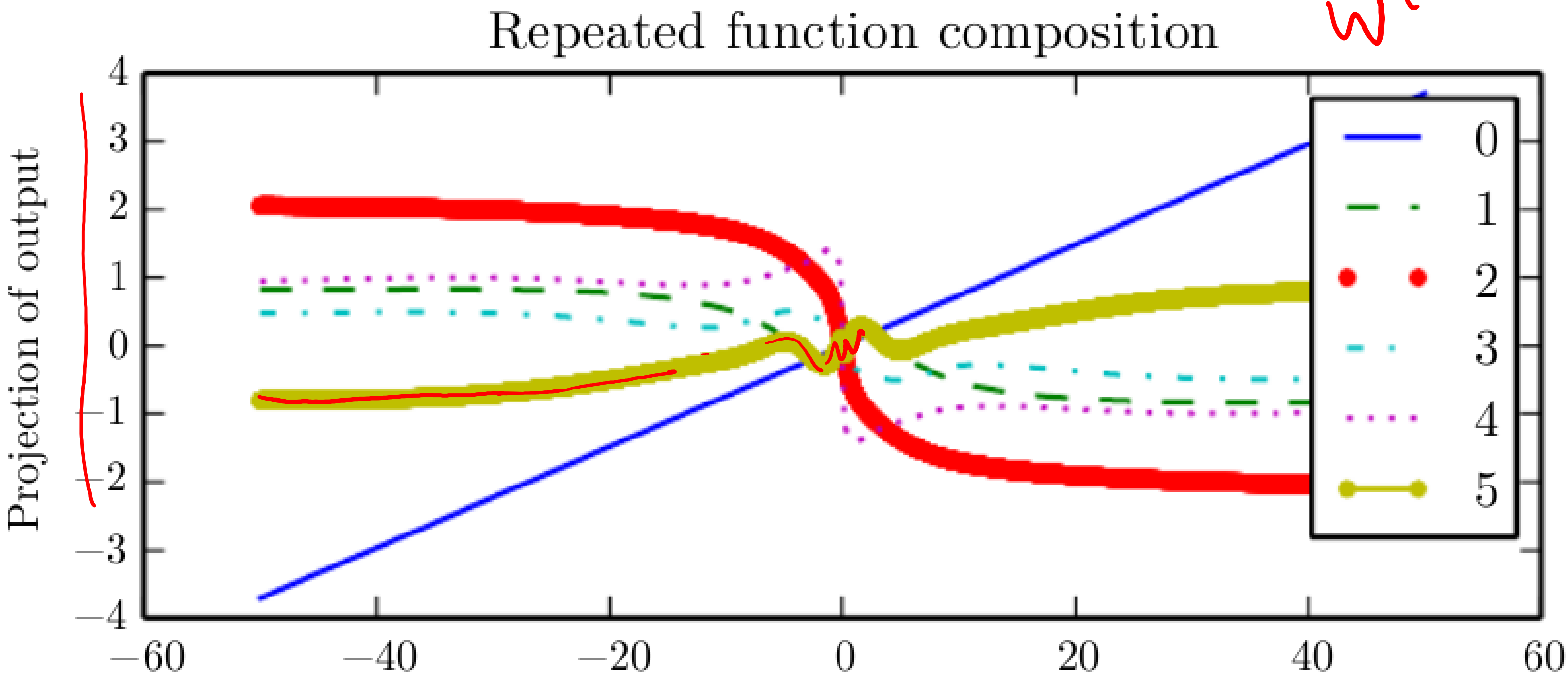
$$\ldots \boxed{\cdot}\, V \sigma' \cdot h^{t-2} + \ldots + \equiv O(W^t)$$



Exploding and vanishing gradient problem

Product of non-linear interactions: gradient either small or large

w^t



Fixes for vanishing/exploding gradient problem

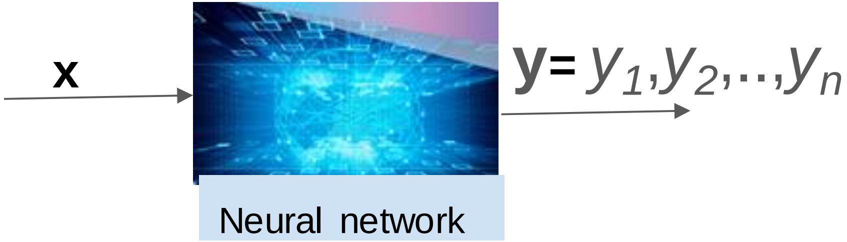
- No parameters for updating state: state is a "reservoir" of all past inputs, output is a learned function of state. E.g. Echo state networks, Liquid networks
- Multiple time scales: add direct connection from far inputs instead of depending on state to capture all far-off inputs.
- Shortcomings of above:
  - How far back we look at each t is same for all t and cannot be changed for different times or different inputs
  - Only accumulate information, cannot forget information.
- Solution: Gated RNNs e.g. LSTMs

Gated RNNs

- Gates control which part of the long past is used for current prediction
- Gates also allow forgetting of part of the state
- LSTM: Long Short Term Memory, one of the most successful gated RNNs.
- An excellent introductions here:
  - http://colah.github.io/posts/2015-08-Understanding-LSTMs/
  - http://blog.echen.me/2017/05/30/exploring-lstms/

The sequence prediction task

- Given a complex input **x**
  - Example: sentence(s), image, audio wave
- Predict a sequence **y** of discrete tokens $y_1, y_2, .., y_n$
  - Typically a sequence of words.
  - A token can be any term from a huge discrete vocabulary
  - Tokens are inter-dependent
    - Not n independent scalar classification task.



Motivation

- Applicable in diverse domains spanning language, image, and speech processing.
- Before deep learning each community solved the task in their own silos → lot of domain expertise
- The promise of deep learning: as long as you have lots of labeled data, domain-specific representations learnable
- This has brought together these communities like never before!

Translation

Context: **x**

Predicted sequence: **y**

Where can I find healthy and traditional Indian food? →

स्वस्थ और पारंपरिक भारतीय भोजन कहां मिल सकता है?

- Pre-DL translation systems were driven by transfer grammar rules painstakingly developed by linguists and elaborate phrase translation
- Whereas, modern neural translation systems are scored almost 60% better than these domain-specific systems.

Image captioning

Context: **x**

Predicted sequence: **y**

A person riding a motorcycle on a dirt road

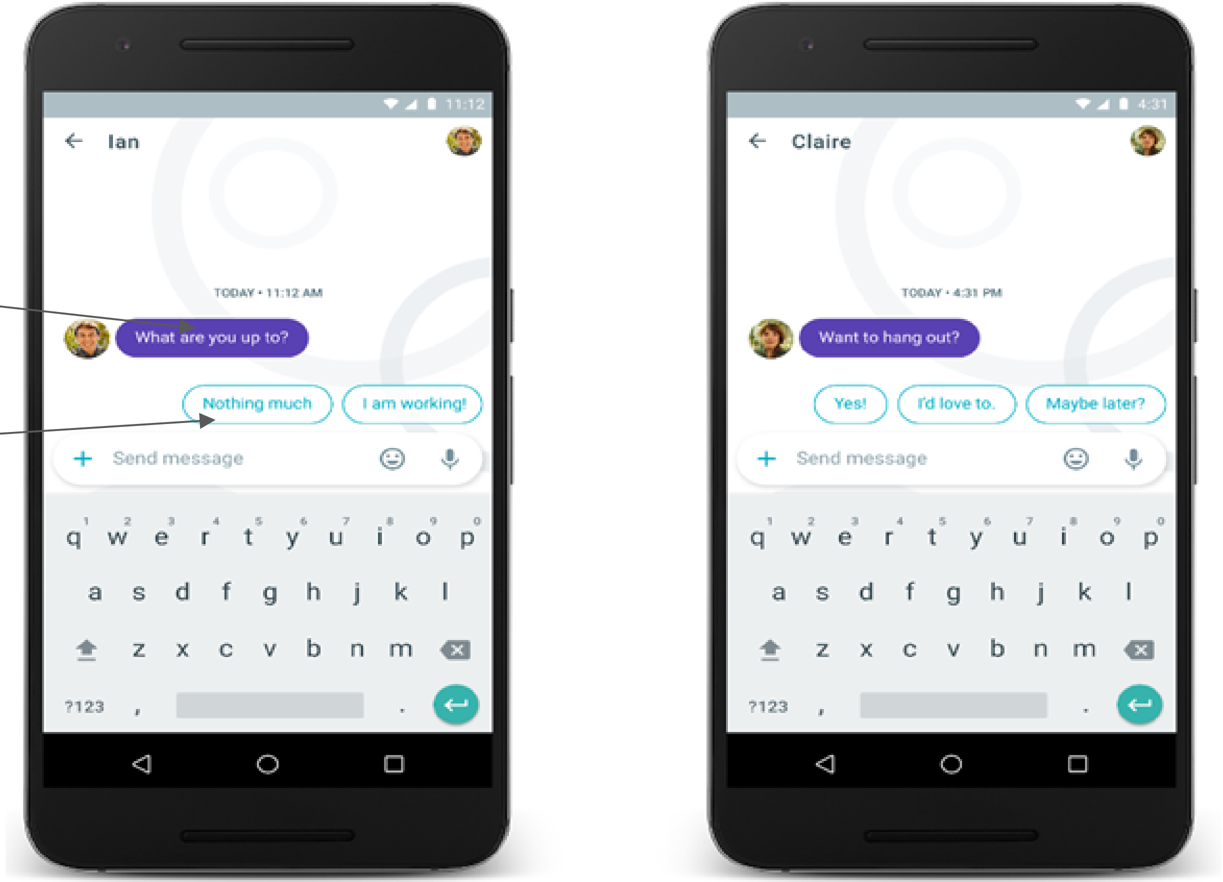
- Early systems: either template-driven or transferred captions from related images
- Modern DL systems have significantly pushed the frontier on this task.

