Neural Machine Translation Internals

By Mohit Kumar

Outline

- **Background**
  - Sentiment Classification
  - Machine Translation: task overview
- **Basic NMT**
  - An encoder-decoder architecture
- **Advanced NMT**
  - The attention aspect
  - The vocabulary aspect
  - The data aspect
- **State-of-art NMT System**
  - GNMT (Google NMT System)

Sentiment Classification

```
from keras.datasets import imdb
train, test= imdb.load_data(path=utils.get_data_path()+'imdb.npz')
train_set_x, train_set_y = train
valid_portion=.1
n_samples = len(train_set_x)
print("n_samples:",n_samples)
sidx = np.random.permutation(n_samples)
n_train = int(np.round(n_samples * (1. - valid_portion)))
valid_set_x = [train_set_x[s] for s in sidx[n_train:]]
valid_set_y = [train_set_y[s] for s in sidx[n_train:]]
train_set_x = [train_set_x[s] for s in sidx[:n_train]]
train_set_y = [train_set_y[s] for s in sidx[:n_train]]
train = (train_set_x, train_set_y)
test = (valid_set_x, valid_set_y)
# IMDB Dataset loading

trainX, trainY = train
testX, testY = test

X = np.concatenate((trainX, testX), axis=0)
Y = np.concatenate((trainY, testY), axis=0)
```

- Imdb reviews one hot encoded in binary form.
- Labels also in binary form.

```
[1, 14, 20, 839, 1674, 8, 61, 523, 308, 3338, 66, 697, 89, 8, 79, 8, 21729, 3385, 940, 14, 389, 431, 121, 113, 9, 2395, 143, 4, 330, 5, 483,
7, 4196, 75, 1974, 39, 58, 8, 58, 19, 4, 339, 7, 2728, 5, 1020, 907, 40, 308, 15, 60, 392, 2373, 40, 4196, 28, 6, 483, 13, 43, 426, 570, 61,
1674, 60, 151, 4, 22, 47, 6, 654, 130, 14, 9, 87, 103, 3086, 7, 108, 13, 219, 143, 61, 113, 4196, 66, 2842, 72, 1685, 14, 9, 103, 4, 1558, 15
679, 4, 333, 431, 15, 66, 679, 72, 6805, 180]
```

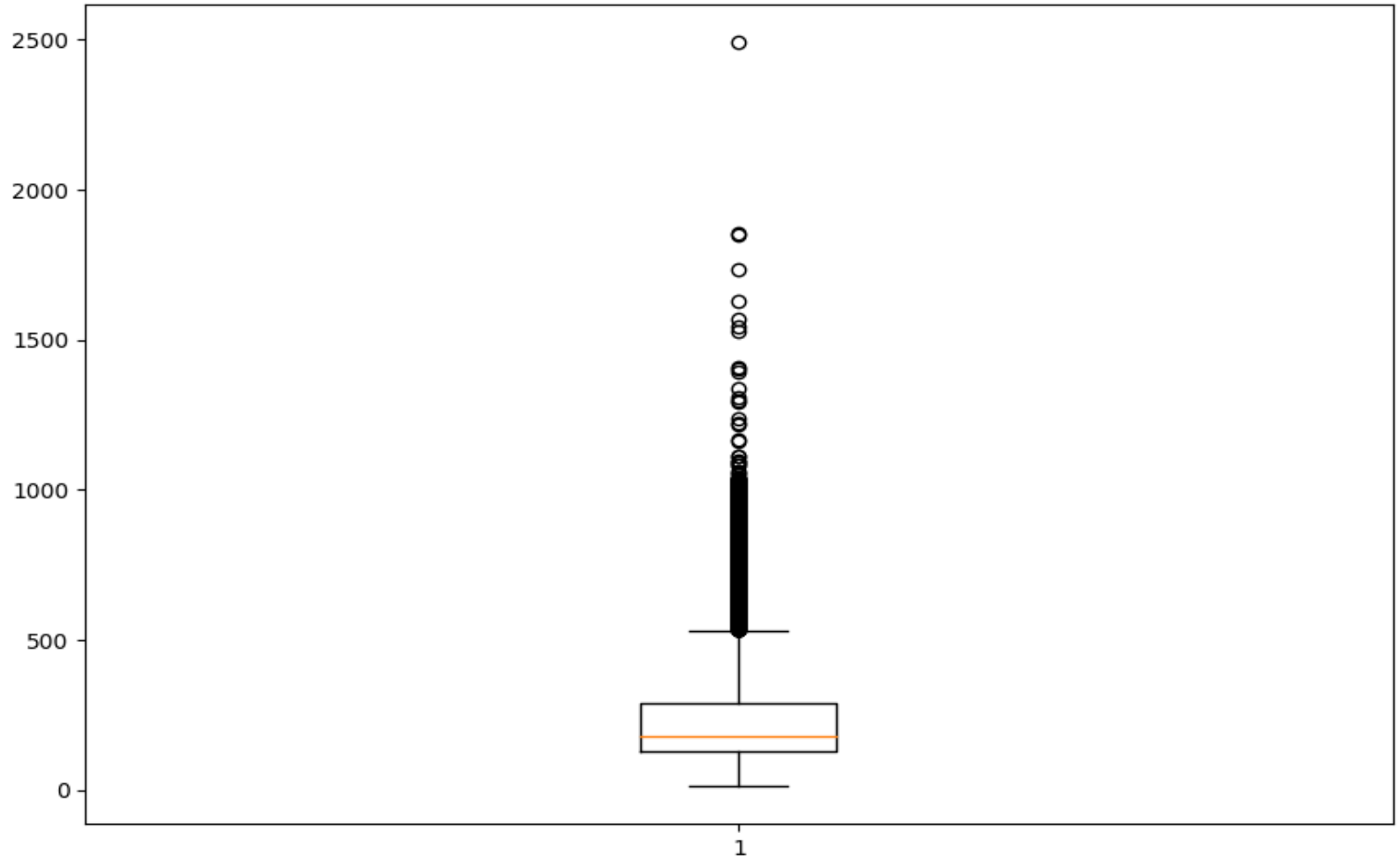
Sentiment Classification

```
def textify(input):
    word_to_id=imdb.get_word_index(path=utils.get_data_path()+'imdb_word_index.json')
    word_to_id = {k:(v+INDEX_FROM) for k,v in word_to_id.items()}
    word_to_id["<PAD>"] = 0
    word_to_id["<START>"] = 1
    word_to_id["<UNK>"] = 2

    id_to_word = {value:key for key,value in word_to_id.items()}
    text=' '.join(id_to_word[id] for id in input )
    return text

from enum import Enum
class SentEnum(Enum):
    NEGATIVE = 0
    POSITIVE = 1

def sentiment(pred):
    pred=np.reshape(pred,[1,2])
    ans=[np.where(r == 1)[0][0] for r in pred]
    return SentEnum(ans[0]).name
```

