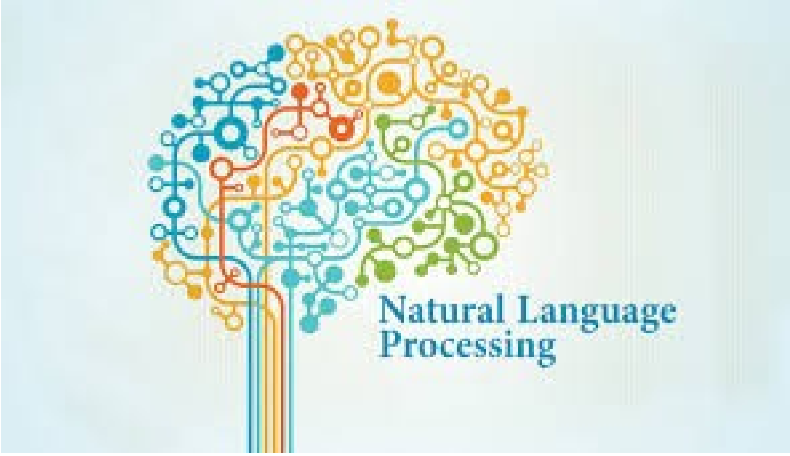
Text Sequence Processing

Narada Warakagoda



Topics

- Word Representations
- Sequence-to-sequence transformation
  - Recurrent networks
  - Convolutions networks
  - Self-attention (Transformers)
- Reinforcement Learning

Word Representations

Why Word Representations?

- Words are symbols
- Neural networks operate on numerical values

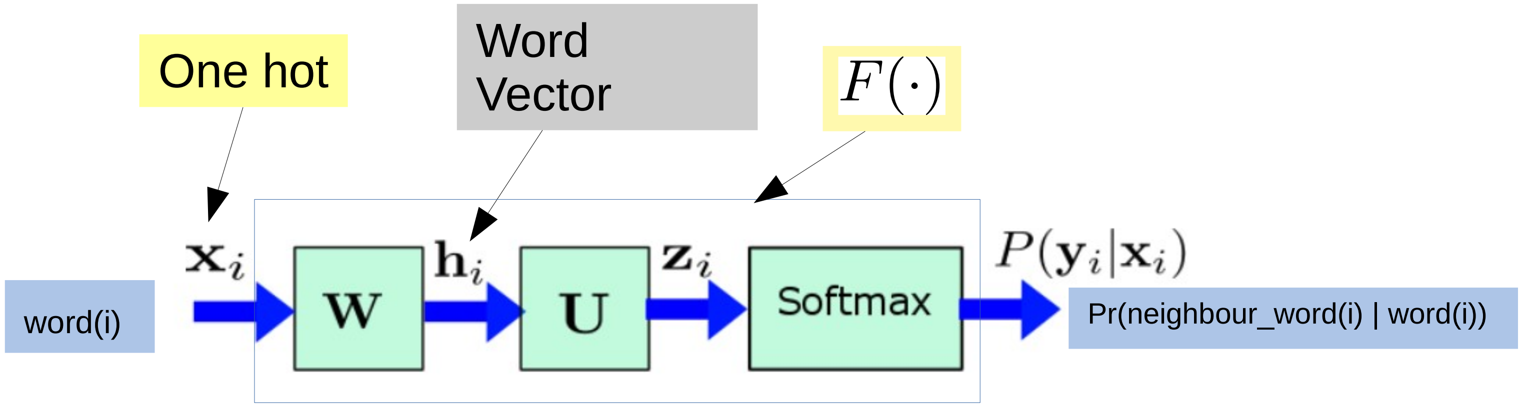
Trivial Approach

- One Hot encoding
  - Use the word index in vector form
- Example
  - Consider a vocabulary of 5 words

1	Man	[1,0,0,0,0]
2	Woman	[0,1,0,0,0]
3	Boy	[0,0,1,0,0]
4	Girl	[0,0,0,1,0]
5	House	[0,0,0,0,1]

- Disadvantages
  - Dimension of the representation vector would be very high for natural vocabularies
  - All vectors are equally spread (vector similarity does not represent semantic similarity)

Better Approach



$\mathbf{x}_i \in \mathbb{R}^{V\times 1}$, $\mathbf{h}_i \in \mathbb{R}^{d\times 1}$, $\mathbf{W} \in \mathbb{R}^{V\times d}$, $\mathbf{U} \in \mathbb{R}^{V\times d}$

- Projection:

$$\mathbf{h}_i = \mathbf{W}^T \mathbf{x}_i$$

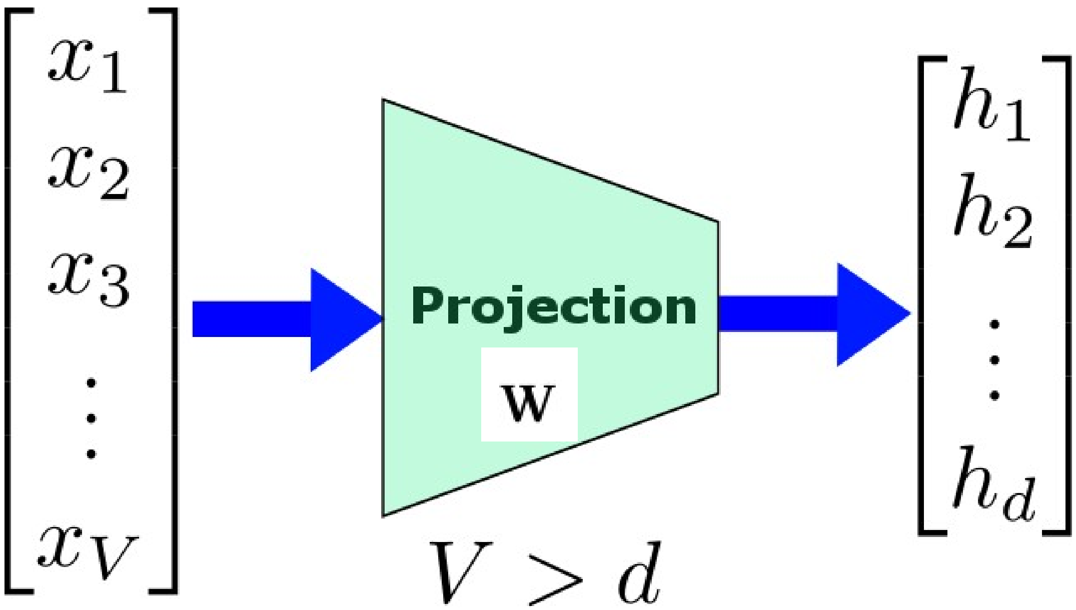
- Second layer:

$$\mathbf{z}_i = \mathbf{U}\mathbf{h}_i$$

- Softmax:

$$P(y_i = j|\mathbf{x}_i) = \frac{\exp(z_i(j))}{\sum_k \exp(z_i(k))}$$

Issue1: High Dimension

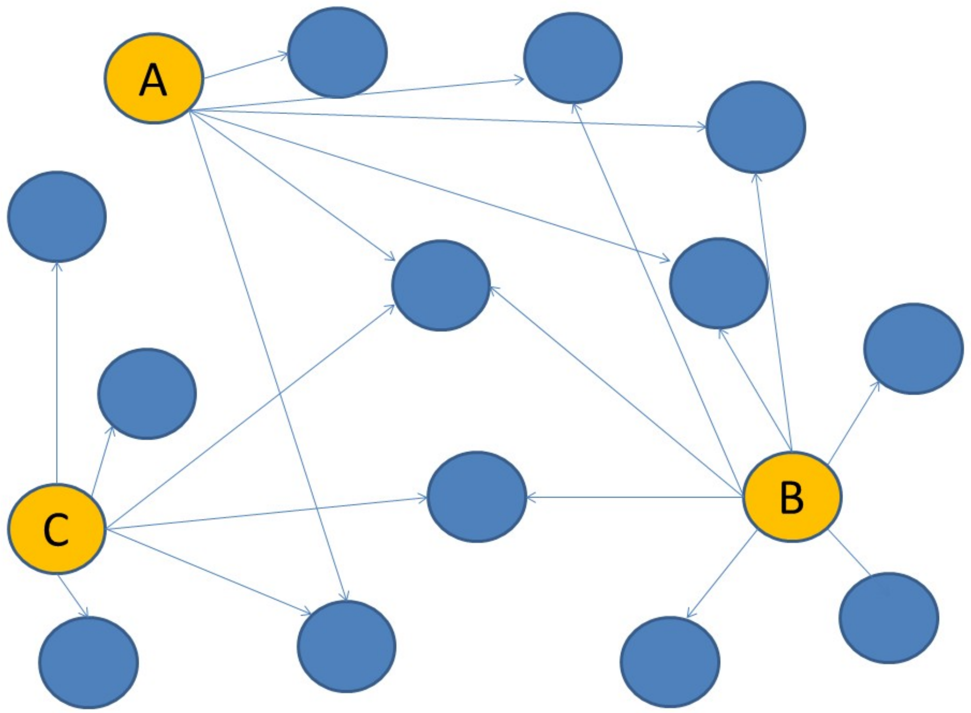


- Project one-hot encoded vectors to a lower dimensional space (Reduce the dimension of the representation)
- Also known as ***embedding***
- Linear projection = Multiplication by a $\mathbf{h}_{1\times d} = \mathbf{x}_{1\times V}\mathbf{W}_{V\times d}$

Issue 2: Similar Words

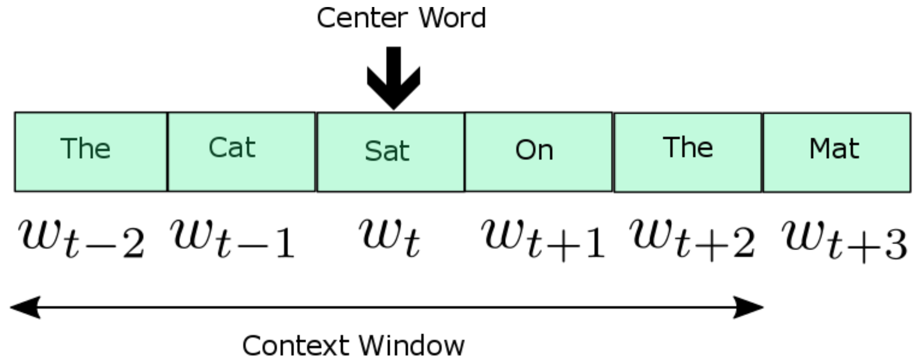
- Force vector distance between similar words to be low
- How to quantify word similarity?

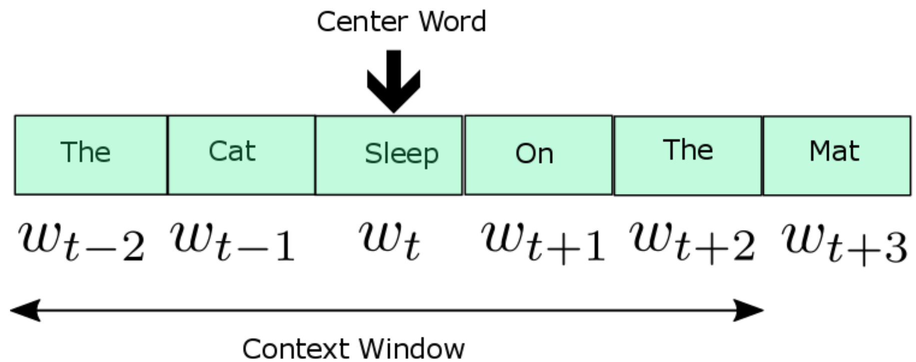
Quantifying Word Similarity



- A is ”more similar” to B than C ?
- A is ”more similar” to C than B ?

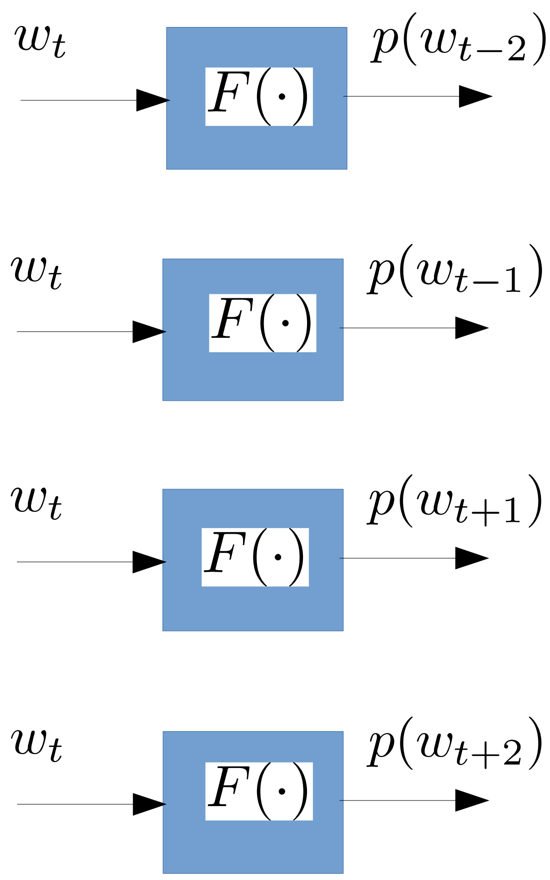
Quantifying Word Similarity





- Context of a word = Words occurring before and after within a predefined window
- Words that have similar contexts, should be represented by word vectors close to each other