Image from http://idealog.co.nz/tech/2014/11/googles-latest-auto-captioning-experiment-and-its-deep-fascination-artificial-intelligence

Conversation assistance

Context: **x**

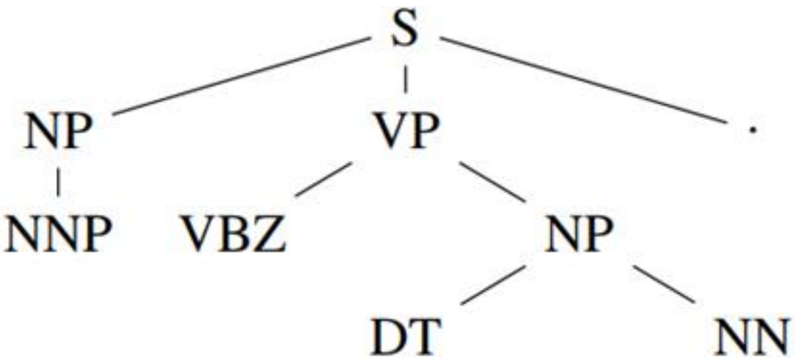
Predicted sequences: **y**



From https://research.googleblog.com/search?updated-max=2016-06-20T05:00:00-07:00&max-results=7&start=35&by-date=false

Syntactic parsing

John has a dog . →



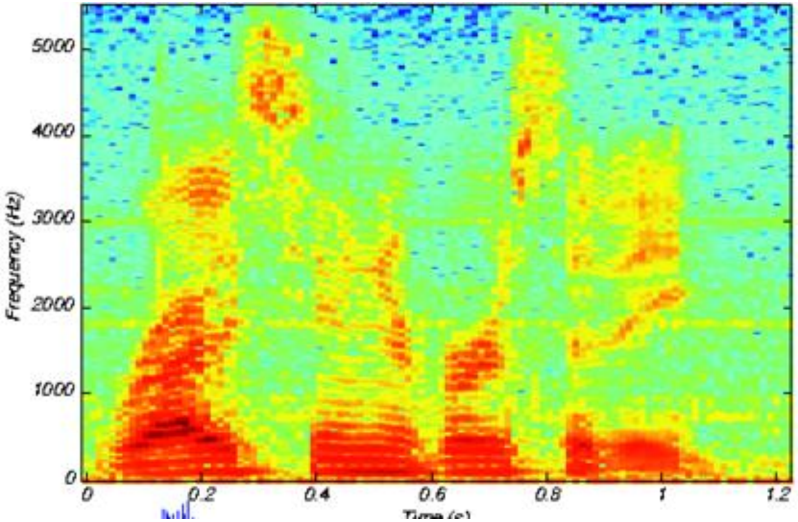
Context: **x**

Predicted sequence: **y**

John has a dog . → (S (NP NNP $)_{NP}$ (VP VBZ (NP DT NN $)_{NP}$ $)_{VP}$. $)_{S}$

Speech recognition

Context: **x (Speech spectrogram)**



Output: **Y (Phoneme Sequence)**

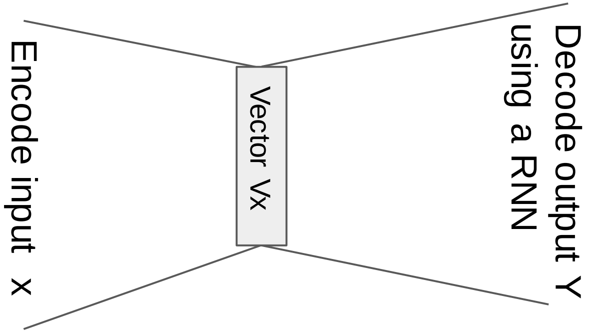
Ri ce Uni ver si ty

Challenges

- Capture long range dependencies
  - No conditional independencies assumed
  - Example during correct anaphora resolution in output sentence:
    - How is your son? I heard he was unwell.
- Prediction space highly open-ended
  - No obvious alignment with input unlike in tasks like POS, NER
  - Sequence length not known. Long correct response has to compete with short ones
    - How are you?
      - "Great" Vs "Great, how about you?"

The Encoder Decoder model for sequence prediction

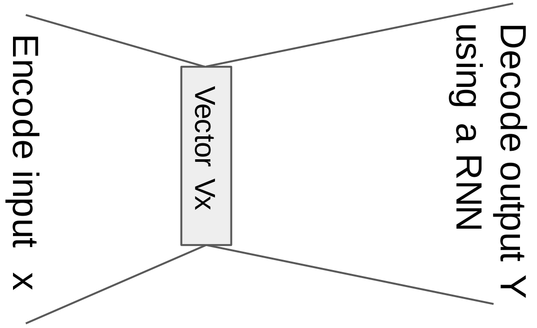
- Encode x into a fixed-D real vector X
- Decode y token by token using a RNN
  - Initialize a RNN state with X
  - Repeat until RNN generates a EOS token
    - Feed as input previously generated token
    - Get a distribution over output tokens, and choose best.



The Encoder Decoder model for sequence prediction

- Encode x into a fixed-D real vector X
- Since Y has many parts, need a graphical model to express the joint distribution over constituent tokens y1,...,yn. Specifically, we choose a special Bayesian network, called a RNN

$$P(\mathbf{y}|\mathbf{x},\theta) = \prod_{t=1}^{n} P(y_t|y_1,\ldots,y_{t-1},\mathbf{x},\theta)$$



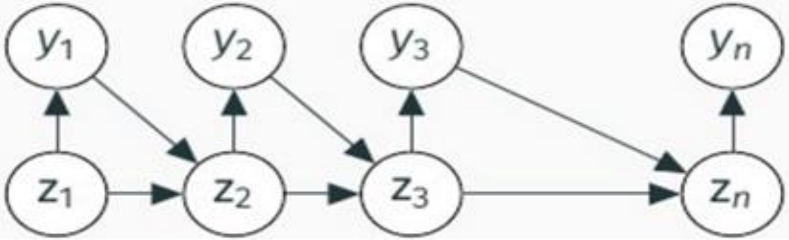
Encoder decoder model

- Let $\mathbf{v}_x$=fixed dimensional vector summary of input $\mathbf{x}$
- Difficult to define CPD $\Pr(y_t|y_1, \ldots, y_{t-1}, \mathbf{v}_x)$ with variable length of parents. Need to share parameters.
- Redesign the BN by summarizing parents as state.

$$\Pr(y_t|y_1, \ldots, y_{t-1}, \mathbf{v}_x, \theta) = P(y_t|\mathbf{z}_t, \theta), \quad (1)$$

where $\mathbf{z}_t$ is a state vector implemented using a recurrent neural network as

$$\mathbf{z}_t = \begin{cases} \mathbf{v}_x & \text{if } t = 0, \\ \text{RNN}(\mathbf{z}_{t-1}, y_{t-1}, \theta_R) & \text{otherwise.} \end{cases} \quad (2)$$



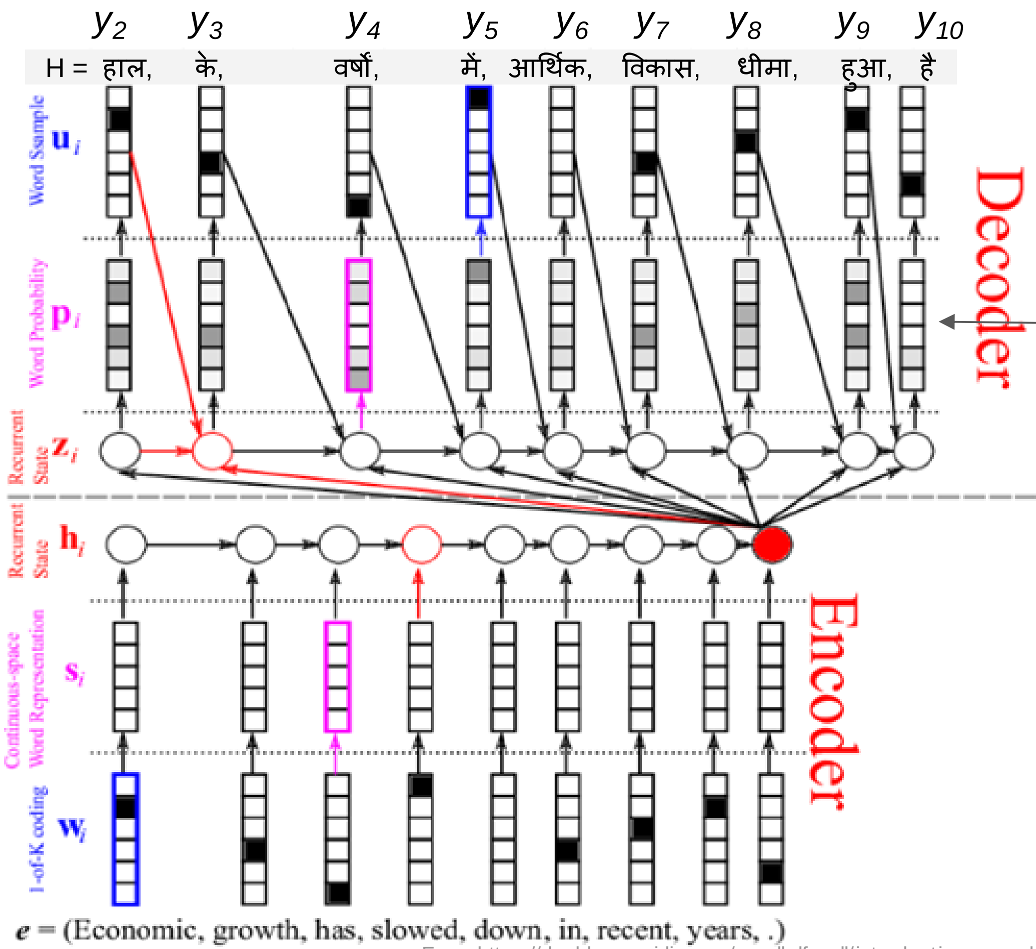
Encoder-decoder model

- Models full dependency among tokens in predicted sequence
  - Chain rule $P(\mathbf{y}|\mathbf{x},\theta) = \prod_{t=1}^{n} P(y_t|y_1,\ldots,y_{t-1},\mathbf{x},\theta)$
  - No conditional independencies assumed unlike in CRFs
- Training:
  - Maximize likelihood. Statistically sound!
- Inference
  - Find **y** with maximum probability → intractable given above
  - Beam search: branch & bound expansion of frontier of ‘beam width’
    - Probability of predicted sequence increases with increasing beam width.