1. Used the word index to textify the binary.

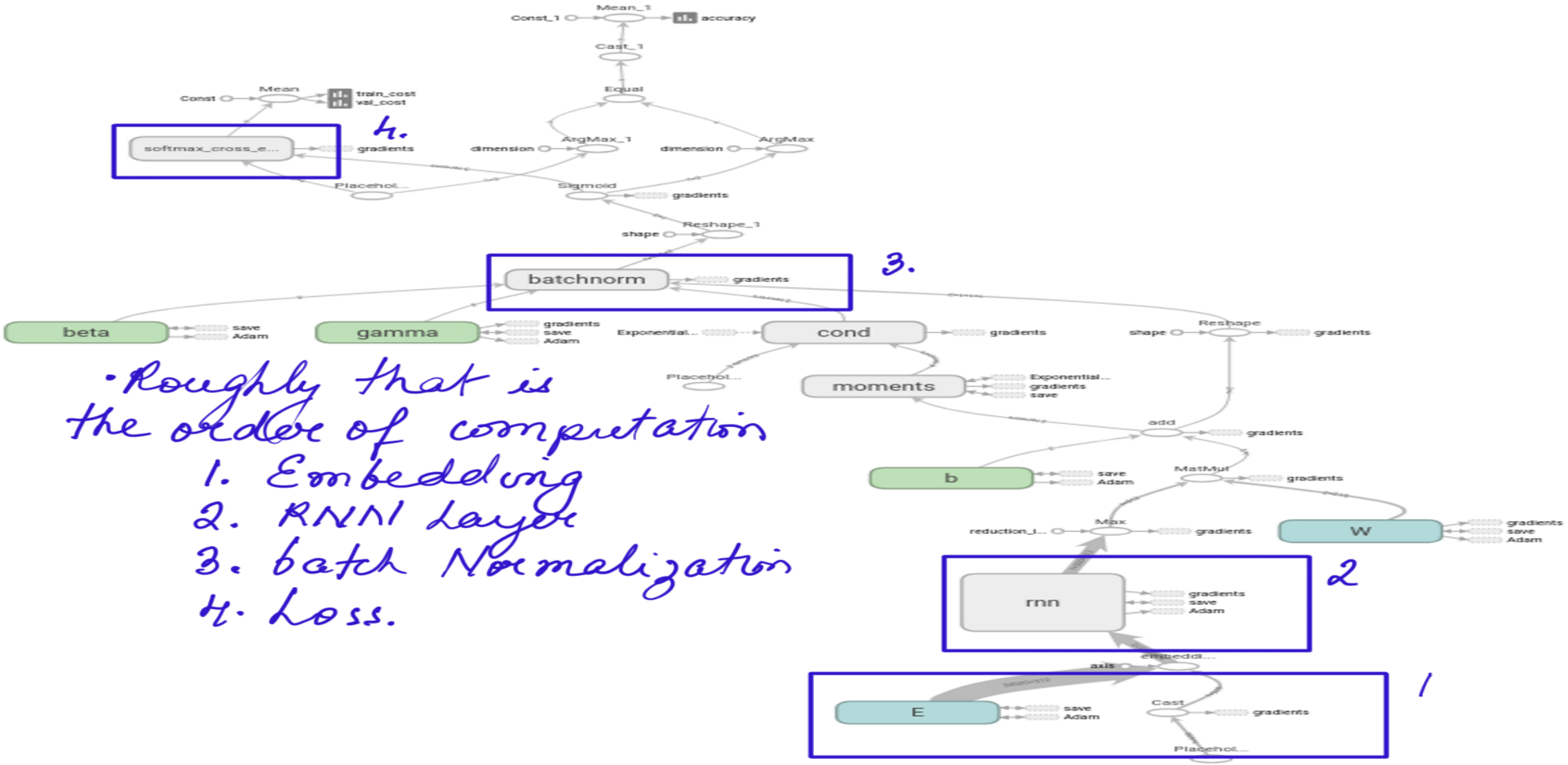
```
text: <START> moonstruck is a lovely little film directed by superb story teller norman jewison in the heat of the night fiddler on the roof
the hurricane the film is great on many levels it shows a good slice of italian culture has a touching romance and best of all is a hilarious
 comedy br br one thing i liked most about the film was the relative unconventional looks of the actors nicolas cage looks positively odd for
 most of the film and cher well cher always looks a little odd br br overall it's a fun film and easy to recommend br br 7 4 out of 10 <PAD>
<PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PA

label: POSITIVE
```

Sentiment Classification



- Box and whisker plot of the reviews by word length.