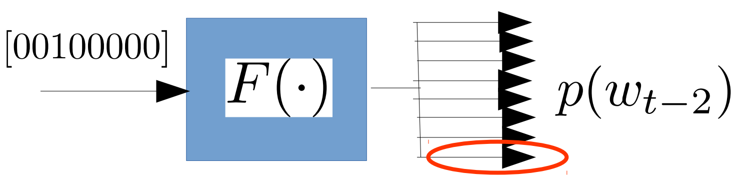
Training Objective

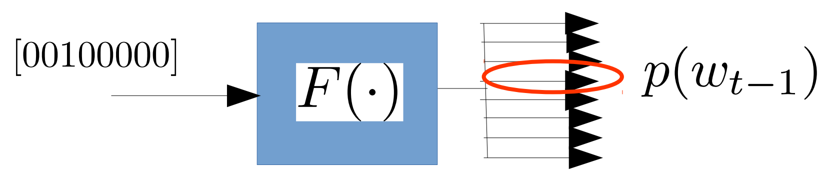


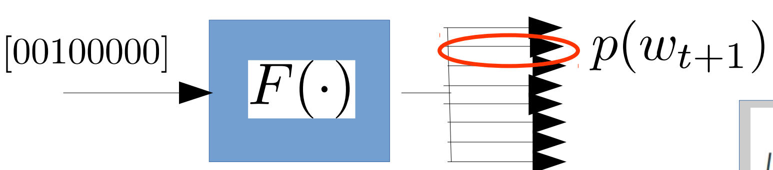
- Train $F(\cdot)$ to maxi $L = \prod_{t=1}^{T} \prod_{-C \leq j \leq C, j \neq 0} P(w_{t+j}|w_t)$

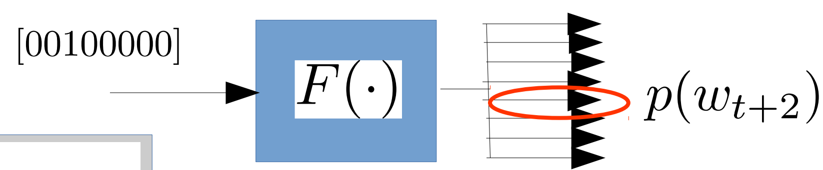
Practical Details

Word Index y	One Hot representation x	Word
1	00000001	
2	00000010	w_{t+1}
3	00000100	
4	00001000	w_{t-1}
5	00010000	w_{t+2}
6	00100000	w_t
7	01000000	
8	10000000	w_{t-2}



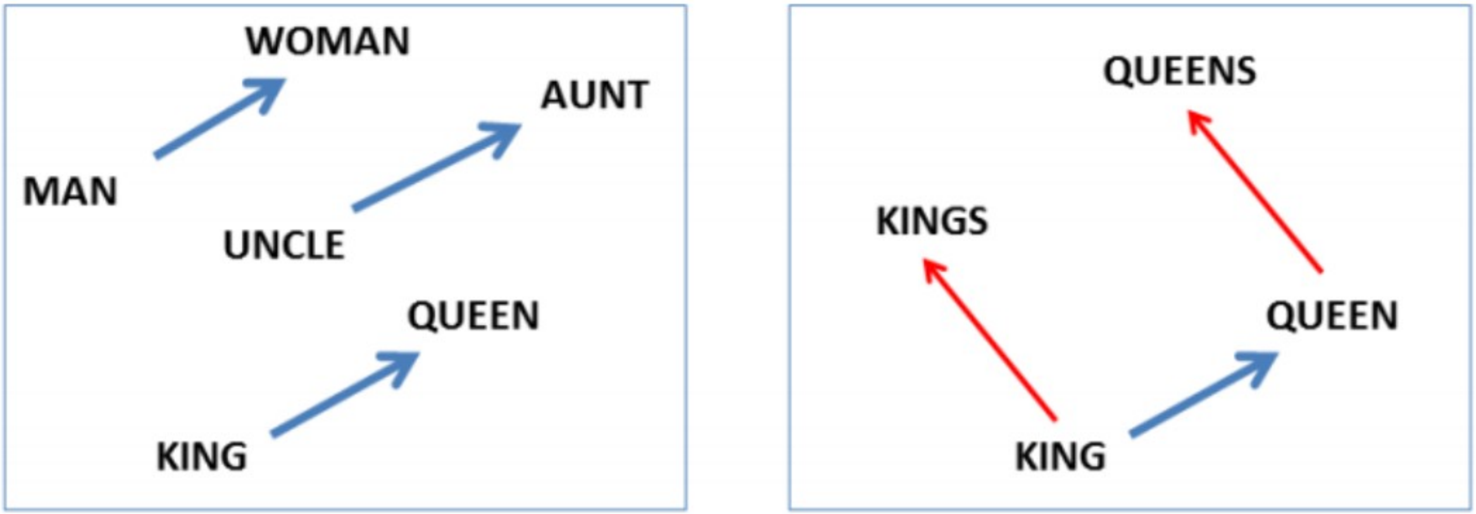






$$L = \prod_{t=1}^{T} \prod_{-C \leq j \leq C, j \neq 0} P(w_{t+j} | w_t)$$

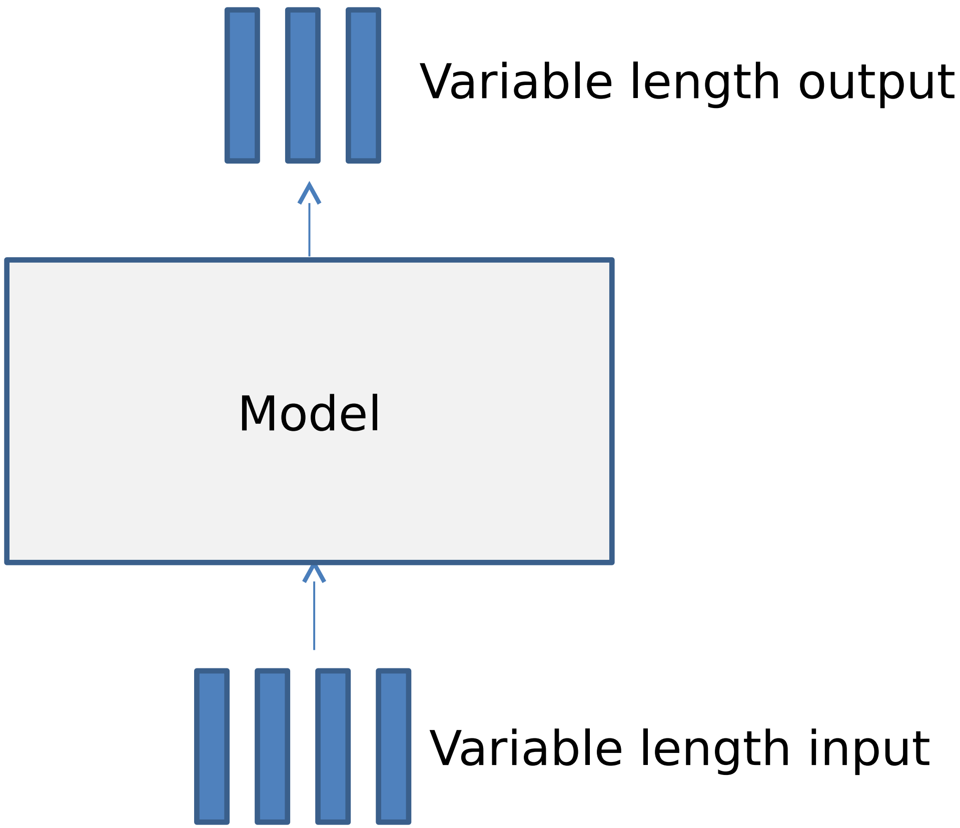
Word Vector Visualization



(Mikolov et al., NAACL HLT, 2013)

Sequence-to-sequence Transforms

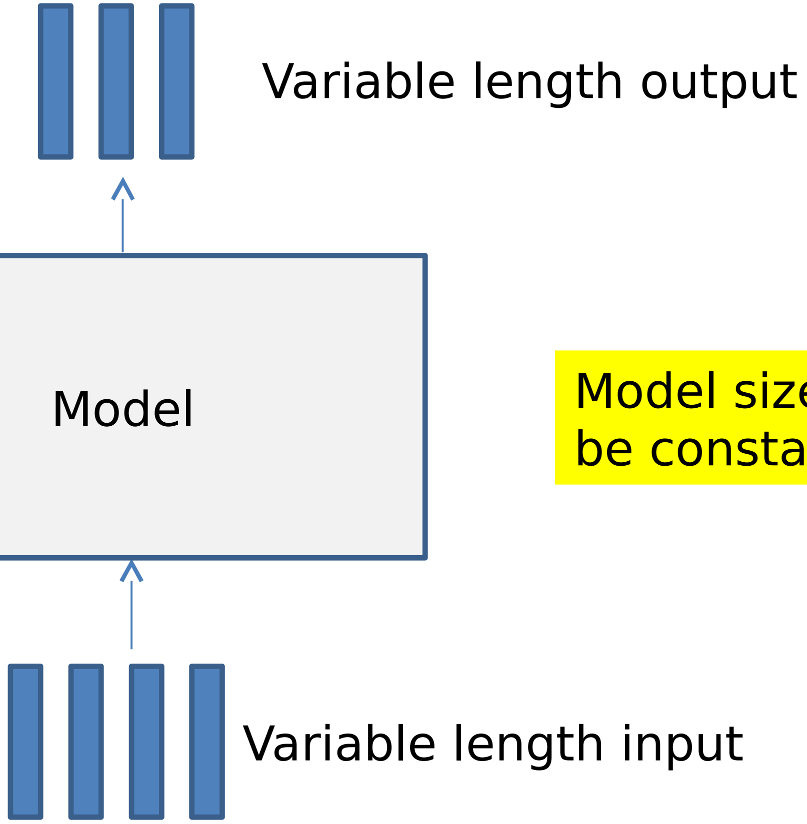
Seq2seq Transformation



Example Applications

- Summarization (extractive/abstractive)
- Machine translation
- Dialog systems /chatbots
- Text generation
- Question answering

Seq2seq Transformation



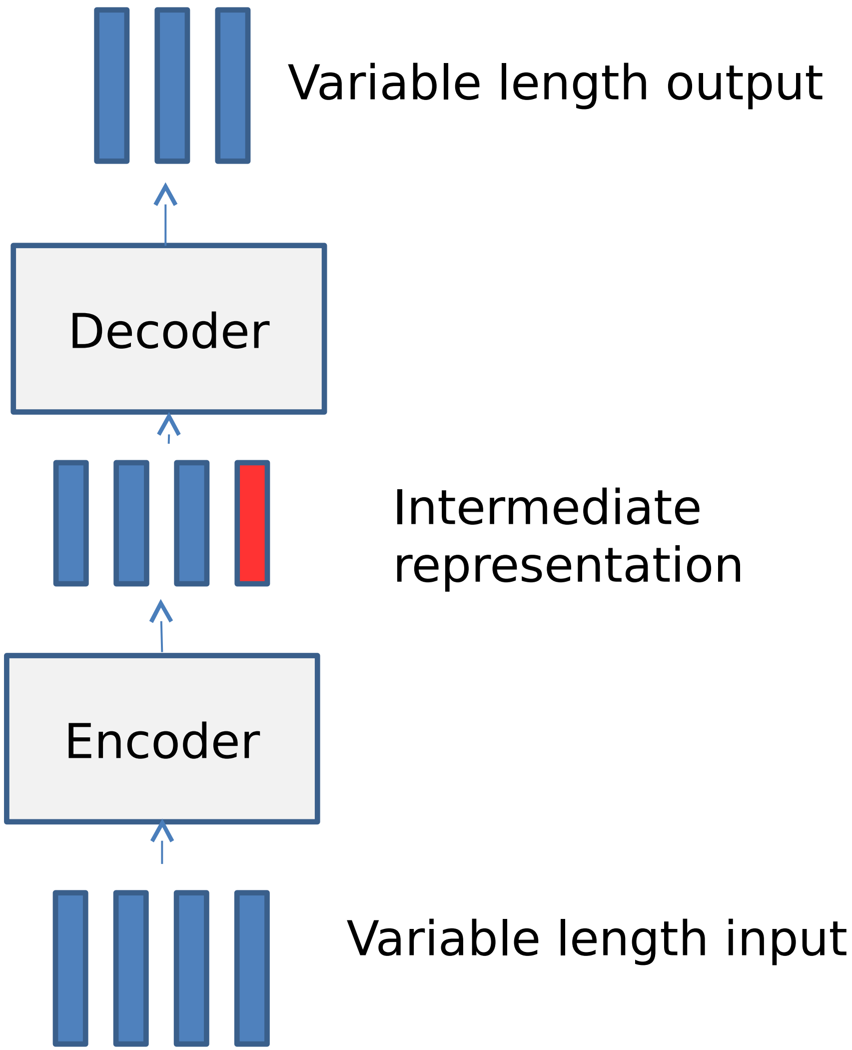
Model size should be constant.