Inference

- Finding the sequence of tokens y1,....,yn for which product of probabilities is maximized
- Cannot find the exact MAP efficiently since fully connected Bayesian network ⇒ intractable junction tree. The states z are high-dimensional real-vectors.
- Solution: approximate inference
  - Greedy
  - Beam-search

Encoder-decoder for sequence to sequence learning



Predicted sequence: **y**

Choose high probability token and feed to next step.

$$\Pr(y_t | y_1, \ldots, y_{t-1}, \mathbf{x})$$

RNN to generate **y**

RNN e.g. LSTMs to summarize x token-by-token

Embedding layer to convert each word to a fixed-D real vector

Context: **x**

From https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-2/

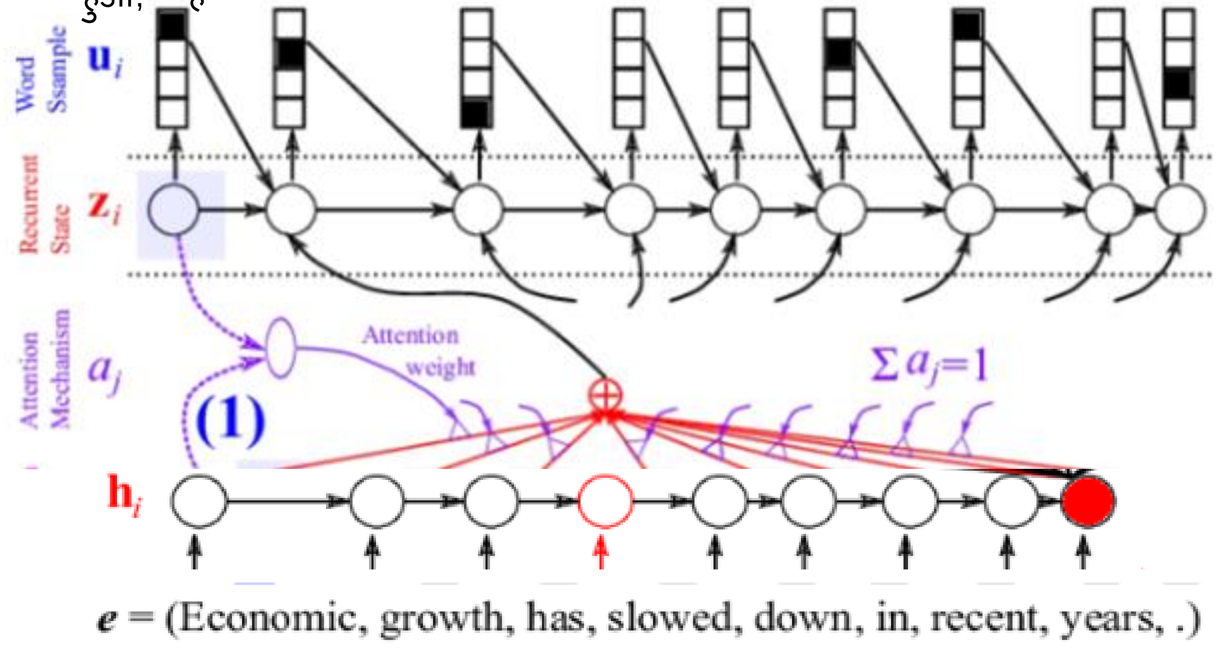
Where does the encoder-decoder model fail?

- Single vector cannot capture enough of input.
  - Fix: Attention (Bahdanau 2015, several others)
- Slow training: RNNs processed sequentially, replace with
    - CNN (Gehring, ICML 2017)
    - Transformer (Self Attention(Vaswani, June 2017))
- Training loss flaws
  - Global loss functions

Single vector not powerful enough ---> revisit input

Deep learning term for this ⇒ Attention!

H = हाल, के, वर्षों, में, आर्थिक, विकास, धीमा, हुआ, है

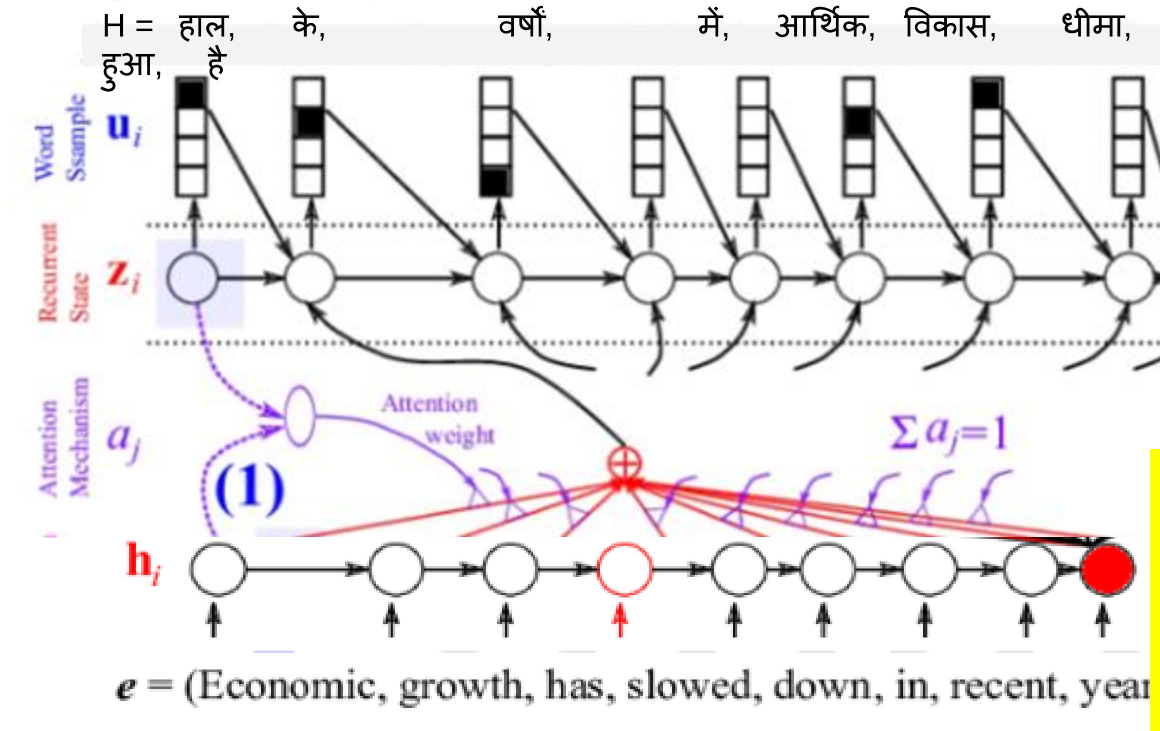


How to learn attention automatically, and in a domain neutral manner?

From https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-2/

Single vector not powerful enough ---> revisit input

Deep learning term for this ⇒ Attention!



$$\mathrm{Attn}(z_{t-1}, h_i) = A_{ti}$$

$$\alpha_{ti} = \frac{\exp(A_{ti})}{\sum_p \exp(A_{tp})}$$

$$v_{x,t} = \sum \alpha_{ti} h_i$$

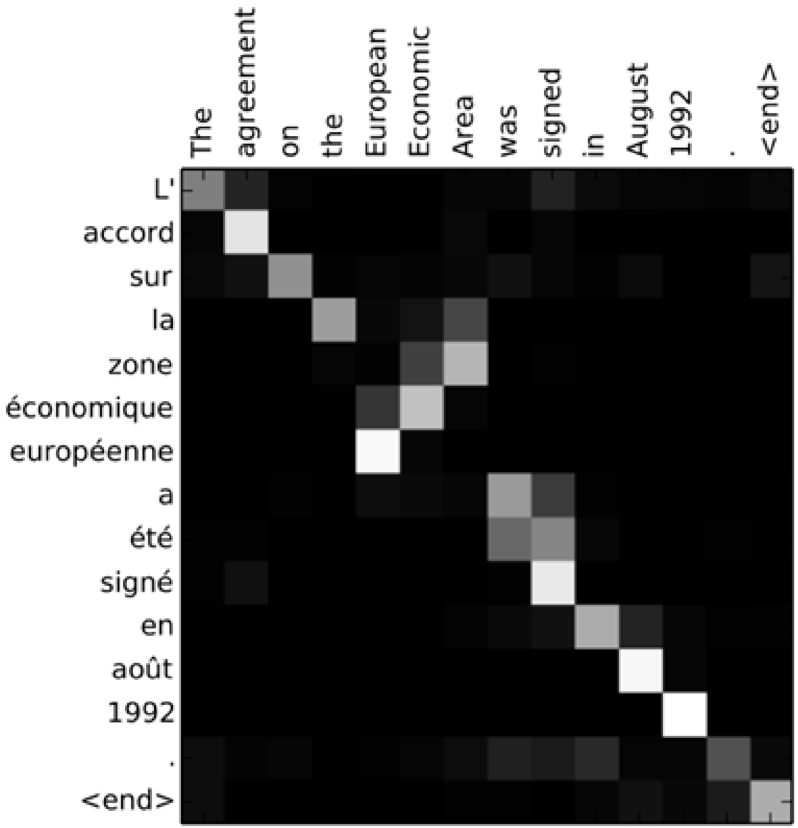
End-to-end trained and magically learns to align automatically given enough labeled data