Sentiment Classification



- Roughly that is the order of computation
  1. Embedding
  2. RNN Layer
  3. batch Normalization
  4. Loss.

Sentiment Classification

```python
def embedding_layer(input, weight_shape):
    weight_init = tf.random_normal_initializer(stddev=(1.0/weight_shape[0])**0.5)
    E = tf.get_variable("E", weight_shape,
                        initializer=weight_init)
    incoming = tf.cast(input, tf.int32)
    embeddings = tf.nn.embedding_lookup(E, incoming)
    return embeddings
```

```python
def lstm(input, hidden_dim, keep_prob, phase_train):
    lstm = tf.nn.rnn_cell.BasicLSTMCell(hidden_dim)
    dropout_lstm = tf.nn.rnn_cell.DropoutWrapper(lstm, input_keep_prob=keep_prob, output_keep_prob=keep_prob)
    # stacked_lstm = tf.nn.rnn_cell.MultiRNNCell([dropout_lstm] * 2, state_is_tuple=True)
    lstm_outputs, state = tf.nn.dynamic_rnn(dropout_lstm, input, dtype=tf.float32)
    return tf.reduce_max(lstm_outputs, reduction_indices=[1])
```

```python
def inference(input, phase_train):
    embedding = embedding_layer(input, [30000, 512])
    lstm_output = lstm(embedding, 512, 0.5, phase_train)
    output = layer(lstm_output, [512, 2], [2], phase_train)
    return output
```

- The word embeddings are learnt jointly with the sentiment analysis.

Sentiment Classification

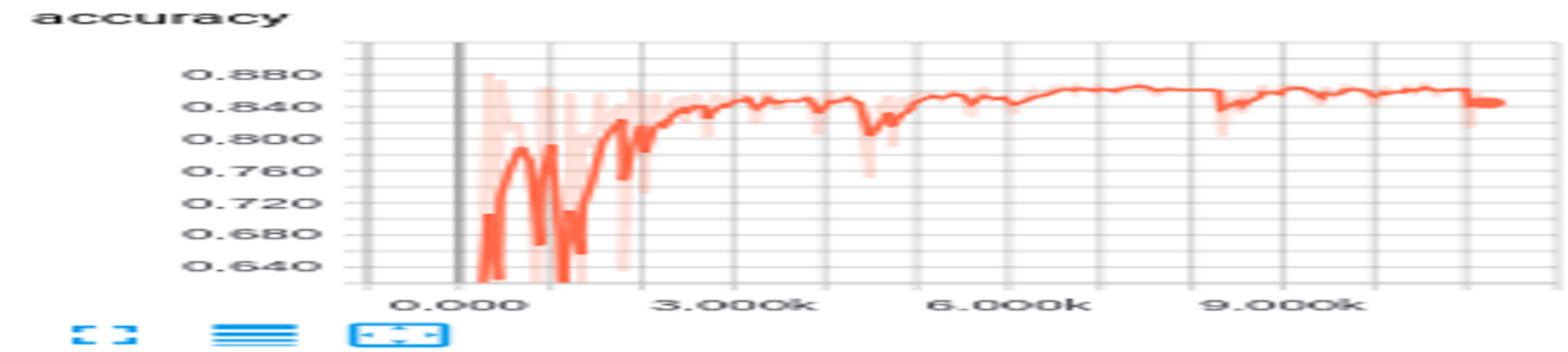
```python
sess=tf.Session()

saver = tf.train.Saver()
print("saver:", saver)
saver.restore(sess,tf.train.latest_checkpoint(utils.get_project_data_path("imdb_lstm")))

minibatch_x ,label=data.val.minibatch(1)
print("minibatch_x:",data.textify(minibatch_x[0]))

print("label:",data.sentiment(label[0]))
pred,yhat=sess.run([output,tf.nn.softmax(output)], feed_dict={x: minibatch_x, phase_train: False})

print("pred:",pred)
print("yhat:",yhat)
print("predicted sentiment:",data.sentiment(data.probstoonehot(yhat)))
```

```
MB memory) -> physical GPU (device: 0, name: GeForce GTX 1080, pci bus id: 0000:01:00.0, compute capability: 6.1)
saver: <tensorflow.python.training.saver.Saver object at 0x7fe833fc21d0>
minibatch_x: <START> those who know who know the kelly legend are hoping that this film would be an accurate depiction of his life may be disappointed with the creativ
e license taken with this film eg naomi watt's character never existed in reality but if you look at it purely as a piece of entertainment it holds up pretty well ledg
ers performance in the title role is quite solid taking the mantle of cinema's best ned not hard considering the previous ned's include yahoo serious mick jagger forme
r carlton champion australian rules football bob chitty a great footballer but a poor actor some location shooting film in the area i live bacchus marsh outside melbou
rne as well as clunes ballarat <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD
```