Solution: Apply a constant sized neural net module repeatedly on the data

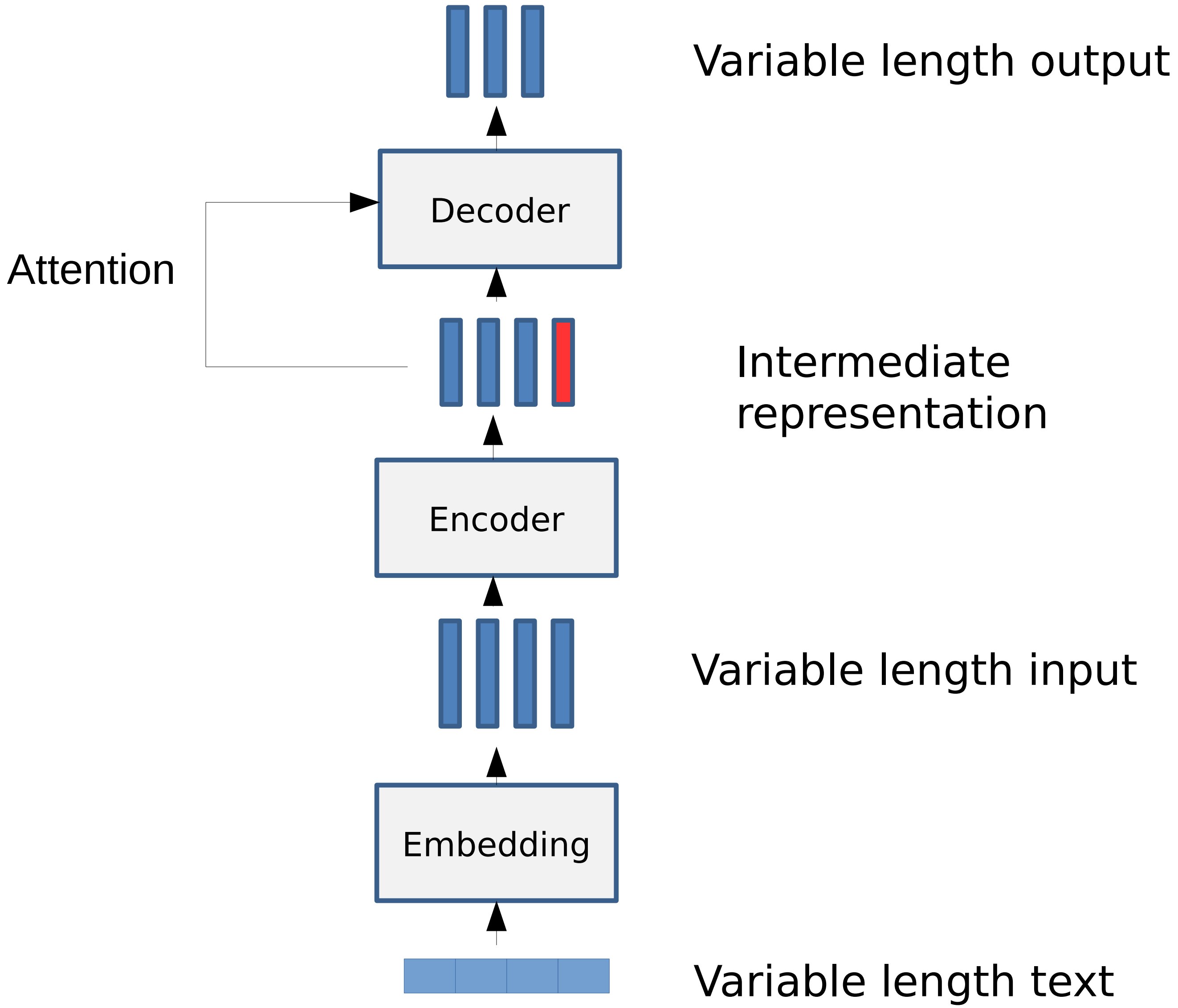
Possible Approaches

- Recurrent networks
  - Apply the NN module in a serial fashion
- Convolutions networks
  - Apply the NN modules in a hierarchical fashion
- Self-attention (Transformers)
  - Direct interaction in the inputs

Processing Pipeline



Processing Pipeline



Architecture Variants

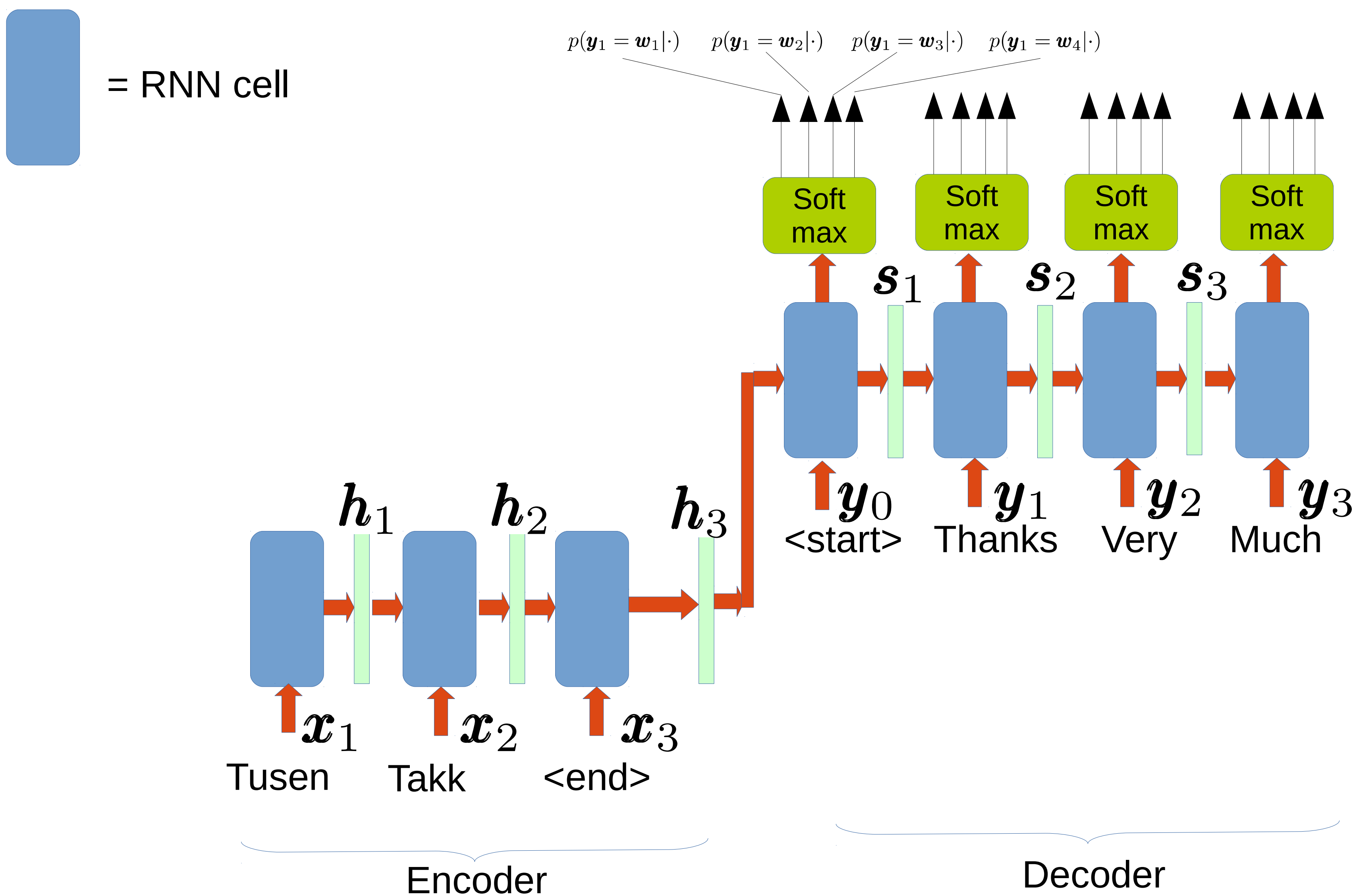
Encoder	Decoder	Attention
Recurrent net	Recurrent net	No
Recurrent net	Recurrent net	Yes
Convolutional net	Convolutional net	No
Convolutional net	Recurrent net	Yes
Convolutional net	Convolutional net	Yes
Fully connected net with self-attention	Fully connected net with self-attention	Yes

Possible Approaches

- Recurrent networks
  - Apply the NN module in a serial fashion
- Convolutions networks
  - Apply the NN modules in a hierarchical fashion
- Self-attention
  - Direct interaction in the inputs

RNN-decoder with RNN-encoder

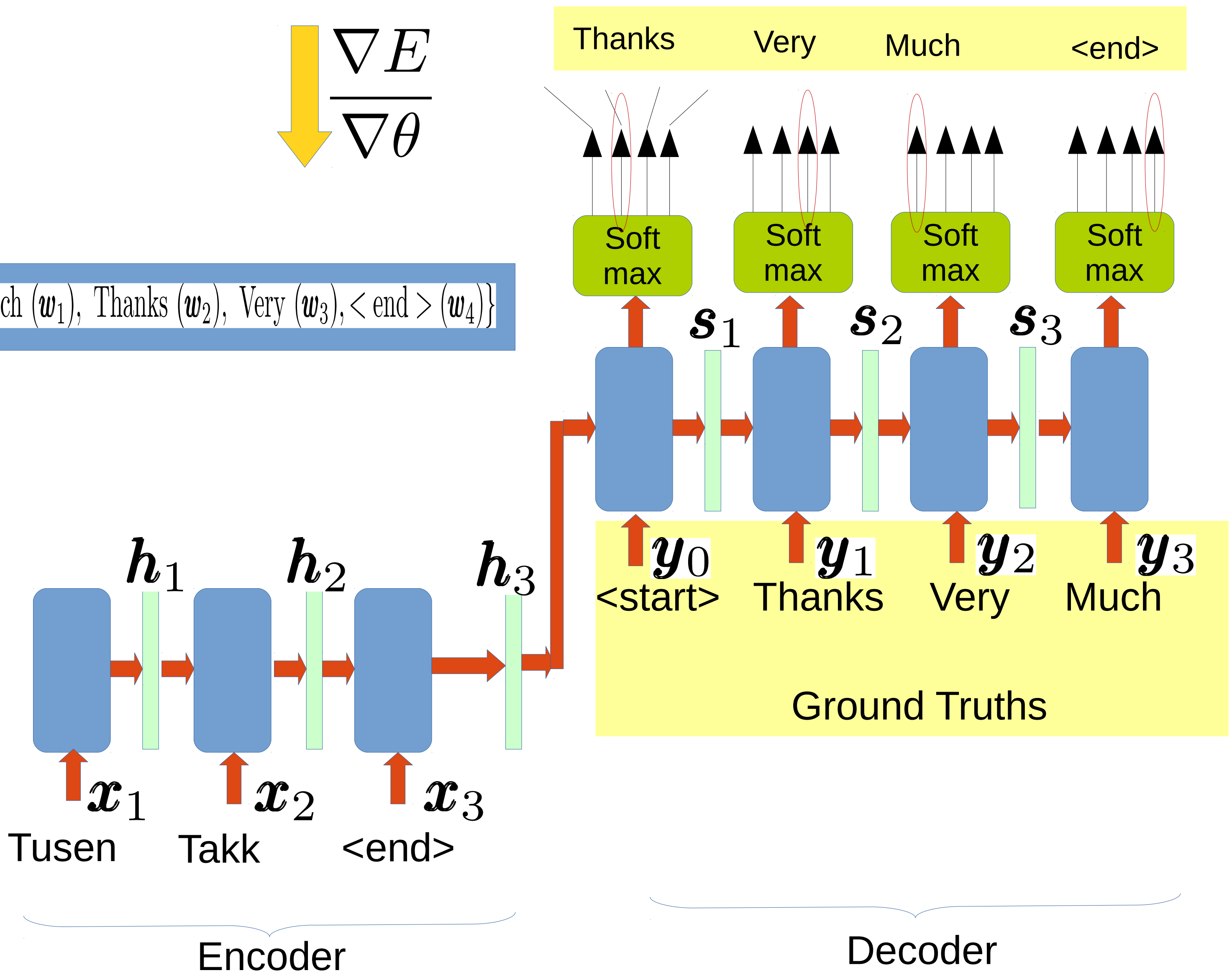
$$\text{Decoder vocabulary} = \{\text{Much } (\boldsymbol{w}_1),\ \text{Thanks } (\boldsymbol{w}_2),\ \text{Very } (\boldsymbol{w}_3), <\text{end}> (\boldsymbol{w}_4)\}$$



RNN-dec with RNN-enc, Training

$$E = \log L = \log\left[p(\boldsymbol{y}_1 = w_2|\boldsymbol{X}) \cdot p(\boldsymbol{y}_2 = w_3|w_2, \boldsymbol{X}) \cdot p(\boldsymbol{y}_3 = w_1|w_2, w_3, \boldsymbol{X}) \cdot p(\boldsymbol{y}_4 = w_4|w_2, w_3, w_1, \boldsymbol{X})\right]$$

$$\text{Decoder vocabulary} = \{\text{Much } (\boldsymbol{w}_1),\ \text{Thanks } (\boldsymbol{w}_2),\ \text{Very } (\boldsymbol{w}_3), <\text{end}> (\boldsymbol{w}_4)\}$$