From https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-2/

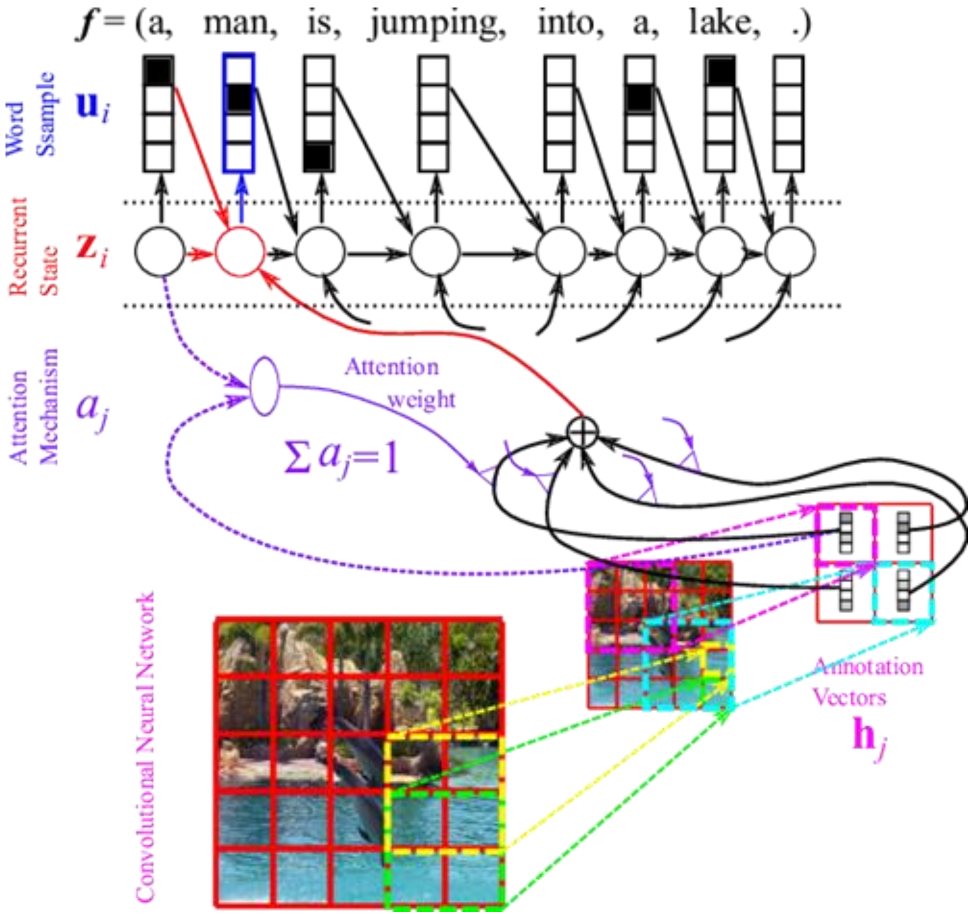
Example of attention in translation

Nice animated explanations for attention.

https://distill.pub/2016/augmented-rnns/#attentional-interfaces



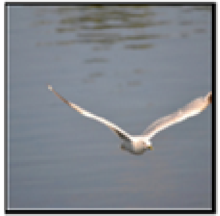
Same attention logic applies to other domains too



Attention over CNN-derived features of different regions of image

From https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-2/

Attention in image captioning. Attention over CNN



A bird flying over a body of water.

A woman is throwing a frisbee in a park.

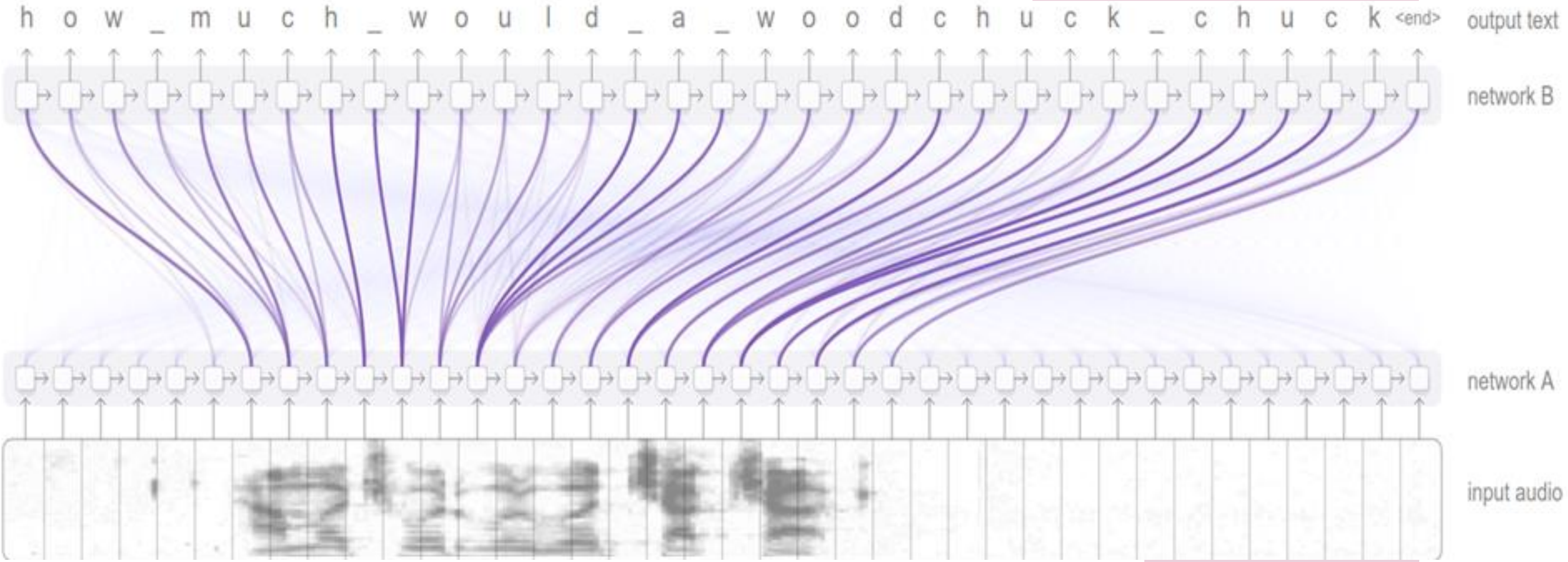
A dog is standing on a hardwood floor.

A stop sign is on a road with a

From https://arxiv.org/pdf/1502.03044v3.pdf

Attention in Speech to Text Models

Predicted sequence: **y**

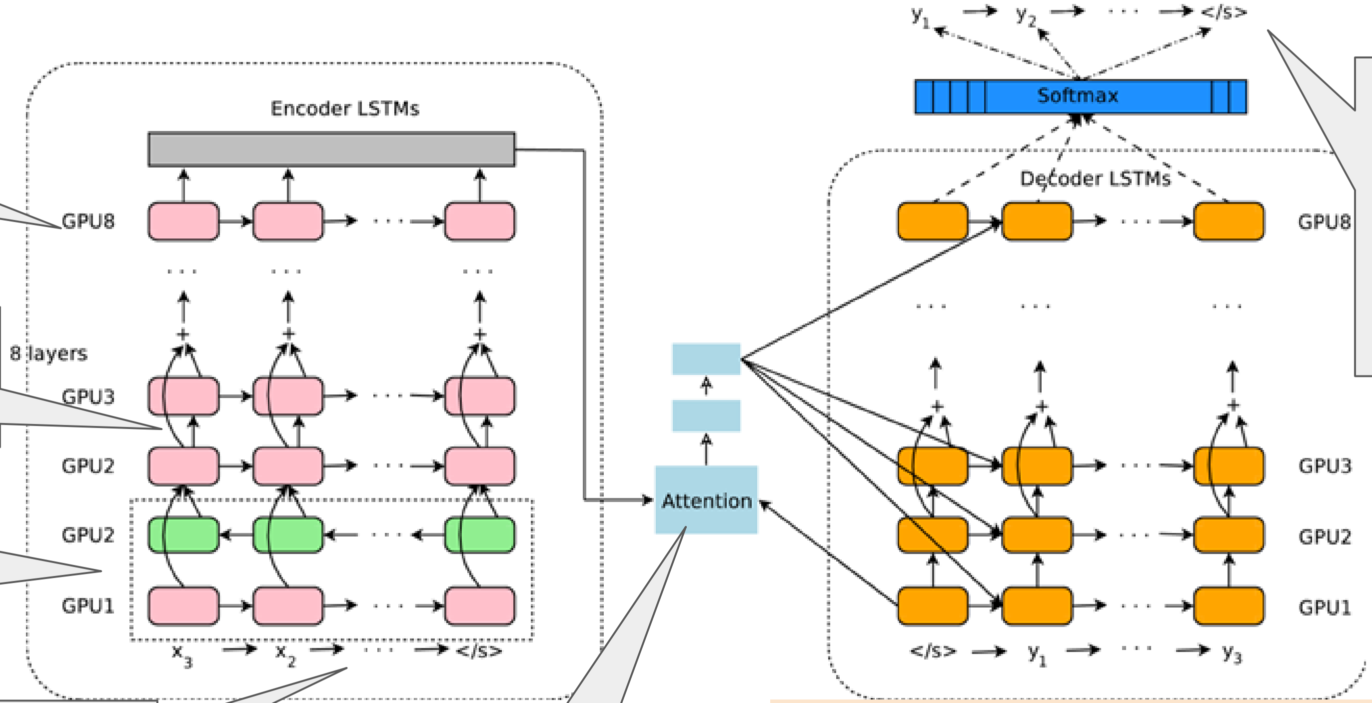


Context: **x**

We see that attention is focussed in middle part and nicely skips the prefix and suffix that is silence.