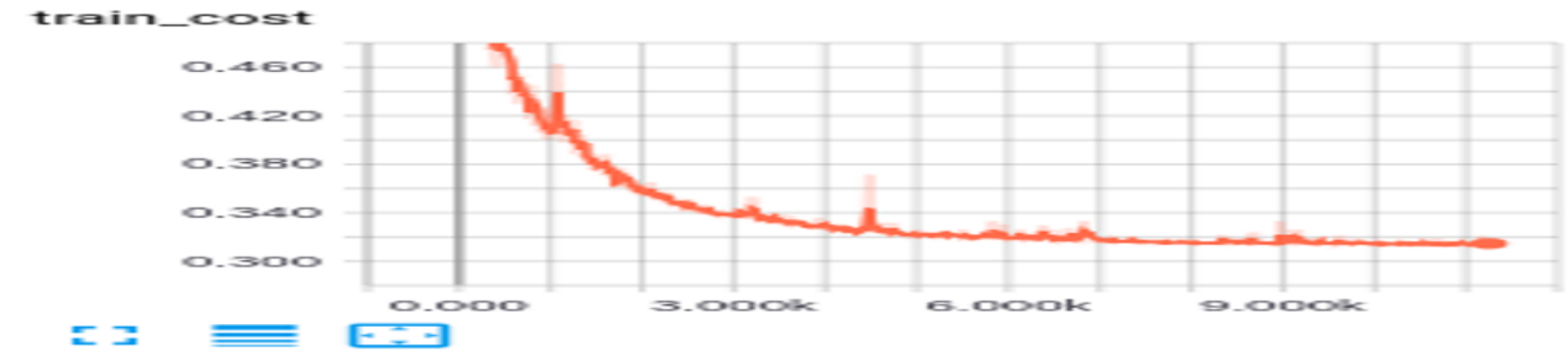
```
label: POSITIVE
pred: [[4.216243e-04 9.995819e-01]]
yhat: [[0.26910657 0.73089343]]
[[0.26910657 0.73089343]]
predicted sentiment: POSITIVE
```

Sentiment Classification

accuracy



train_cost



val_cost

- At its best it has accuracy of 88-1.

Machine Translation

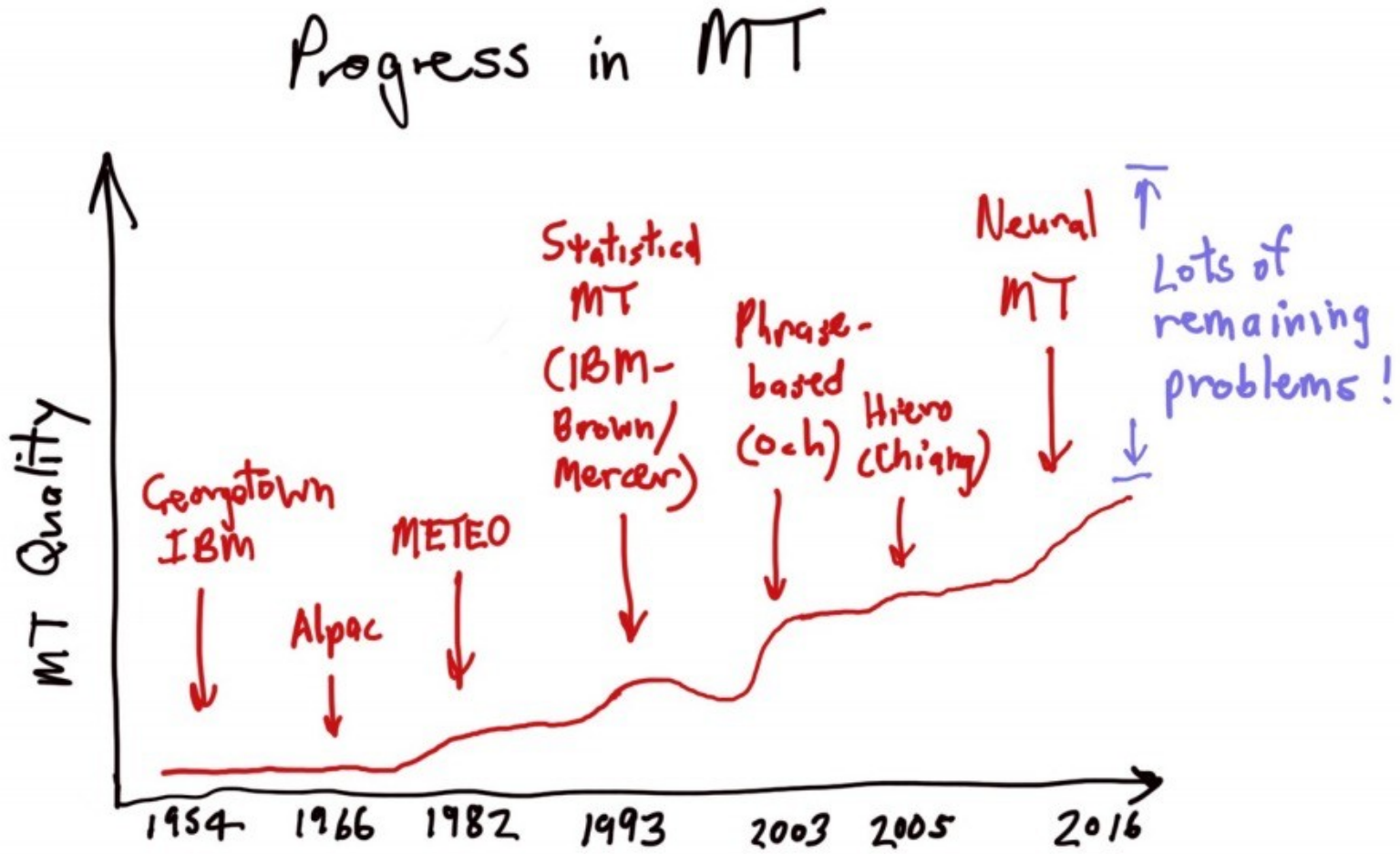
- Translation carried out by a computer
- Classic test of language understanding!
  - Both language **analysis** and **generation**