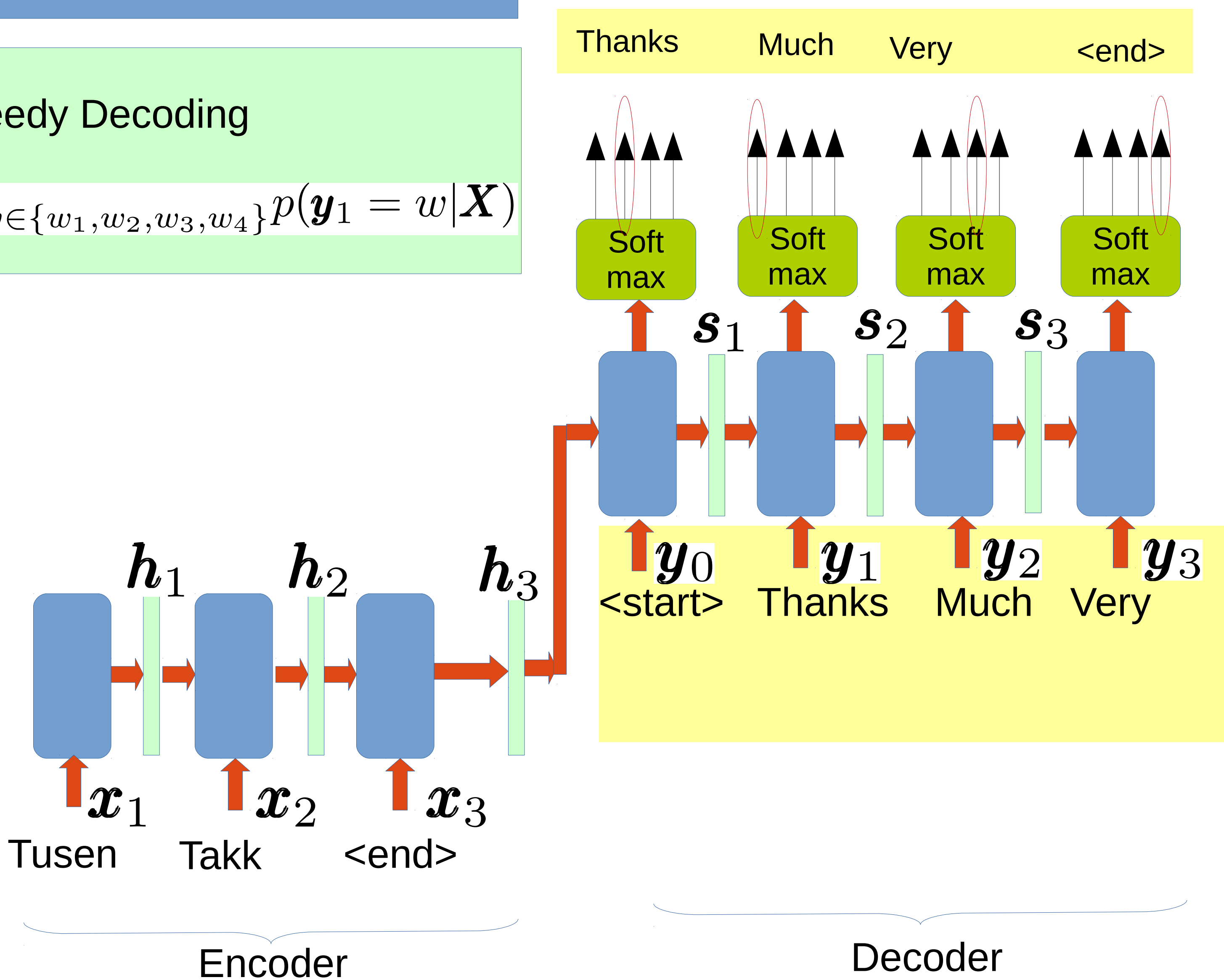


RNN-dec with RNN-enc, Decoding

$\text{Decoder vocabulary} = \{\text{Much } (\boldsymbol{w}_1),\ \text{Thanks } (\boldsymbol{w}_2),\ \text{Very } (\boldsymbol{w}_3), < \text{end} > (\boldsymbol{w}_4)\}$

Greedy Decoding

$$\boldsymbol{y}_1 = \text{argmax}_{w \in \{w_1, w_2, w_3, w_4\}} p(\boldsymbol{y}_1 = w | \boldsymbol{X})$$



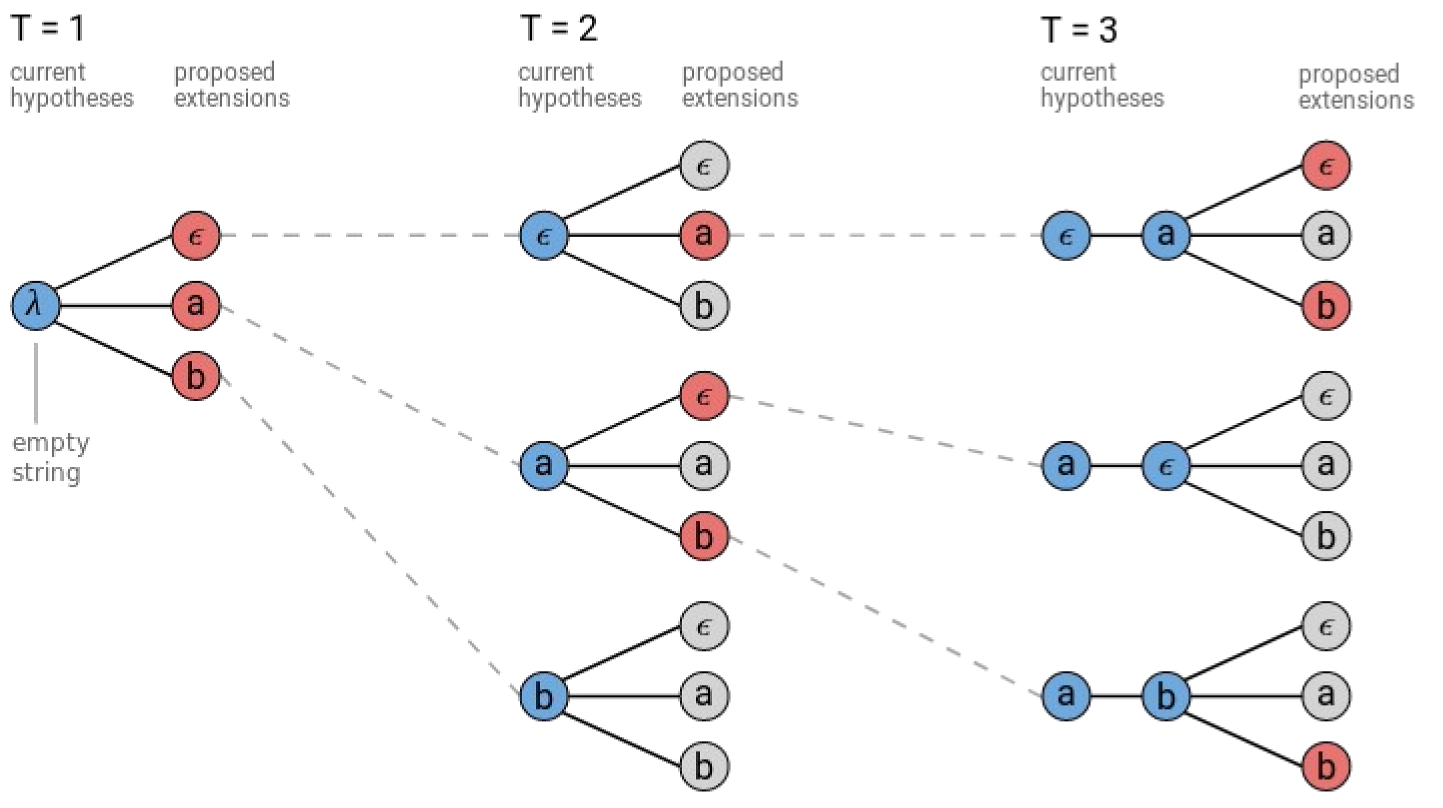
Decoding Approaches

- Optimal decoding

  Find $\boldsymbol{w} = \{w_1, w_2, w_3, w_4\}$ such that $p(w_1, w_2, w_3, w_4|\boldsymbol{X})$ is maximum

- Greedy decoding
  - Easy
  - Not optimal
- Beam search
  - Closer to optimal decoder
  - Choose top N candidates instead of the best one at each step.

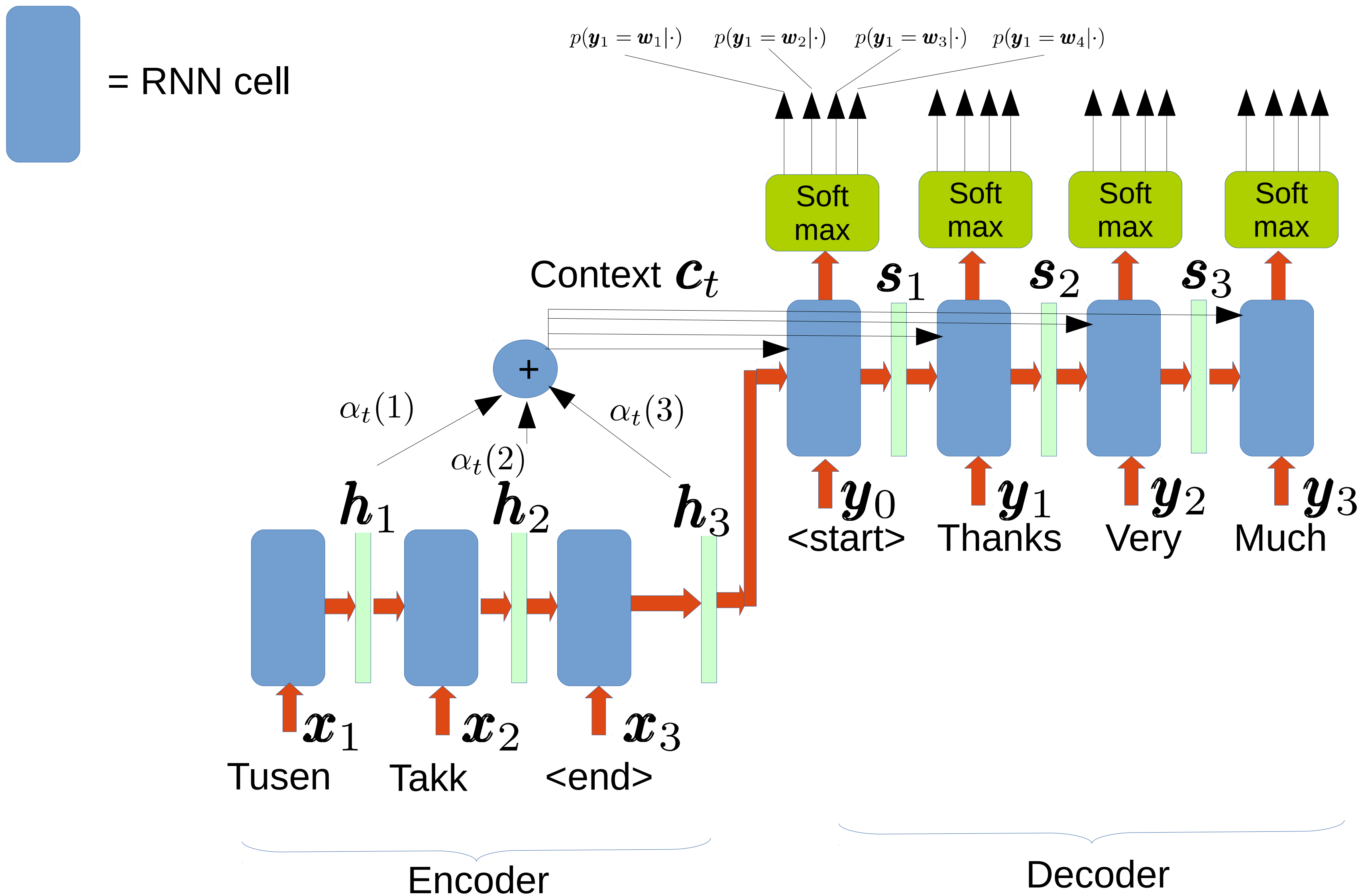
Beam Search Decoding



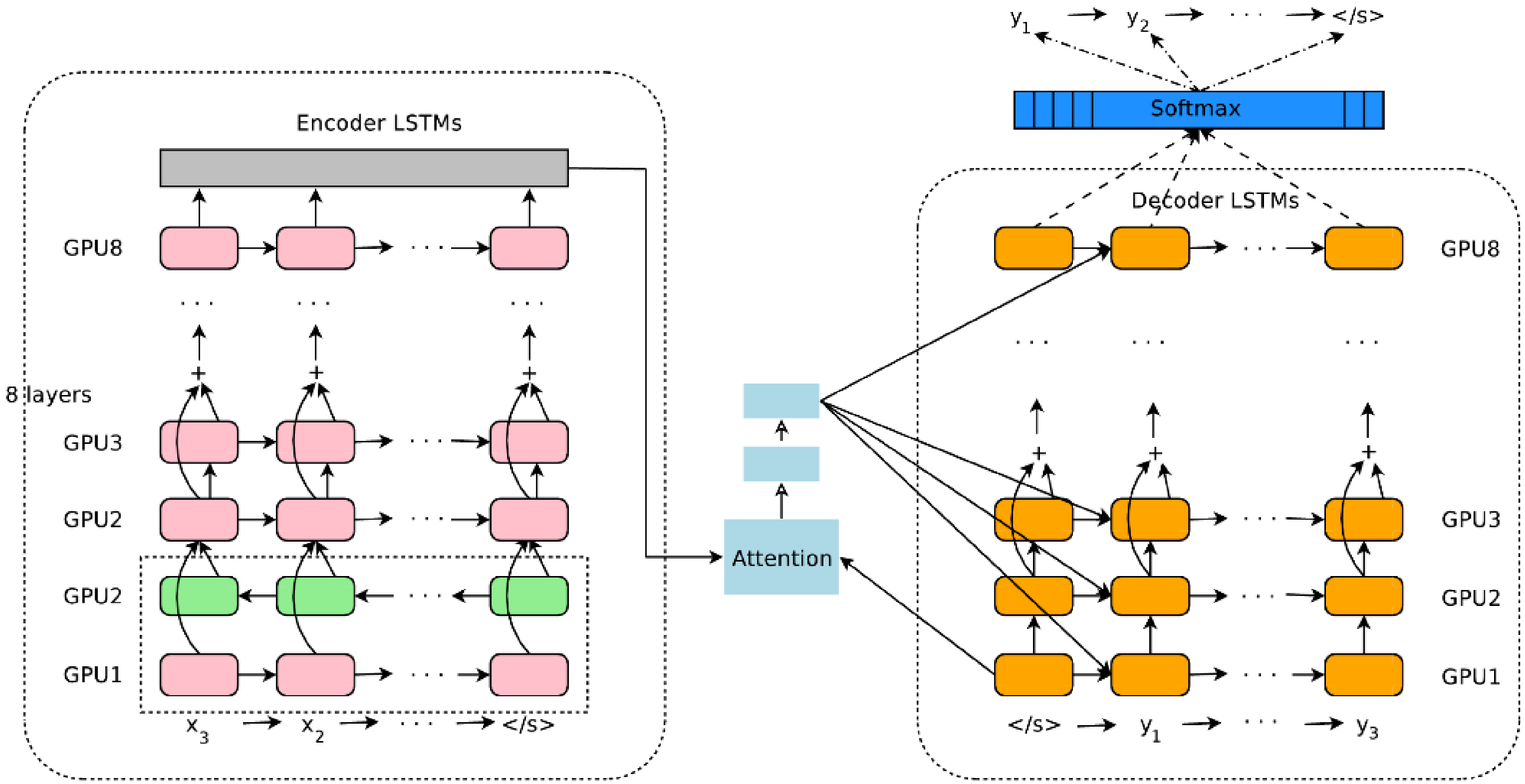
Beam Width = 3

RNN-decoder with RNN-encoder with Attention

$$\text{Decoder vocabulary} = \{\text{Much } (\boldsymbol{w}_1),\ \text{Thanks } (\boldsymbol{w}_2),\ \text{Very } (\boldsymbol{w}_3), <\text{end}> (\boldsymbol{w}_4)\}$$