Diagram from https://distill.pub/2016/augmented-rnns/

Google’s Neural Machine Translation (GNMT) model



8 layers

Residual connections

Bidirectional LSTMs

Special wordpiece tokenization to handle rare words

2-layer attention logic

Length normalization, coverage penalty, low-precision inference

Works on many language pairs
60% better than existing phrase based system on human evaluation.

Results

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

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

Summary

- Deep learning based models for sequence prediction has revolutionized and unified many diverse domains.
- 2015-2018 has seen several improvements to the encoder-decoder method
  - Increase capacity via input attention
  - Eschew RNN bottleneck via multi-layer self-attention
  - Fix loss function via better calibration and global conditioning
- Other interesting developments not covered
  - Memory networks for remembering rare events (Kaiser, ICLR 2017)

What next?

- Move away from black-box, batch-trained, monolithic models to transparent models with more control from humans and evolving continuously.

- Generalize to other structured learning tasks
    - No natural ordering of variables.

Thank you.

Where does the encoder-decoder model fail?

- Single vector cannot capture enough of input
  - Fix: Attention
- Slow training: RNNs processed sequentially, replace with
    - CNN (Gehring, ICML 2017)
    - Attention (Vaswani, June 2017)
- Training loss flaws
  - Systematic bias against long sequences
  - Not aligned with whole sequence error during inference
    - Generate sequences during training, score their errors and minimize (Ranzato 2016, Wiseman & Rush, 2016, Shen 2016, Bahdanau 2016, Norouzi 2016)

Attention is enough. No need for RNN

Attention weighted sum of previous layer

Edge weights determined by self-attention. Multiple of these

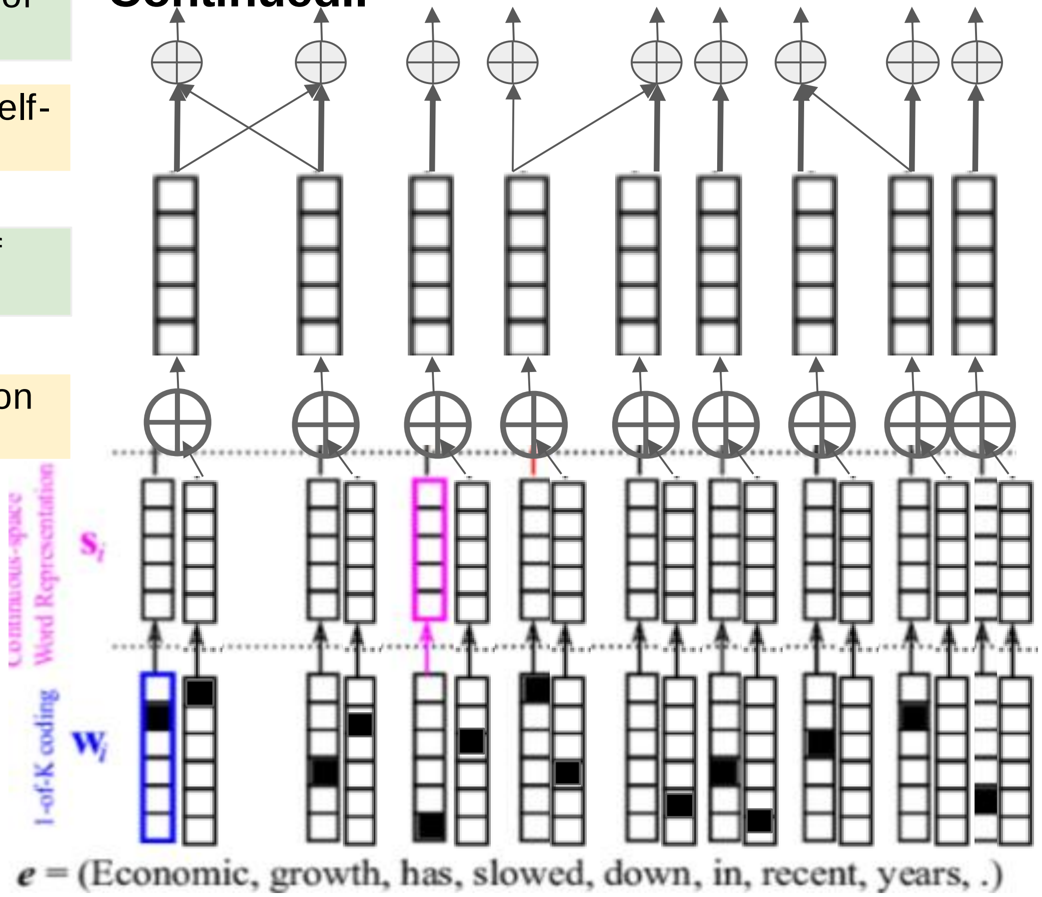
Positional embedding of each input word

Sum up word and position embedding

Compute position embedding, lookup word embedding

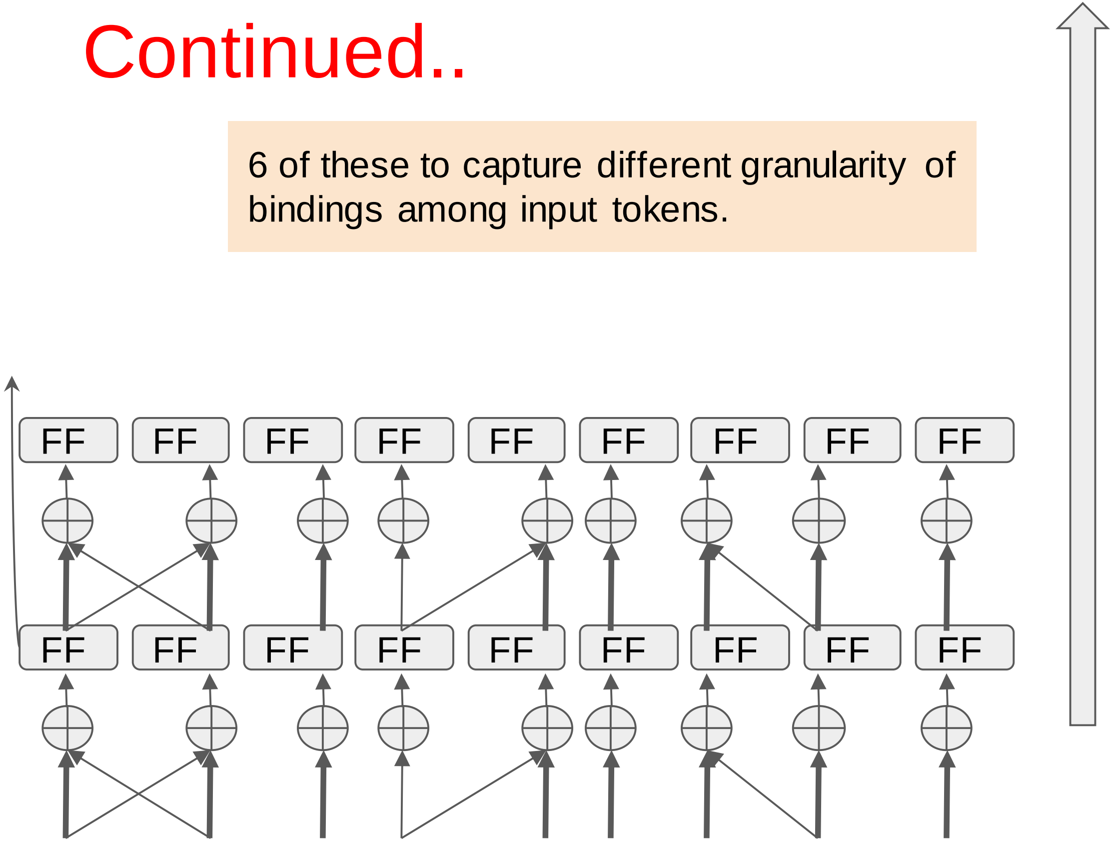
One-hot word, and position(1,2..)

Continued..



Continued..

6 of these to capture different granularity of bindings among input tokens.