Machine Translation

- Huge commercial use
  - Google translates over 100 billion words a day
  - Facebook has just rolled out new homegrown MT

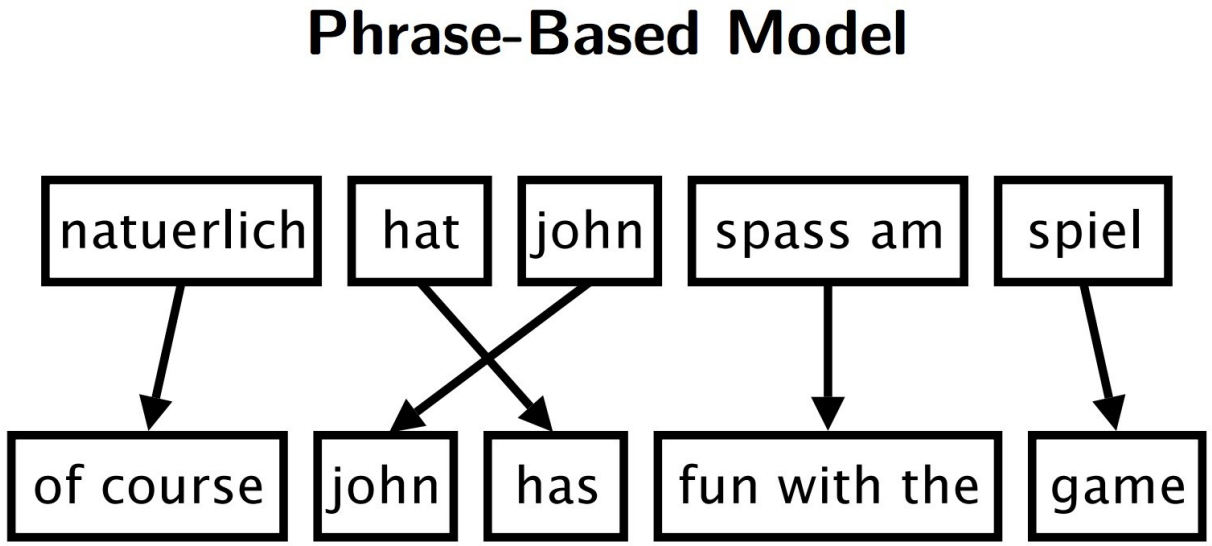
    “When we turned [MT] off for some people, they went nuts!”
  - eBay uses MT to enable cross-border trade

Machine Translation



- Rule-based MT (RBMT)
- Statistical MT (SMT)
- Phrase-based SMT
- Neural MT (NMT)

Machine Translation



- Foreign input is segmented in phrases
- Each phrase is translated into English
- Phrases are reordered

- Rule-based MT (RBMT)
- Statistical MT (SMT)
- **Phrase-based SMT**
- Neural MT (NMT)

Machine Translation:How to evaluate

- Human evaluation
- Automatic evaluation metrics
  - BLEU
  - NIST
  - WER

Machine Translation:How to evaluate

- **Human evaluation**
  - Specially trained human judges
  - Usually studies fidelity, fluency, comprehension, etc.

Machine Translation:How to evaluate

- Human evaluation
- **Automatic evaluation metrics**
  - **BLEU** *(bilingual evaluation understudy)*
    - modified form of precision **(Lec 23)**
    - correlating well with human judgement
    - most widely used

Machine Translation:How to evaluate

- Human evaluation
- **Automatic evaluation metrics**
  - BLEU *(bilingual evaluation understudy)*
  - **NIST**
    - BLEU-score based metric
    - Reward correct rare n-grams

Machine Translation:How to evaluate

- Human evaluation
- **Automatic evaluation metrics**
  - BLEU *(bilingual evaluation understudy)*
  - NIST
  - **WER** *(word error rate)*