Example: Google Neural Machine Translation



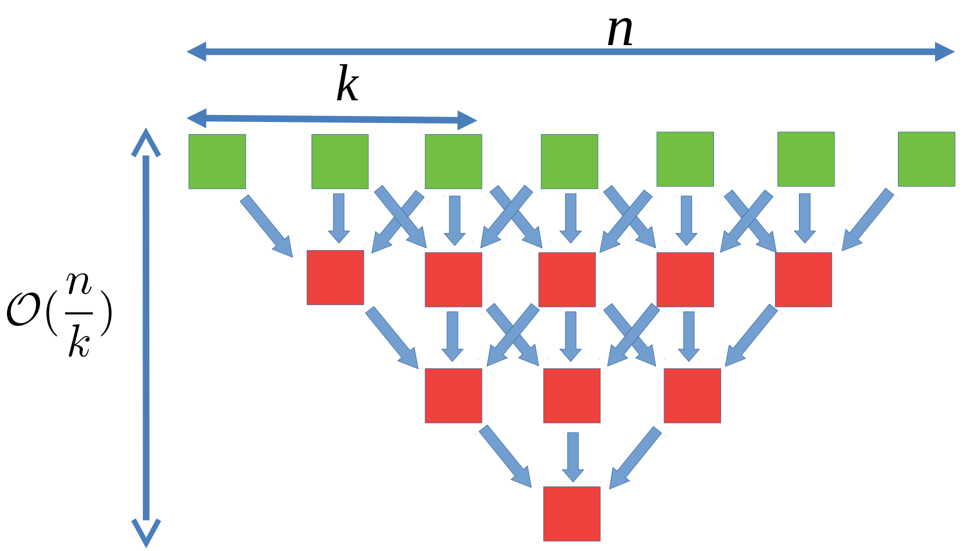
Possible Approaches

- Recurrent networks
  - Apply the NN module in a serial fashion
- Convolutions networks
  - Apply the NN modules in a hierarchical fashion
- Self-attention
  - Direct interaction in the inputs

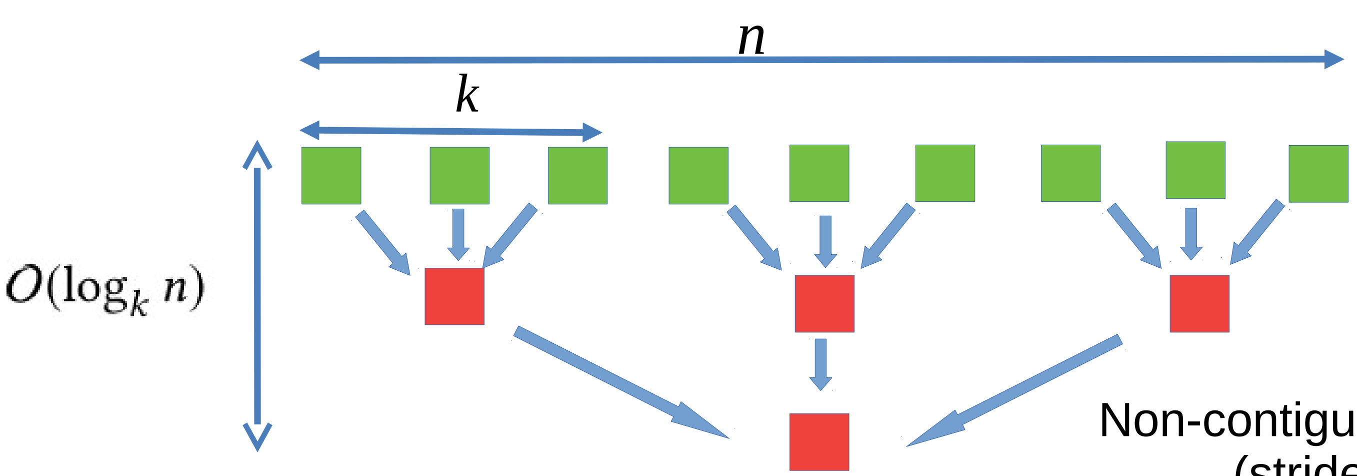
Why Convolution

- Recurrent networks are serial
  - Unable to be parallelized
  - “Distance” between feature vector and different inputs are not constant
- Convolutions networks
  - Can be parallelized (faster)
  - “Distance” between feature vector and different inputs are constant

Distance to feature vector in conv nets

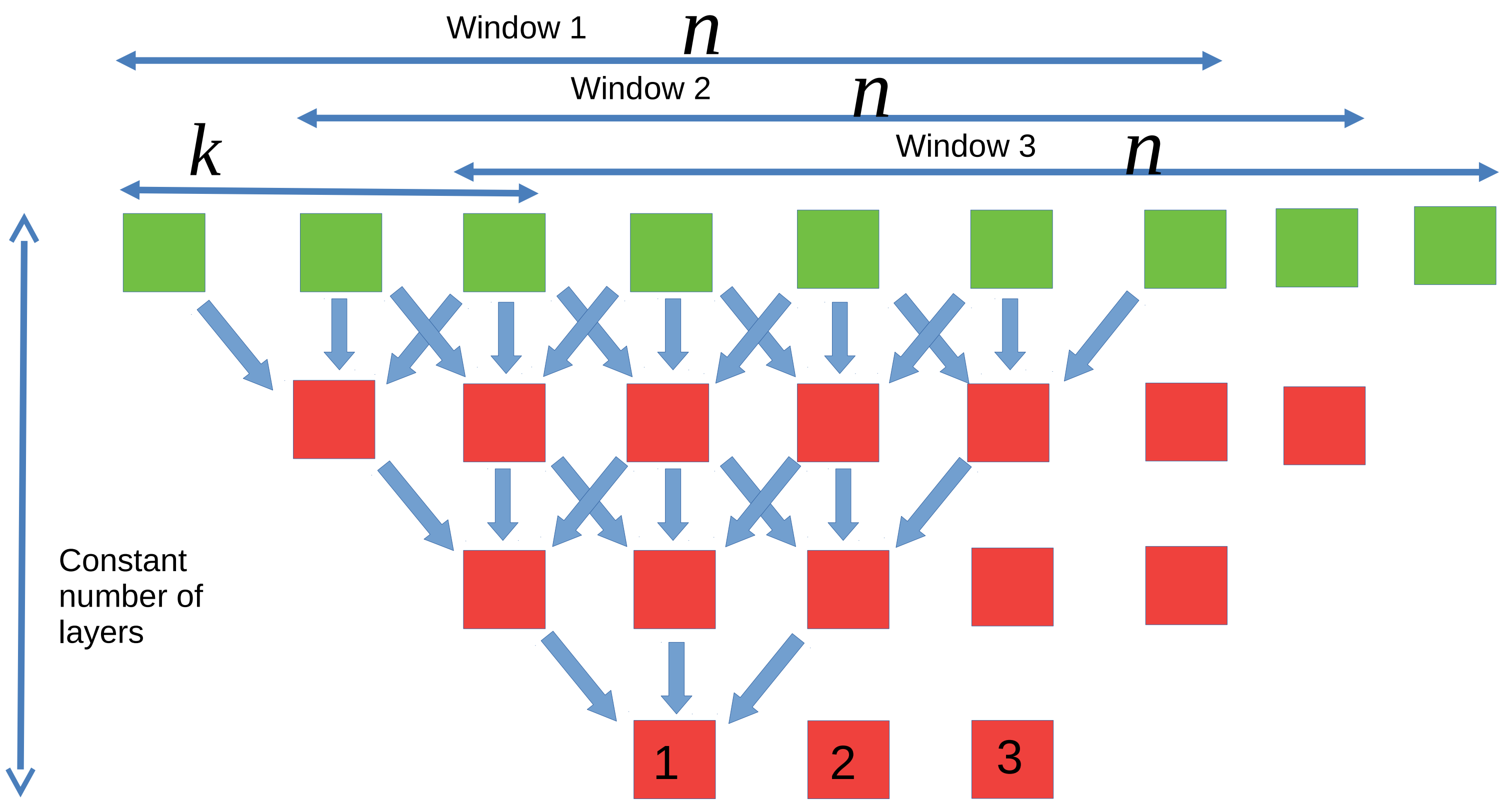


Contiguous convolution
(stride=1)



Non-contiguous convolution
(stride =k)

Context capture with Convolution Networks



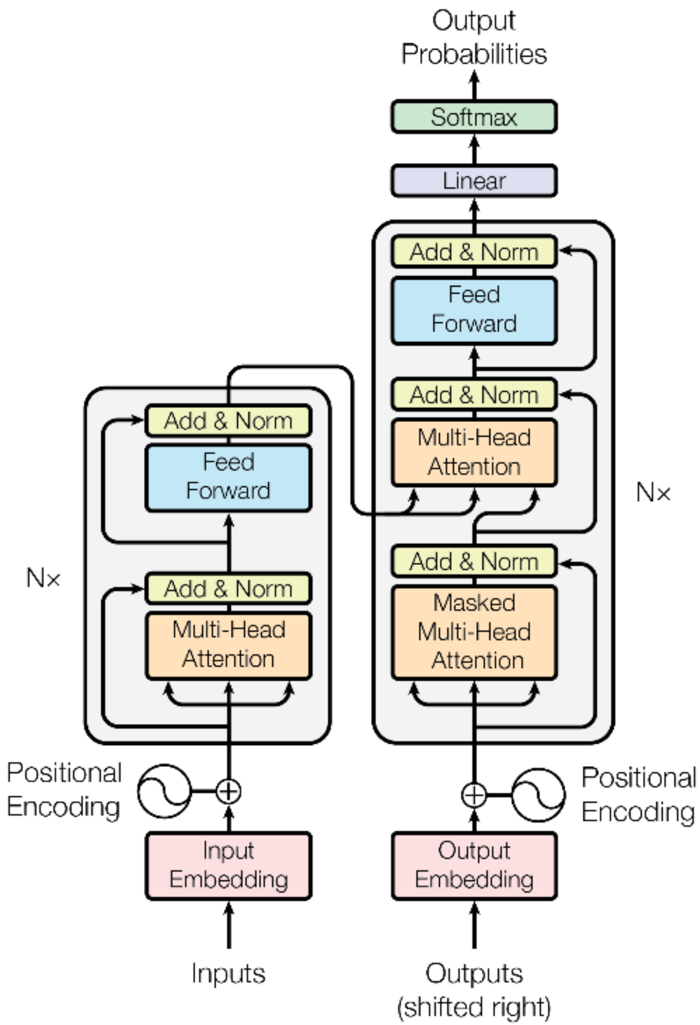
Possible Approaches

- Recurrent networks
  - Apply the NN module in a serial fashion
- Convolutions networks
  - Apply the NN modules in a hierarchical fashion
- Self-attention
  - Direct interaction in the inputs

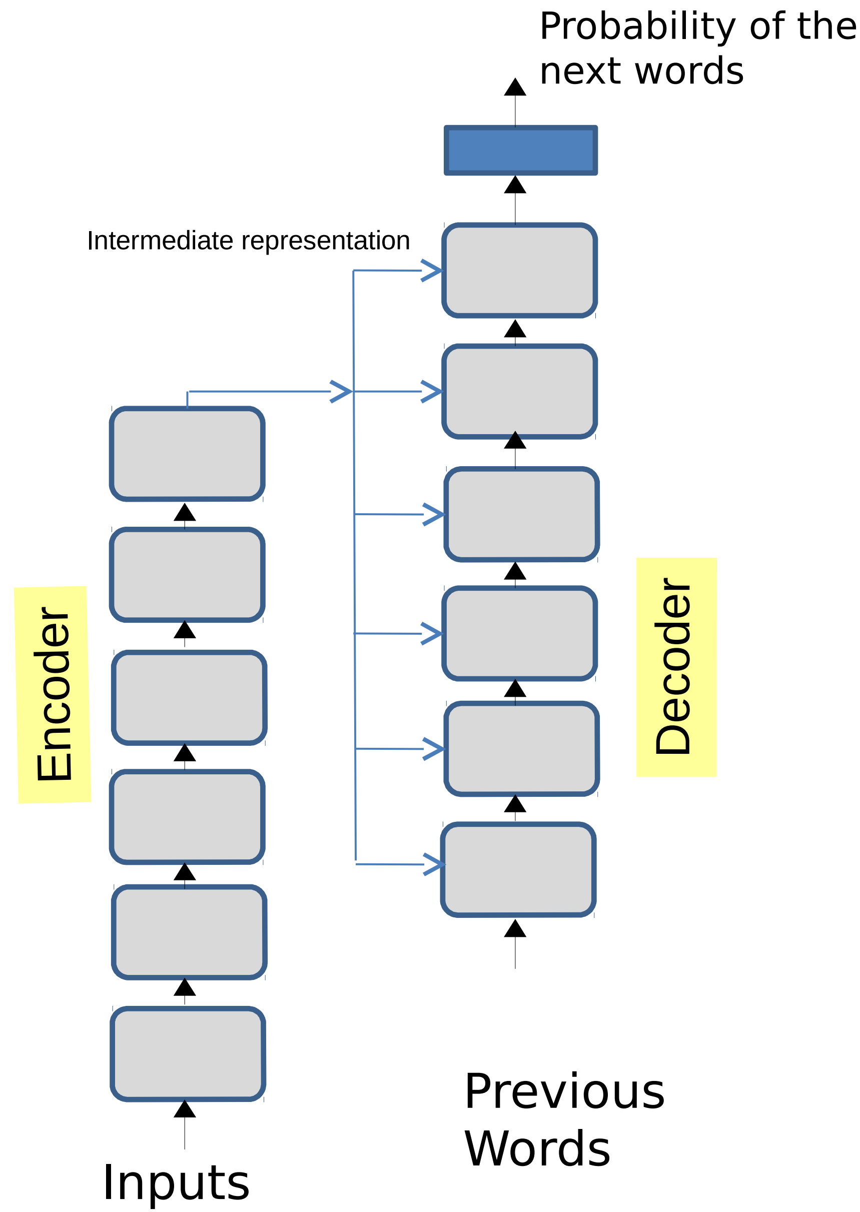
Why Self-attention

- Recurrent networks are serial
  - Unable to be parallelized
  - “Distance” between feature vector and different inputs are not constant
- Self-attention networks
  - Can be parallelized (faster)
  - “Distance” between feature vector and different inputs does not depend on the input length