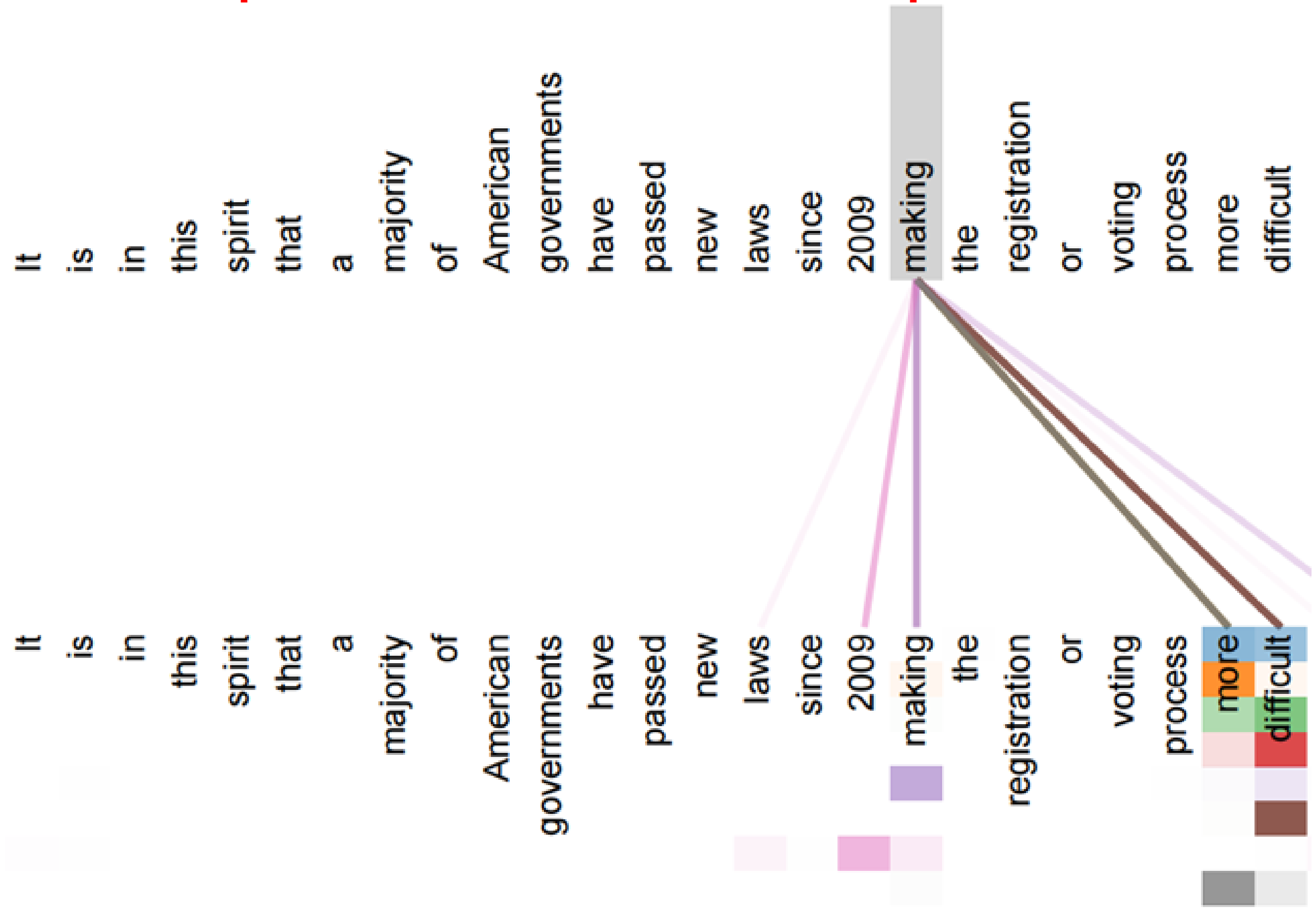


Repeat similar 6-layers to replace RNN for decoder too and between decoder and encoder

Tokens at all positions processed in parallel --- only sequentiality among the 6 layers which are fixed.

Author’s slides https://www.slideshare.net/ilblackdragon/attention-is-all-you-need

Example: how attention replaces RNN state



Attention around "making" converts it to phrase "making more difficult"

Performance

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [17]	23.75			
Deep-Att + PosUnk [37]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [36]	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 [31]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [37]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [36]	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	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

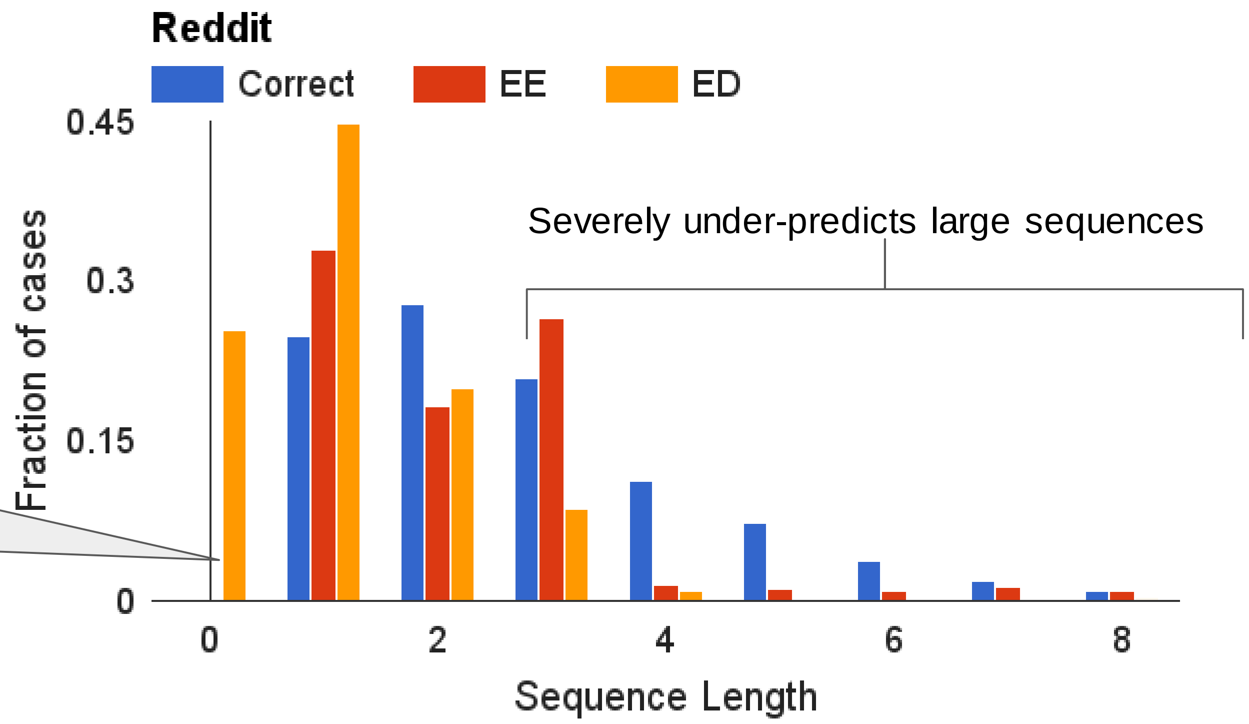
RNNs/CNNs no longer indispensable for sequence prediction

Attention captures relevant bindings at much lower cost

Where does the encoder-decoder model fail?

- Single vector cannot capture enough of input.
  - Fix: Attention
- Slow training: RNNs processed sequentially, replace with
    - CNN (Gehring, ICML 2017)
    - Attention (Vaswani, June 2017)
- Training loss flaws
  - Poor calibration
  - Not aligned with whole sequence error during inference
    - Generate sequences during training, score their errors and minimize (Ranzato 2016, Wiseman & Rush, 2016, Shen 2016, Bahdanau 2016, Norouzi 2016)

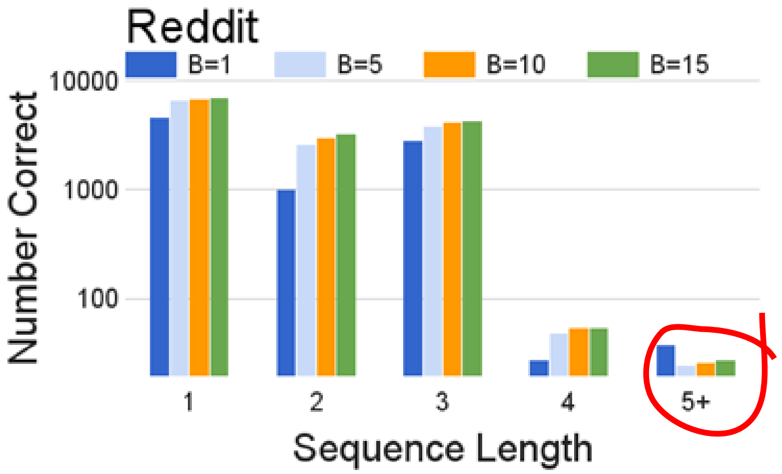
Bias against longer sequences

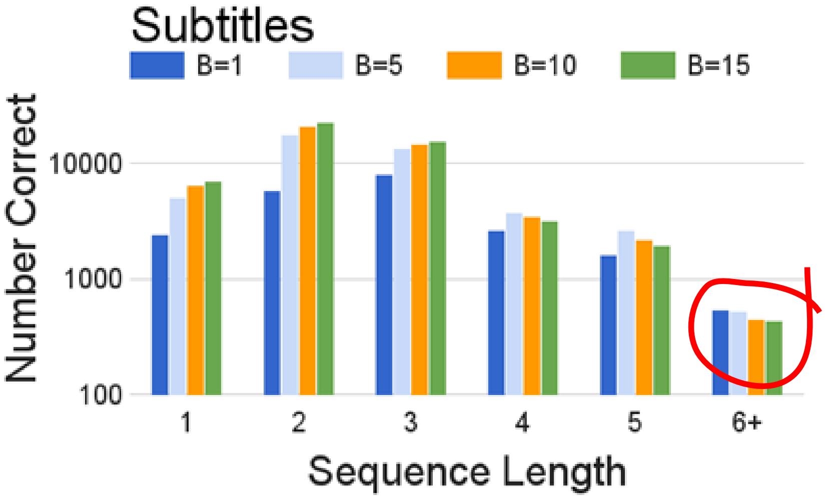


26% ED predictions of zero length. None in data.

ED over-predicts short sequences

Surprising drop in accuracy with better inference





For long sequences, accuracy drops when inference predicts a higher scoring sequence ---- why?