$$WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C}$$

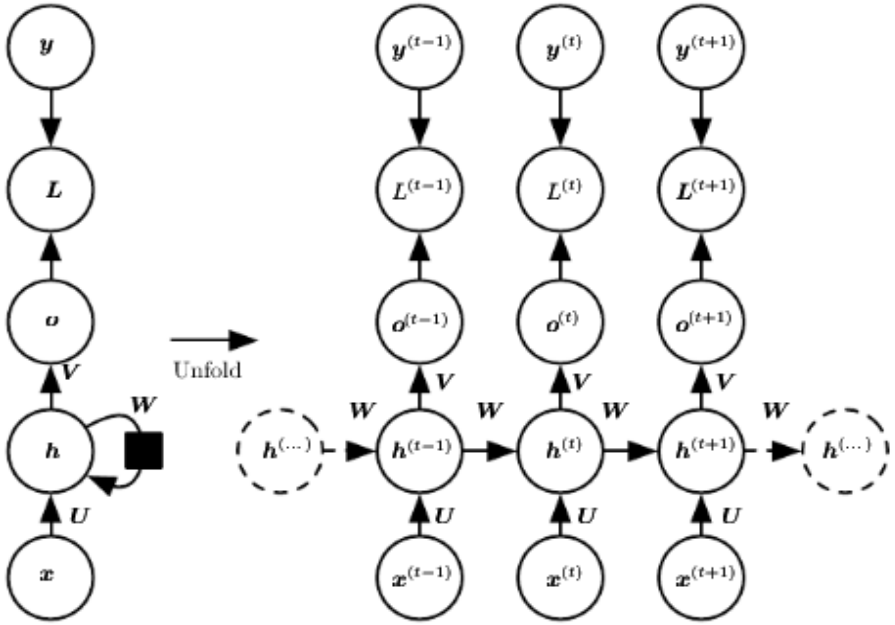
NMT:What is it?

- Neural machine translation (NMT)
- **Goal**:
  - End-to-end trained neural translation model
  - Model entire MT process via one big artificial neural network
- **Formulation**:
  - Sequence-to-sequence modeling
  - RNN encoder-decoder architecture

NMT:What is it?:RNN

Recall: What is RNN?

- Previous session in RNN and variants

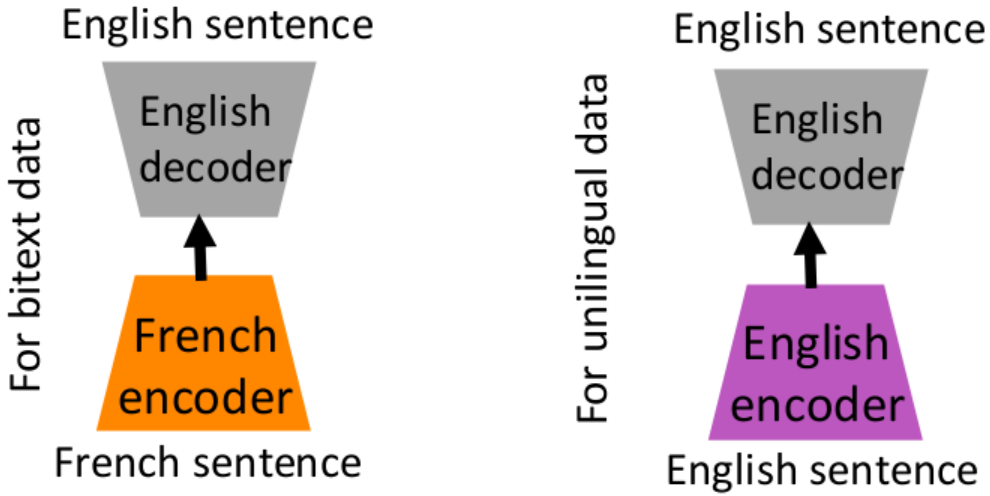


$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \mathrm{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

Figure from: Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. **Deep learning**. MIT Press, 2016.