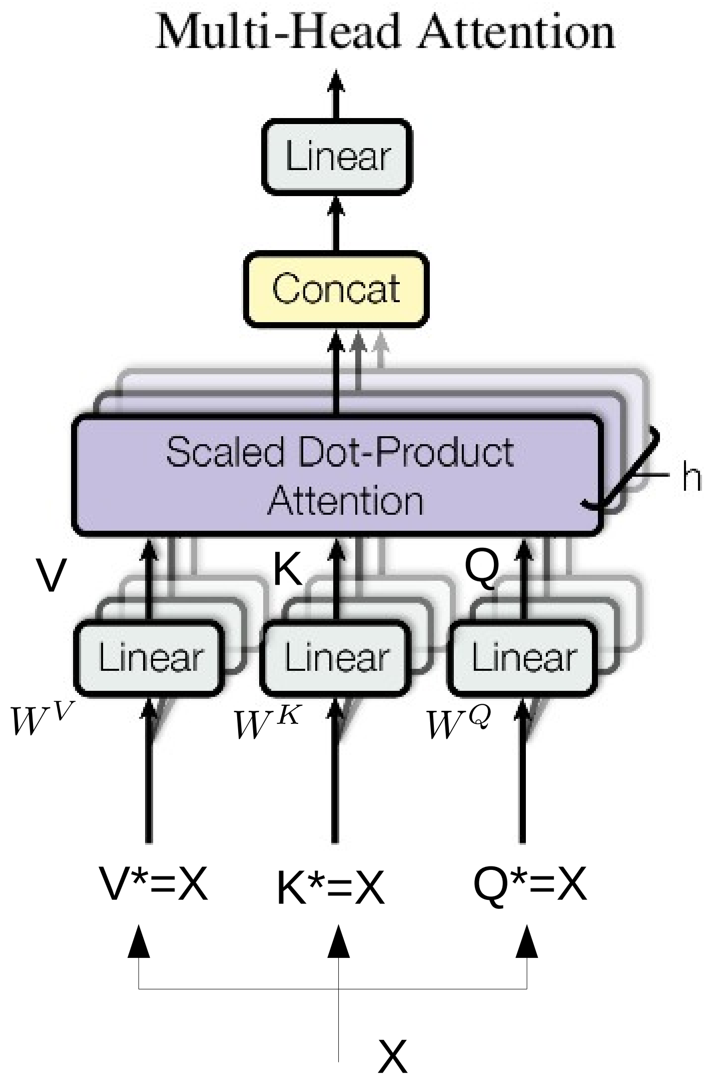
Transformer network with self-attention



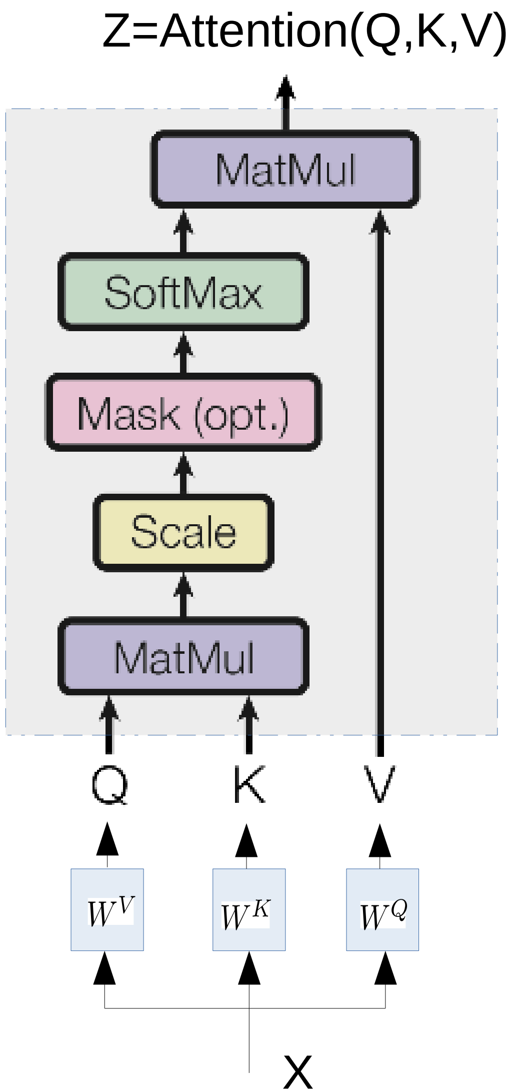
Vasvani et.al, *Attention is all you need*, 2017



Multi-Head Attention



Scaled dot product attention



Input word vectors $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n]^T$

Query $\boldsymbol{Q} = [\boldsymbol{q}_1, \boldsymbol{q}_2, \cdots, \boldsymbol{q}_n]^T$

Keys $\boldsymbol{K} = [\boldsymbol{k}_1, \boldsymbol{k}_2, \cdots, \boldsymbol{k}_n]^T$

Values $\boldsymbol{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_n]^T$

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q$$

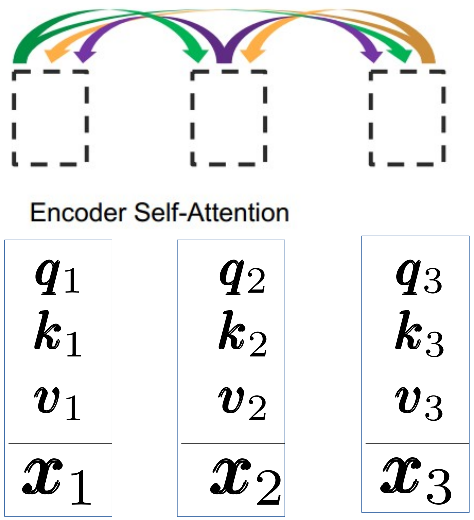
$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K$$

$$\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V$$

$\boldsymbol{W}^Q, \boldsymbol{W}^K, \boldsymbol{W}^V$ Trainable weight vectors

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

Encoder Self-attention

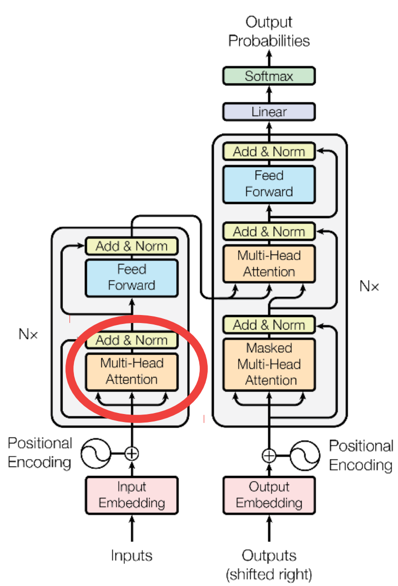


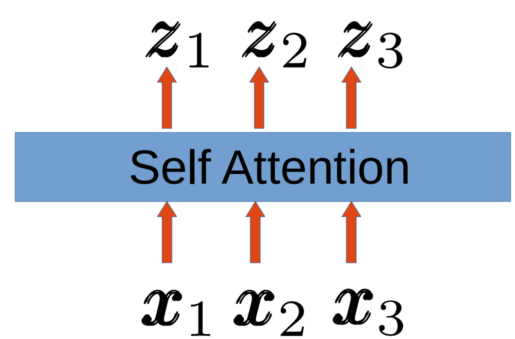
$$\alpha_{1,1} = \frac{\exp(\boldsymbol{q}_1 \boldsymbol{k}_1^T)}{\sum_j \exp(\boldsymbol{q}_1 \boldsymbol{k}_j^T)} \quad \alpha_{1,2} = \frac{\exp(\boldsymbol{q}_1 \boldsymbol{k}_2^T)}{\sum_j \exp(\boldsymbol{q}_1 \boldsymbol{k}_j^T)} \quad \alpha_{1,3} = \frac{\exp(\boldsymbol{q}_1 \boldsymbol{k}_3^T)}{\sum_j \exp(\boldsymbol{q}_1 \boldsymbol{k}_j^T)}$$

$$\alpha_{2,1} = \frac{\exp(\boldsymbol{q}_2 \boldsymbol{k}_1^T)}{\sum_j \exp(\boldsymbol{q}_2 \boldsymbol{k}_j^T)} \quad \alpha_{2,2} = \frac{\exp(\boldsymbol{q}_2 \boldsymbol{k}_2^T)}{\sum_j \exp(\boldsymbol{q}_2 \boldsymbol{k}_j^T)} \quad \alpha_{2,3} = \frac{\exp(\boldsymbol{q}_2 \boldsymbol{k}_3^T)}{\sum_j \exp(\boldsymbol{q}_2 \boldsymbol{k}_j^T)}$$

$$\alpha_{3,1} = \frac{\exp(\boldsymbol{q}_3 \boldsymbol{k}_1^T)}{\sum_j \exp(\boldsymbol{q}_3 \boldsymbol{k}_j^T)} \quad \alpha_{3,2} = \frac{\exp(\boldsymbol{q}_3 \boldsymbol{k}_2^T)}{\sum_j \exp(\boldsymbol{q}_3 \boldsymbol{k}_j^T)} \quad \alpha_{3,3} = \frac{\exp(\boldsymbol{q}_3 \boldsymbol{k}_3^T)}{\sum_j \exp(\boldsymbol{q}_3 \boldsymbol{k}_j^T)}$$

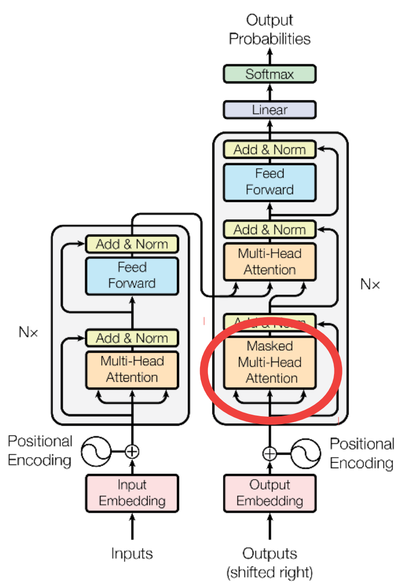
$$\boldsymbol{z}_1 = \alpha_{1,1}\boldsymbol{v}_1 + \alpha_{1,2}\boldsymbol{v}_2 + \alpha_{1,3}\boldsymbol{v}_3$$
$$\boldsymbol{z}_2 = \alpha_{2,1}\boldsymbol{v}_1 + \alpha_{2,2}\boldsymbol{v}_2 + \alpha_{2,3}\boldsymbol{v}_3$$
$$\boldsymbol{z}_3 = \alpha_{3,1}\boldsymbol{v}_1 + \alpha_{3,2}\boldsymbol{v}_2 + \alpha_{3,3}\boldsymbol{v}_3$$

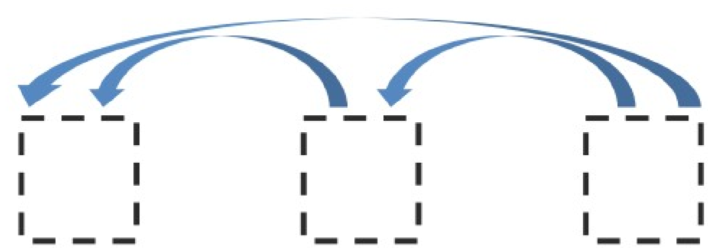




Decoder Self-attention

- Almost same as encoder self attention
- But only leftward positions are considered.
  - Rightward positions are masked



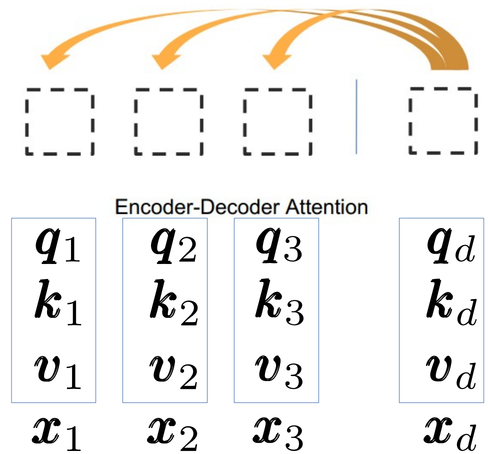


MaskedDecoder Self-Attention

Encoder-decoder attention

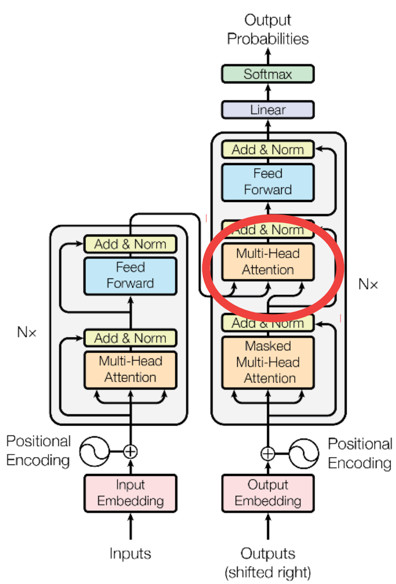
Encoder states

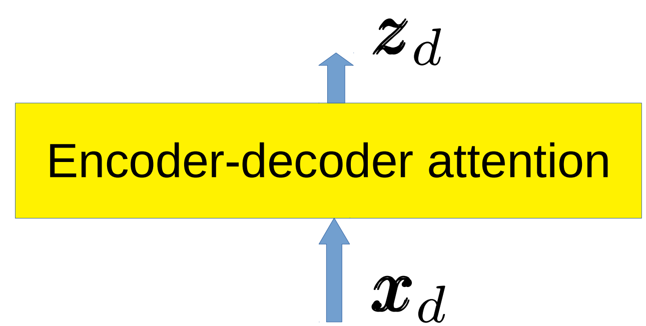
Decoder state



$$\alpha_{1,1} = \frac{\exp(\boldsymbol{q}_d \boldsymbol{k}_1^T)}{\sum_j \exp(\boldsymbol{q}_d \boldsymbol{k}_j^T)} \qquad \alpha_{1,2} = \frac{\exp(\boldsymbol{q}_d \boldsymbol{k}_2^T)}{\sum_j \exp(\boldsymbol{q}_d \boldsymbol{k}_j^T)} \qquad \alpha_{1,3} = \frac{\exp(\boldsymbol{q}_d \boldsymbol{k}_3^T)}{\sum_j \exp(\boldsymbol{q}_d \boldsymbol{k}_j^T)}$$