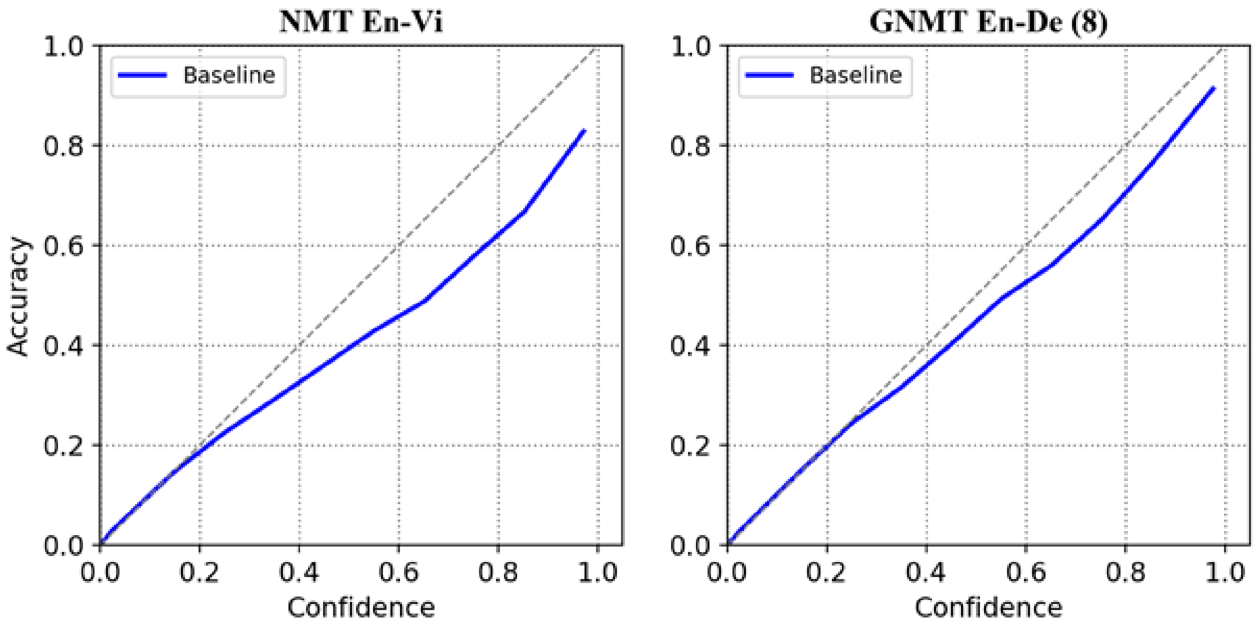
Two Causes

1. Lack of calibration
2. Local conditioning

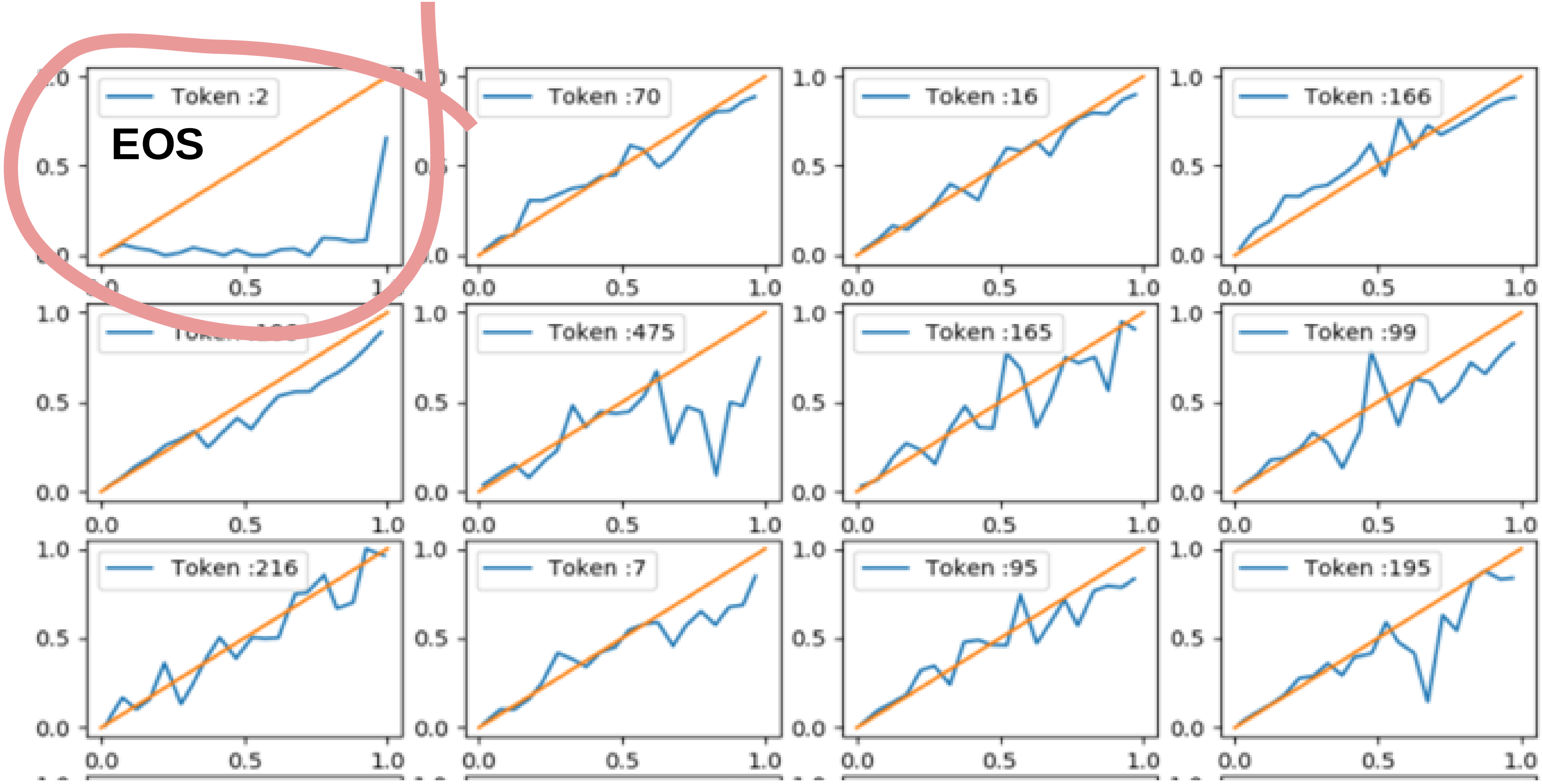
Lack of calibration

- Next token probabilities not well-calibrated.
  - A 0.9 probability of y_t = “EOS”, does not imply 90% chance of correctness.
- Bane of several modern neural architectures e.g. Resnets, not just sequence models
  - High in accuracy but low in reliability!
    - Mostly over-confident.
  - See: On Calibration of Modern Neural Networks, ICML 2017

Calibration plots



Investigating reasons for poor calibration



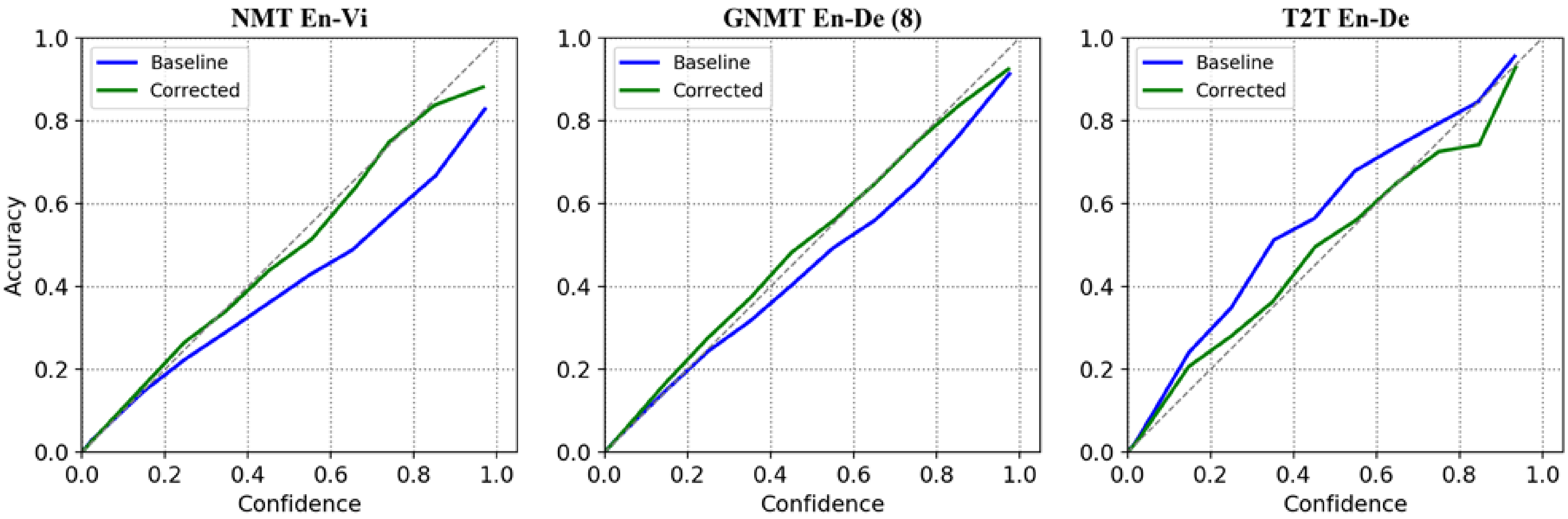
Reasons for poor calibration

- Observations
    a. End of sequence token is seriously over-confident
    b. Calibration is worse when encoder attention is diffused.
    c. Other unexplained reasons.