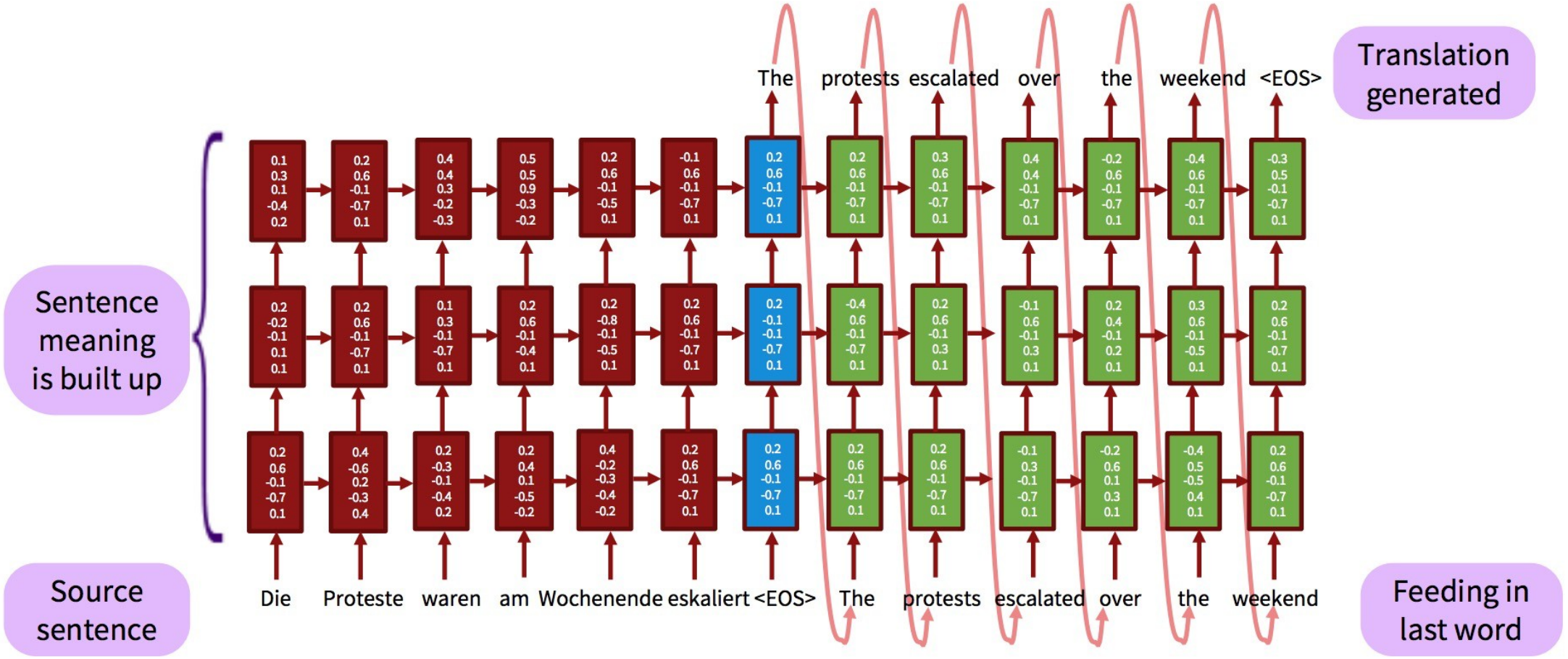
NMT:Encoder-Decoder Architecture

- Intermediate representation of meaning (universal representation)
- Encoder-Decoder architecture
  - Encoder: word sequence (**source language**) → sentence



equence distribution (**target**

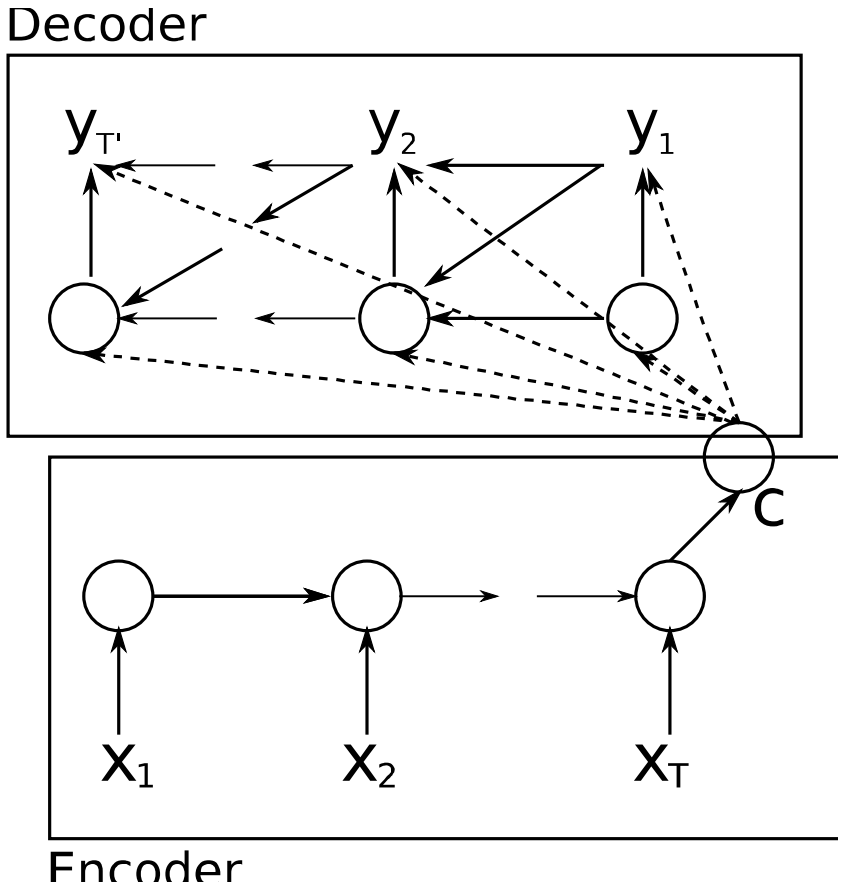
NMT:Encoder-Decoder Architecture



[Neural Machine Translation - Tutorial ACL 2016]

NMT:Encoder-Decoder Architecture

- Encoder-decoder NMT with 2 RNNs
- Any recurrent activation function can be used:
  - GRU [Cho et al., 2014]
  - LSTM [Sutskever et al., 2014]
  - Bi-LSTM (Bidirectional LSTM)
  - CNN [Kalchbrenner & Blunsom, 2013]