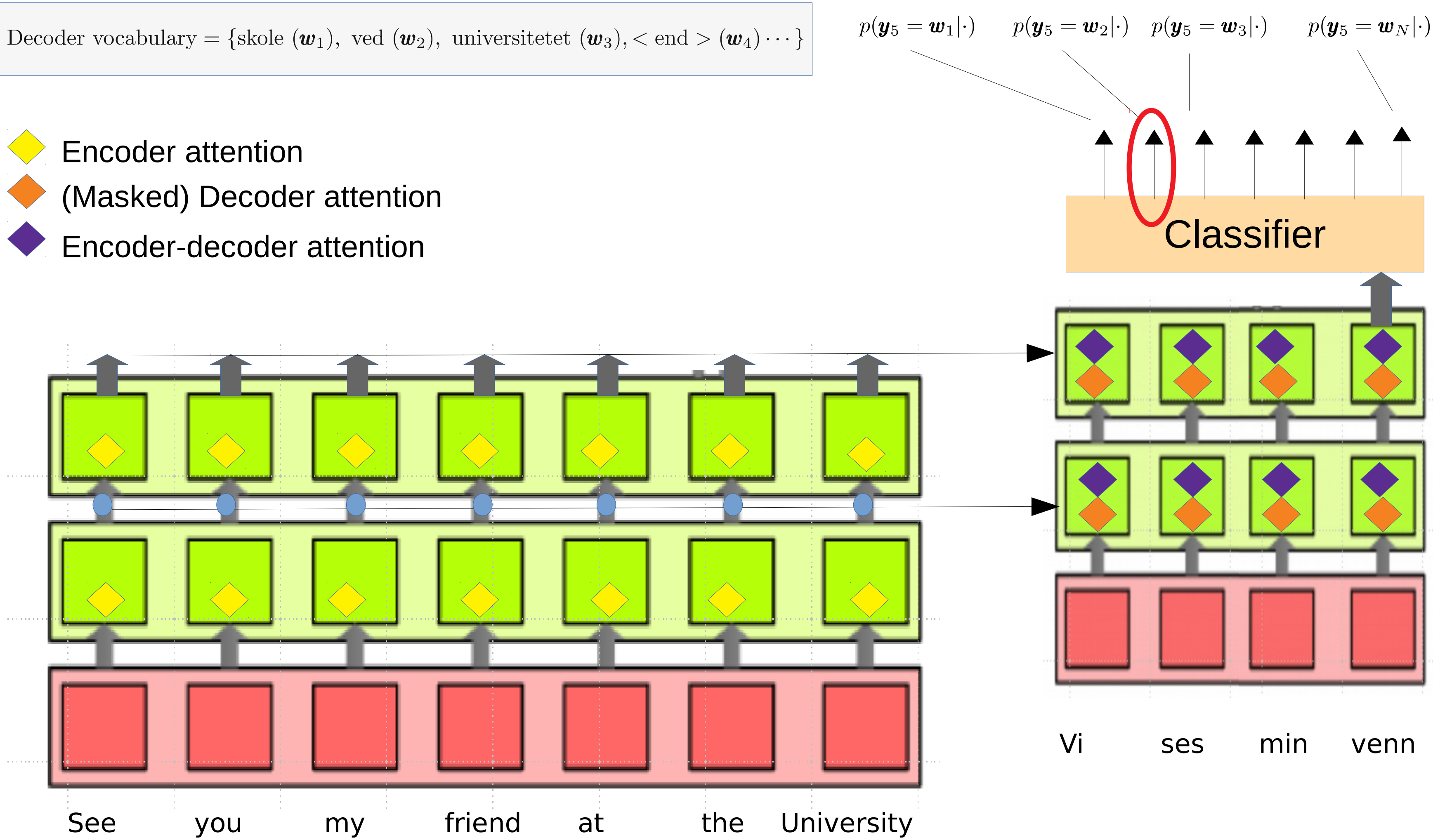
$$\boldsymbol{z}_d = \alpha_{1,1}\boldsymbol{v}_1 + \alpha_{1,2}\boldsymbol{v}_2 + \alpha_{1,3}\boldsymbol{v}_3$$





Overall Operation

$$\text{Decoder vocabulary} = \{\text{skole } (\boldsymbol{w}_1),\ \text{ved } (\boldsymbol{w}_2),\ \text{universitetet } (\boldsymbol{w}_3), <\text{end}> (\boldsymbol{w}_4) \cdots\}$$



Comparison of Seq2Seq Methods

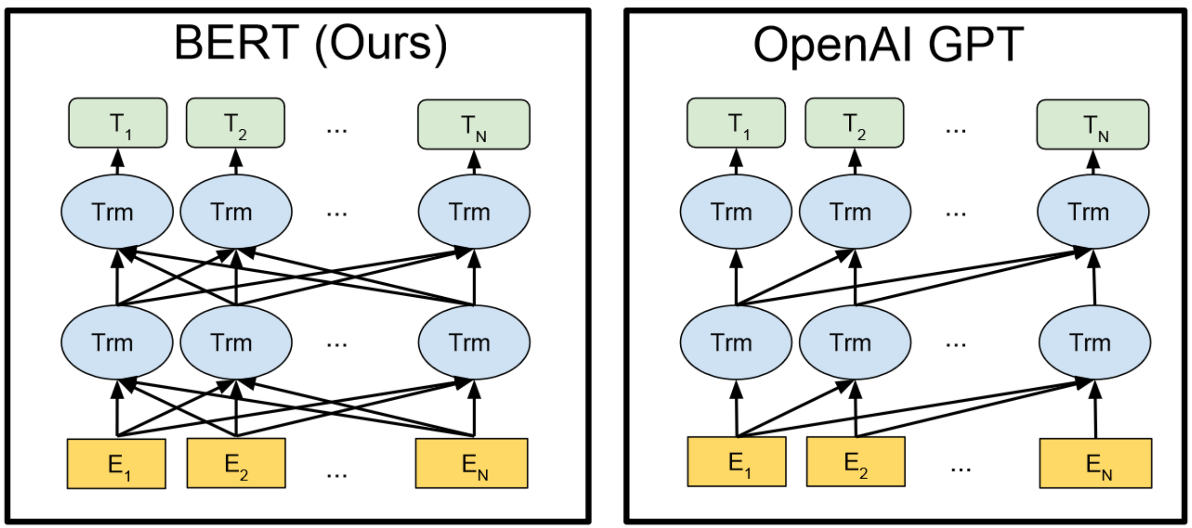
Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

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

Famous Transformer Systems

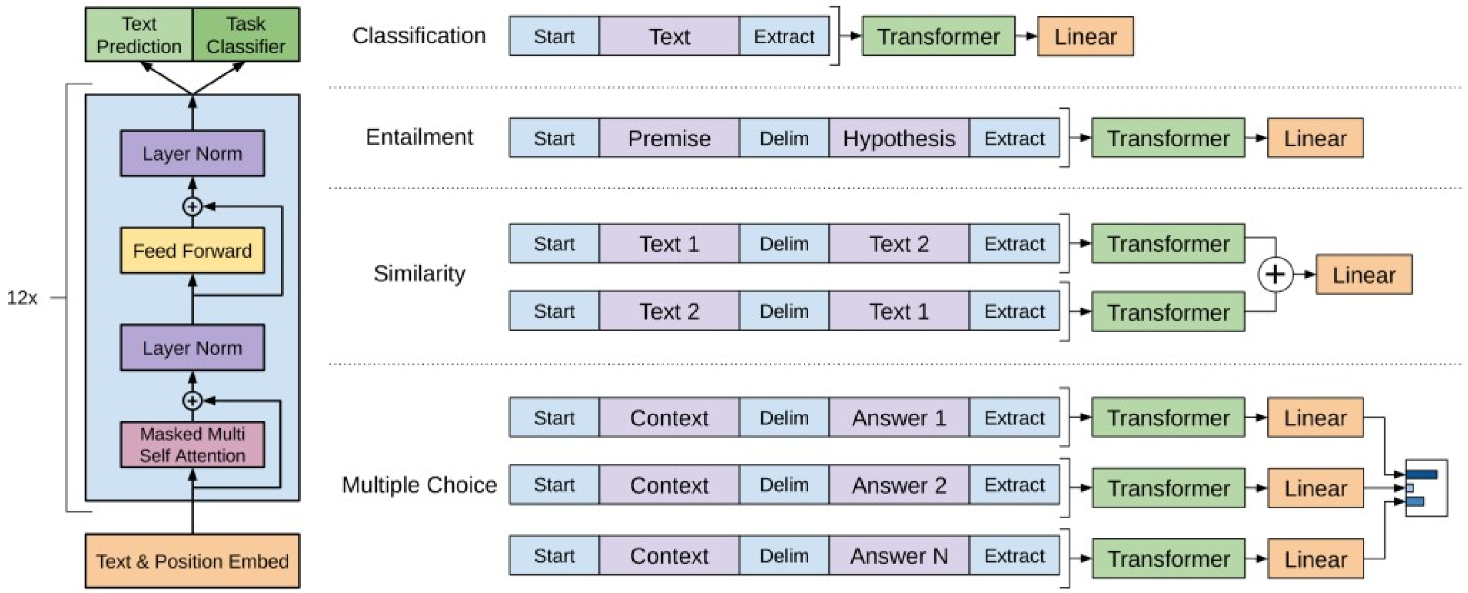
- BERT (Bidirectional Encoder Representations from Transformers)
  - Based on Transformer Encoder block
  - Self-supervised learning approach and pretrained with
    - Masked Language model
    - Next sentence prediction
  - Several down-stream tasks (classification, QA etc.)

- GPT (Generalized Pre-Training, several versions GPT-1,GPT-2, GPT-3)
  - Based on Transformer Decoder block (Masked Self attention)
  - Self-supervised learning and pre-trained with
    - Next word prediction (given the previous prediction) i.e. autoregression
  - Several down-stream tasks (classification, similarity, QA etc.)



GPT – Downstream Tasks

- GPT-1 Downstream tasks
  - Text classification
  - Entailment (Hypothesis follows from the premise?)
  - Similarity (Are two sentences similar?)
  - Multiple choice (Given a text and question select the correct answer)



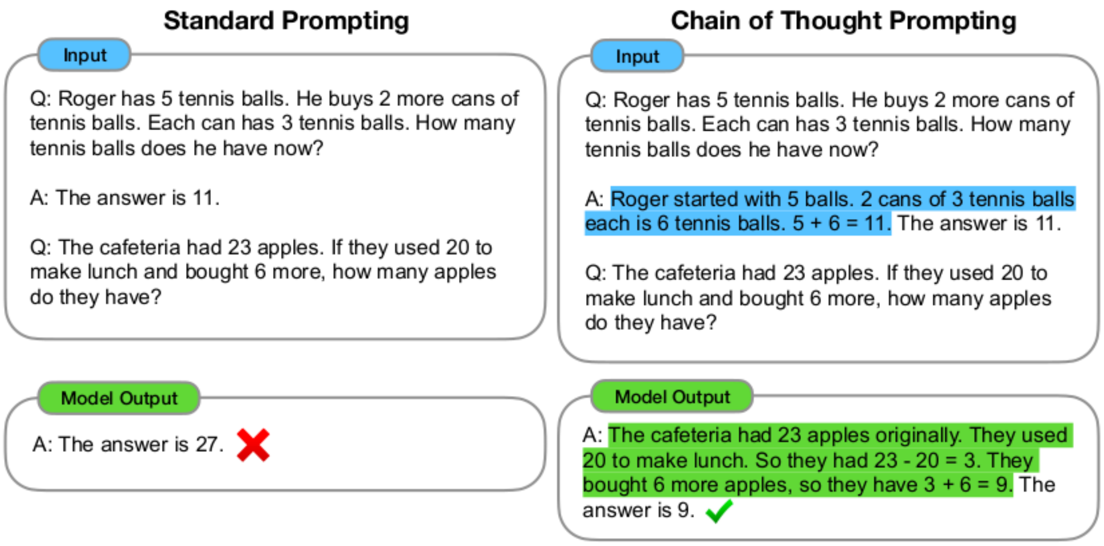
New Trends in Language Models

- Scale up
- Autoregressive training (i.e. Decoder model of transformer)
- Drop fine-tuning
- Use as a few shot learner (eg: 5 shot learning) on many tasks

Model	#Parameters (Millions)	By
GPT	110	OpenAI
BERT-Large	340	Google
GPT-2	1500	OpenAI
Chinchilla	70000	Deepmind
GPT-3	175000	OpenAI
Gopher	280000	Deepmind
Megatron-Turing NLG	530000	Microsoft & Nvidia
PaLM	540000	Google

Quantitative Reasoning with Large Language models

- Standard prompting
  - Training:
    - Concatenate question and answer
    - Train to predict next word
  - Testing:
    - Input the question
    - Generate the answer in an auto-regressive manner
- Chain of thought prompting
  - Training:
    - Concatenate question and detailed answer
    - Train to predict next word
  - Testing (as in standard prompting)



Quantitative Reasoning with Large Language models (ctd.)