Kernel embedding based trainable calibration measure

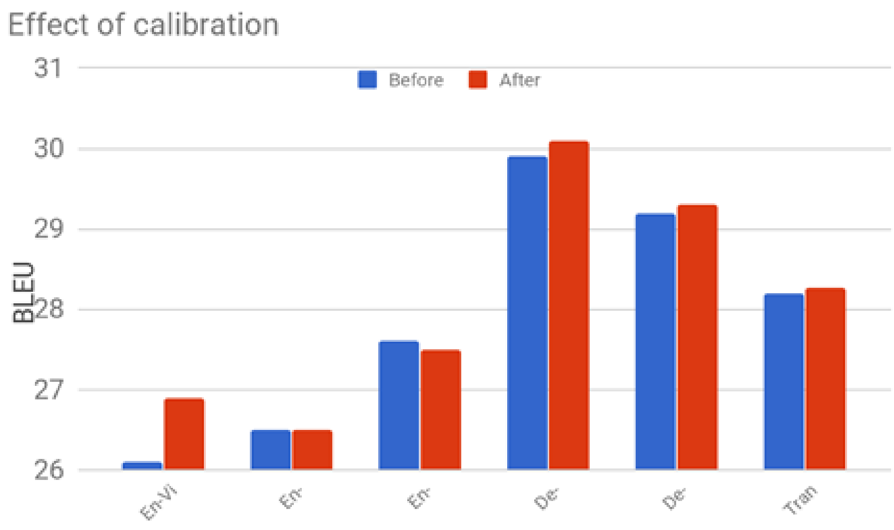
- Train models to minimize weighted combination of 0/1 error and calibration of confidence scores.

Corrected calibrations



Fixing calibration leads to higher accuracy

1. Beam search for predicting highest probability sequence
   a. Grows token-by-token a beam of highest scoring prefixes
   b. Poor calibration misleads beam-search



Two Causes

1. Lack of calibration
2. Local conditioning

Problems of local conditioning

Local conditioning causes the log-probability of each correct token to saturate (get very close to zero) even when the correct sequence does not have the highest probability.

$$P(\mathbf{y}|\mathbf{x}, \theta) = \prod_{t=1}^{n} P(y_t|y_1, \ldots, y_{t-1}, \mathbf{x}, \theta)$$

Local conditioning for sequence prediction

$\log \Pr(y_t | y_1, \ldots, y_{t-1})$ Positive sequence: “S,1,1,1,1,1,1,E”, Negative sequence: “S,0,E”.

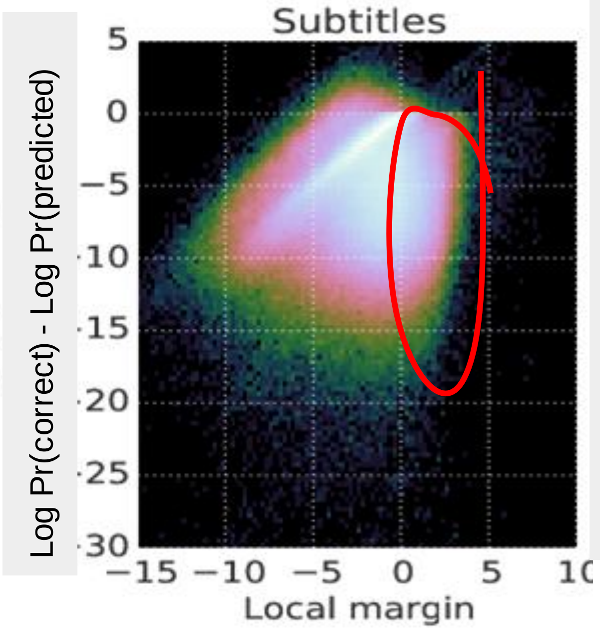
t=	1	2	3	4	5	6	7	8
S	-0.01	-1.6	-1.6	-1.6	-1.6	-1.6	-1.6	-6
1	-6	**- 0.4**	**- 0.3**	**- 0.3**	**- 0.3**	**- 0.3**	**- 0.3**	-6
0	-6	-1.4	-1.5	-1.5	-1.5	-1.5	-1.5	-6
E	-6	-1.8	-1.7	-1.6	-1.5	-1.5	-1.5	-0.01

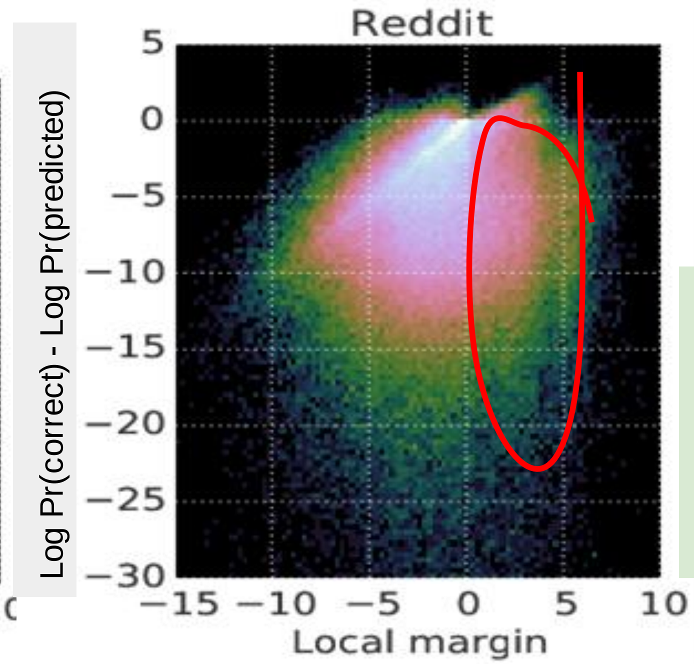
Margin between position and negative sequence optimized by ED local loss is -0.4 - (-1.4) = 1!

-6
-6
-6
-0.01

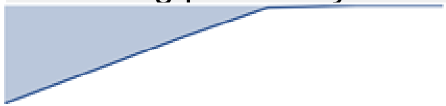
Log-probability of positive sequence = -1.9
Log-probability of negative sequence = -0.4
Margin between positive and negative sequence = -1.5!

ED objective is zero even when prediction is wrong

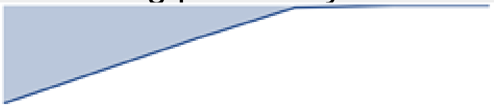




-15 -10 -5 -0.3 -e-3 -e-5
Local log probability



-15 -10 -5 -0.3 -e-3 -e-5
Local log probability -->



More training data will not help if your training loss is broken!

How to fix the ED training loss?

Avoid local conditioning, use global conditioning

$$\Pr(Y_i|X_i,\theta) = \frac{e^{S(Y_i|X_i,\theta)}}{\sum_{Y \in \text{sample}} e^{S(Y|X_i,\theta)}}$$

Use for

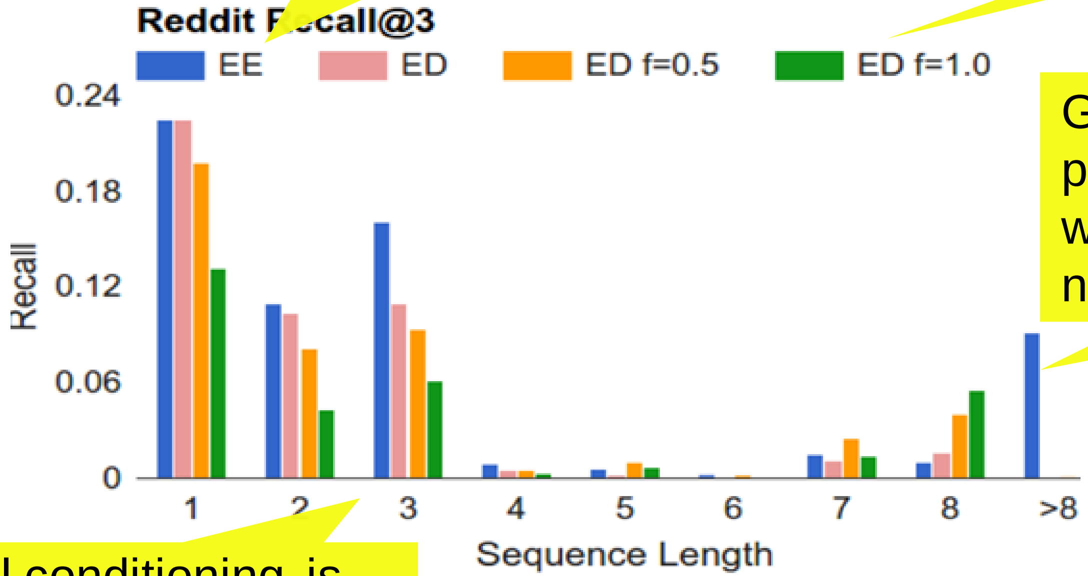
- Applications, like conversation where response restricted to be from a whitelist of responses
- Else, sample responses adaptively during training

More details in Length bias in Encoder Decoder Models and a Case for Global Conditioning by Siege and Sarawagi. EMNLP'16

Results

A method using global conditioning

Length normalized encoder-decoder models



Global conditioning predicts long sequences whereas ED predicts none

Global conditioning is more accurate

Thank you!

Properties of a good loss function for training

- Scoring models

$$(X, Y) \rightarrow \text{Model } (\boldsymbol{\theta}) \rightarrow S(Y|X,\boldsymbol{\theta}) \in \mathbf{R}$$

- Inference: find Y with highest score
- Training: minimize loss per labeled instance $\{(X_i, Y_i)\}$
  - If loss ~ 0, then correct output Yi has the highest score.
  - Not true for encoder decoder models!

Peculiar biases of predictions from ED model

- ED over-predicting short sequences
  - Even after accounting for the fact that short messages are more common given any particular context.
- Increasing the beam width sometimes decreased quality!

These observations are on models trained with billions of examples for a conversation task.

Datasets

- Reddit – comments on user posts
  - 41M posts, 501M comments
- Open Subtitles – subtitles on non-English movies
  - 319M lines of text

For each data set:

- 100K top messages = predicted set.
- 20K top tokens used to encode tokens into ids.