- Minerva from Google (Based on PaLM)

Question: Assume that the variance of the first n natural numbers is 10, and the variance of the first m even natural numbers is 16. Compute $m+n$.

Model output: Let σ^2 be the variance of the first m even natural numbers, and τ^2 be the variance of the first n natural numbers. Then, we have $\tau^2 = \frac{n^2-1}{12}$ and $\sigma^2 = \frac{m^2-1}{3}$. We solve the system of equations:

$$\frac{n^2-1}{12} = 10$$
$$\frac{m^2-1}{3} = 16.$$

This translates into the system

$$n^2 = 121$$
$$m^2 = 48 + 1 = 49.$$

Therefore, $n = \boxed{11}$ and $m = \boxed{7}$, so $n + m = \boxed{18}$.

Question: For every a, b, $b \neq a$ prove that

$$\frac{a^2+b^2}{2} > \left(\frac{a+b}{2}\right)^2.$$

Model output:

$$\frac{a^2+b^2}{2} > \left(\frac{a+b}{2}\right)^2$$
$$\iff \frac{a^2+b^2}{2} > \frac{a^2+b^2+2ab}{4}$$
$$\iff a^2+b^2 > \frac{a^2+b^2+2ab}{2}$$
$$\iff 2a^2+2b^2 > a^2+b^2+2ab$$
$$\iff a^2+b^2 > 2ab$$
$$\iff a^2+b^2-2ab > 0$$
$$\iff (a-b)^2 > 0$$

which is true, because the square of a real number is positive.

Reinforcement Learning

Reinforcement Learning

- Machine Translation/Summarization
- Dialog Systems
-
-

Reinforcement Learning

- Machine Translation/Summarization
- Dialog Systems
-
-

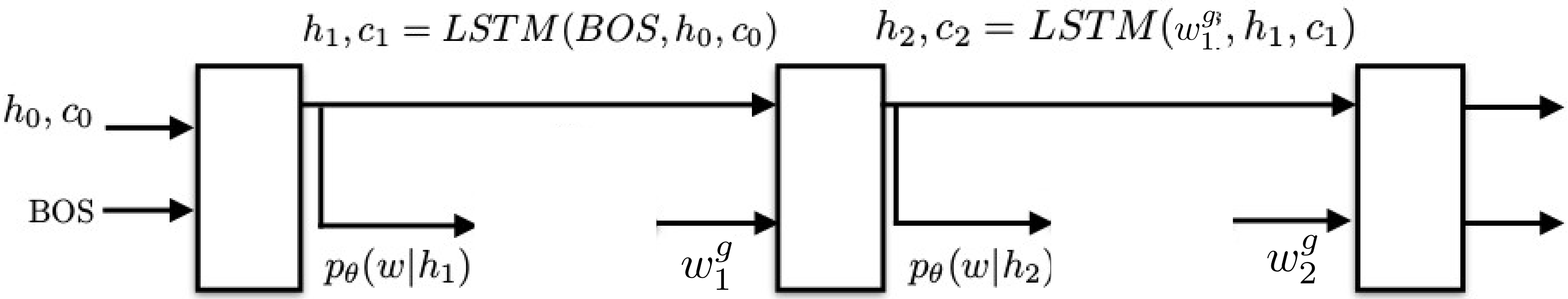
Why Reinforcement Learning

- Exposure bias
  - In training, ground truths are used, (i.e. teacher forcing) and the system develops a bias towards this.
  - In testing, generated word in the previous step is used to generate the next word.
  - Test results are poorer because of the “bias” developed in training.
  - We want a similar approach in training (i.e. use generated words instead of teacher forcing)
  - Use generated words in training needs sampling : Non differentiable
- Maximum Likelihood criterion is not directly relevant to evaluation metrics
  - BLEU (Machine translation)
  - ROUGE (Summarization)
  - Use BLEU/ROUGE in training:
  - Need Non differentiable loss functions

Sequence Generation as Reinforcement Learning

- **Agent:** The Recurrent Net
- **State:** Hidden states, Attention weights etc.
- **Action:** Next Word
- **Policy:** Generate the next word (*action*) given the current hidden states and attention weights (*state*)
- **Reward:** Score computed using the evaluation metric (eg: BLEU)

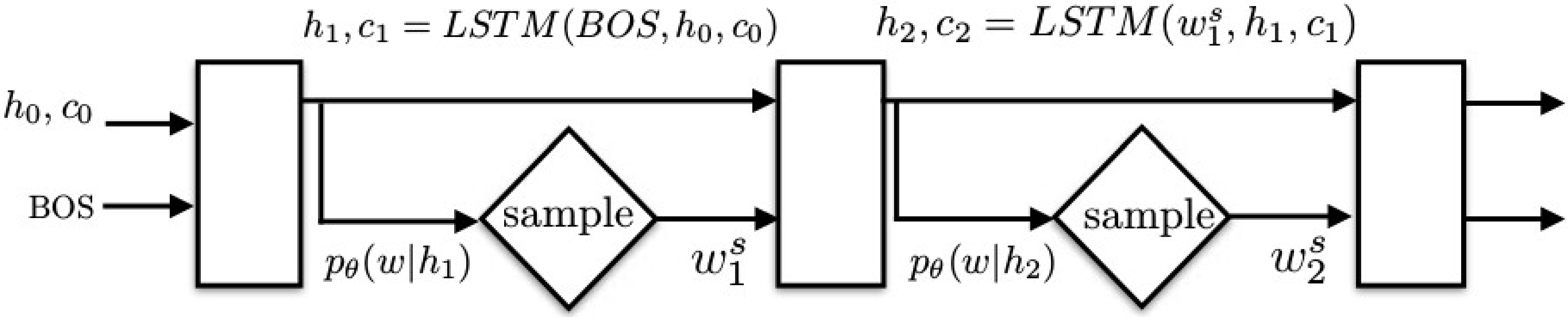
Maximum Likelihood Training (Revisit)



$$\text{Log Likelihood} = \sum_{t=1}^{T} \log p_\theta(w_t^g | h_t)$$

Minimize the negative log likelihood

Reinforcement Learning Formulation



$$\text{Reward} \ = r(w^s) = r(w_1^s, w_2^s, \cdots, w_T^s)$$

Minimize the expected negative reward, $L(\theta) = -\mathbb{E}_{w^s \sim p_\theta}\left[r(w^s)\right]$
using REINFORCE algorithm

Reinforcement Learning Details

- Expected reward $$L(\theta) = -\sum_{w} p_\theta(w) r(w)$$
- We need the gradient $$\nabla_\theta L(\theta) = -\sum_{w} r(w) \nabla_\theta p_\theta(w)$$
- Need to write this as an expectation, so that we can evaluate it using samples. Use the log derivative trick: $$\nabla_\theta L(\theta) = -\sum_{w} r(w) p_\theta(w) \nabla_\theta \log p_\theta(w)$$
- This is an expectation $$\nabla_\theta L(\theta) = -\mathbb{E}_{w^s \sim p_\theta} \left[r(w^s) \nabla_\theta \log p_\theta(w^s) \right]$$
- Approximate this with sample mean

$$\nabla_\theta L(\theta) \approx -\frac{1}{N} \sum_{s=1}^{N} r(w^s) \nabla_\theta \log p_\theta(w^s)$$

- In practice we use only one sample

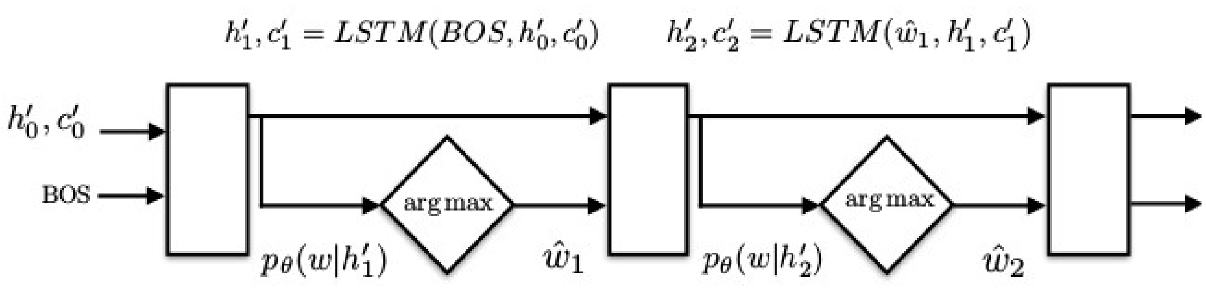
$$\nabla_\theta L(\theta) \approx -r(w^s) \nabla_\theta \log p_\theta(w^s)$$

Reinforcement Learning Details

- Gradient $\nabla_\theta L(\theta) \approx -r(w^s)\nabla_\theta \log p_\theta(w^s)$
- This estimation has high variance. Use a baseline to combat this problem.

$$\nabla_\theta L(\theta) \approx -(r(w^s) - b)\nabla_\theta \log p_\theta(w^s)$$

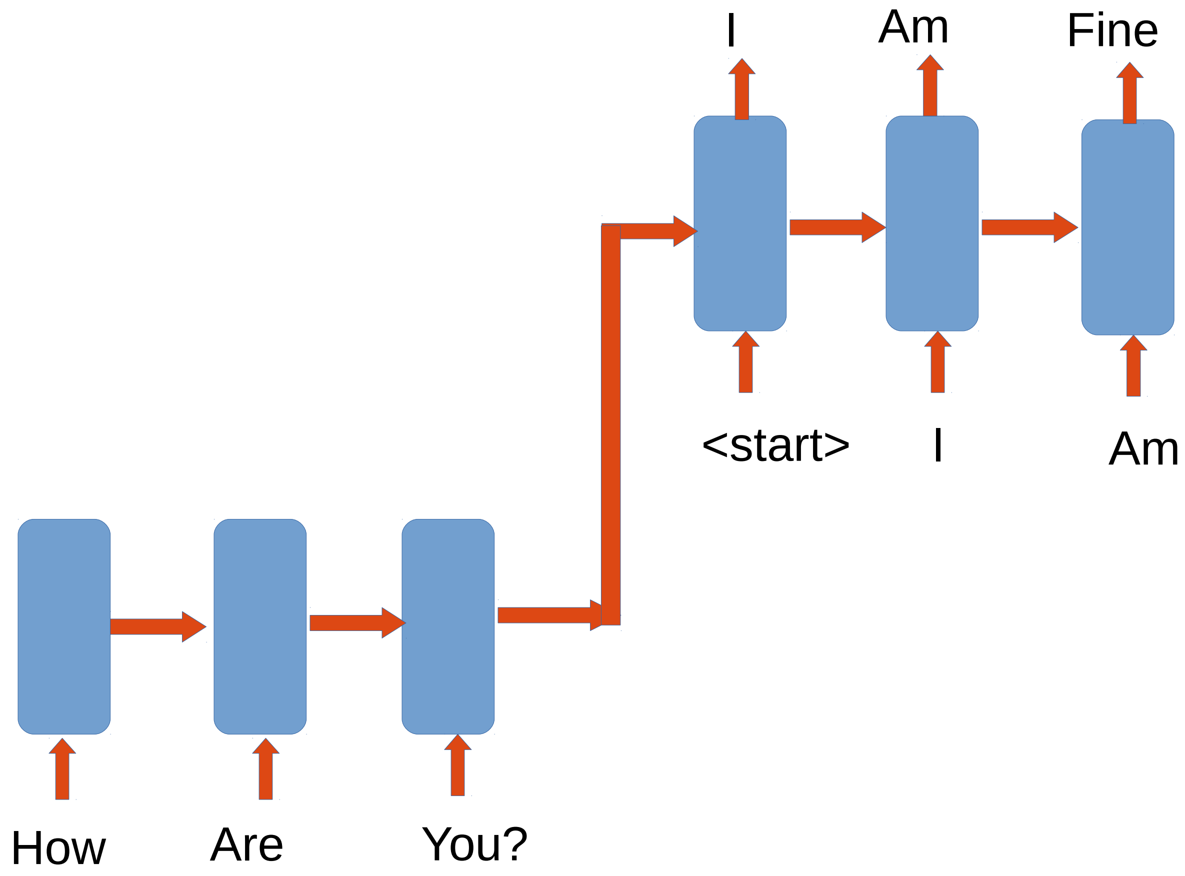
- Baseline can be anything independent of w^s
- It can for example be estimated as the reward for word sequence generated using argmax at each cell. $b = r(\hat{w}_1, \hat{w}_2, \hat{w}_3, \cdots, \hat{w}_T)$



Reinforcement Learning

- Machine Translation/Summarization
- Dialog Systems
-
-

Maximum Likelihood Dialog Systems



Why Reinforcement Learning

- Maximum Likelihood criterion is not directly relevant to successful dialogs
  - Dull responses (“I don’t know”)
  - Repetitive responses
- Need to integrate developer defined rewards relevant to longer term goals of the dialog

Dialog Generation as Reinforcement Learning

- **Agent:** The Recurrent Net
- **State:** Previous dialog turns
- **Action:** Next dialog utterance
- **Policy:** Generate the next dialog utterance (*action*) given the previous dialog turns (*state*)
- **Reward:** Score computed based on relevant factors such as ease of answering, information flow, semantic coherence etc.

Training Procedure

- From the viewpoint of a given agent, the procedure is similar to that of sequence generation
  - REINFORCE algorithm
- Appropriate rewards must be calculated based on current and previous dialog turns.
- Can be initialized with maximum likelihood trained models.