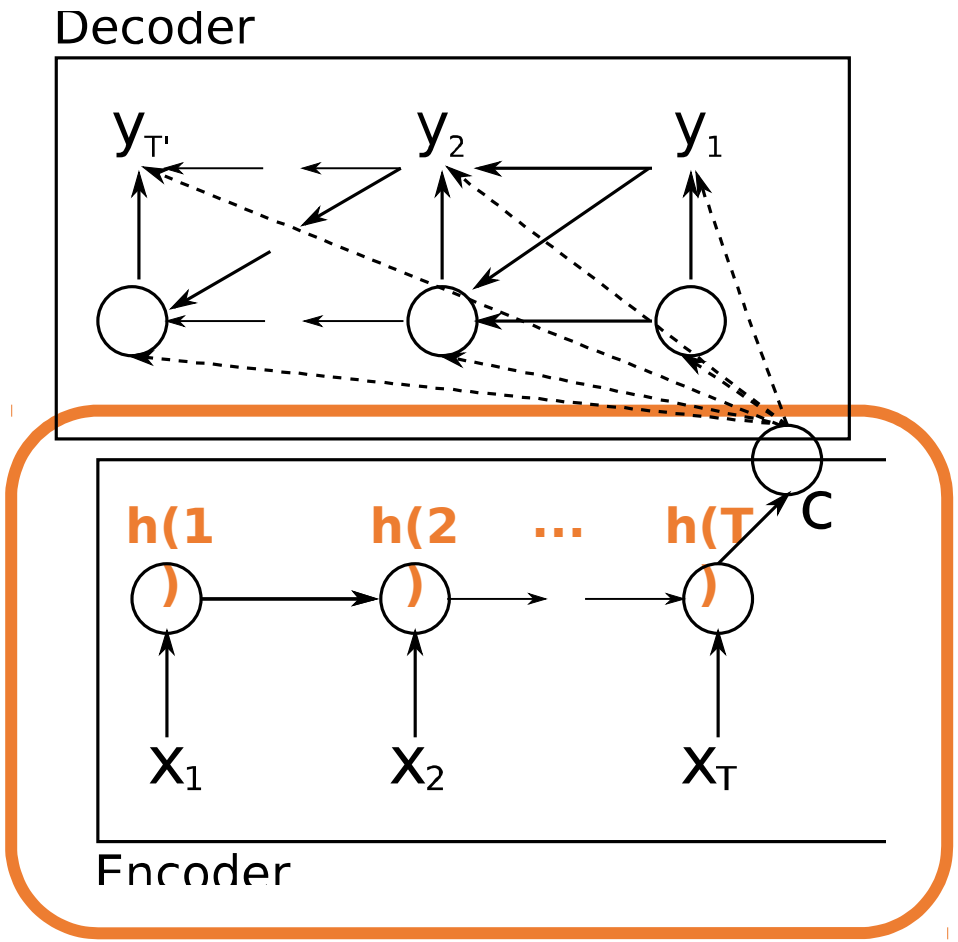


Cho, Kyunghyun, et al.
Learning phrase representations using RNN encoder-decoder for statistical machine translation

arXiv preprint arXiv:1406.1078 (2014).

NMT:Encoder-Decoder Architecture:Encoder

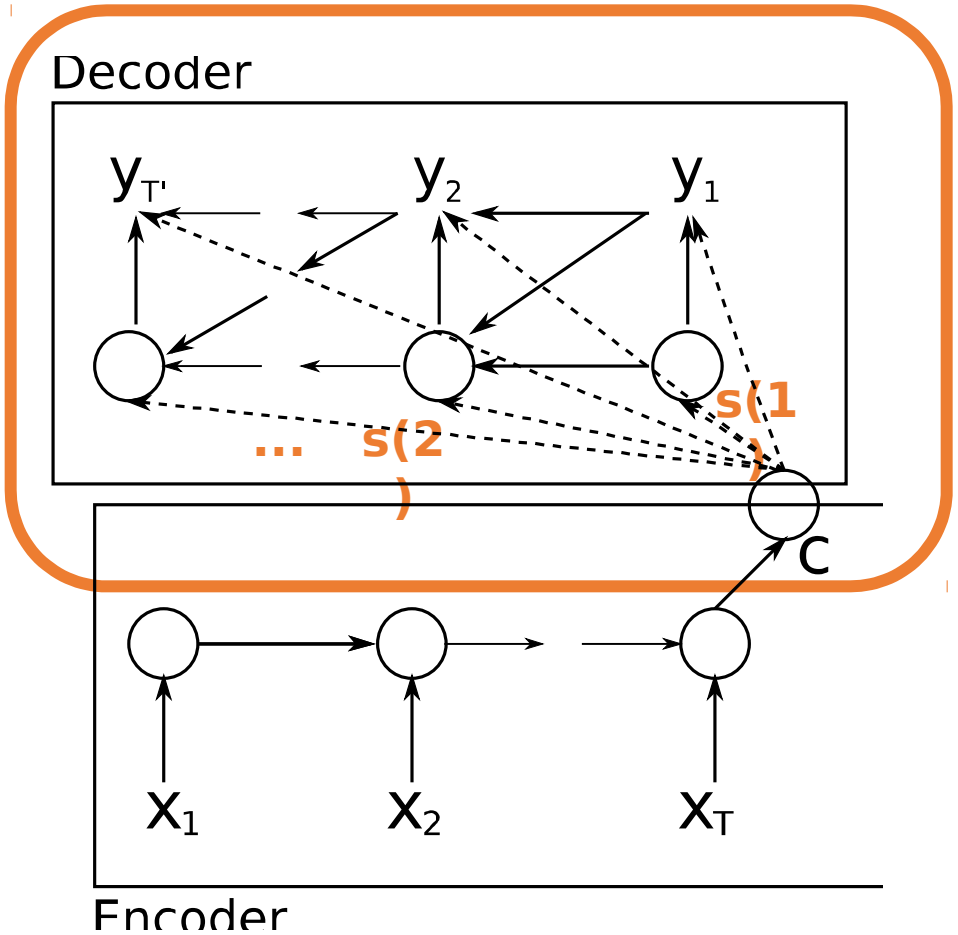
- **Input**:
  - word sequence (source) $\mathrm{x} = (x_1, \cdots, x_T)$
- **Encoder Transition**:
  - $h_t = RNN_{enc}(x_t, h_{t-1})$
- **Context vector**:
  - Represent source sentence as a fixed-length vector
  - $c = h_T$



Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le.
Sequence to sequence learning with neural networks. NIPS 2014

NMT:Encoder-Decoder Architecture:Decoder

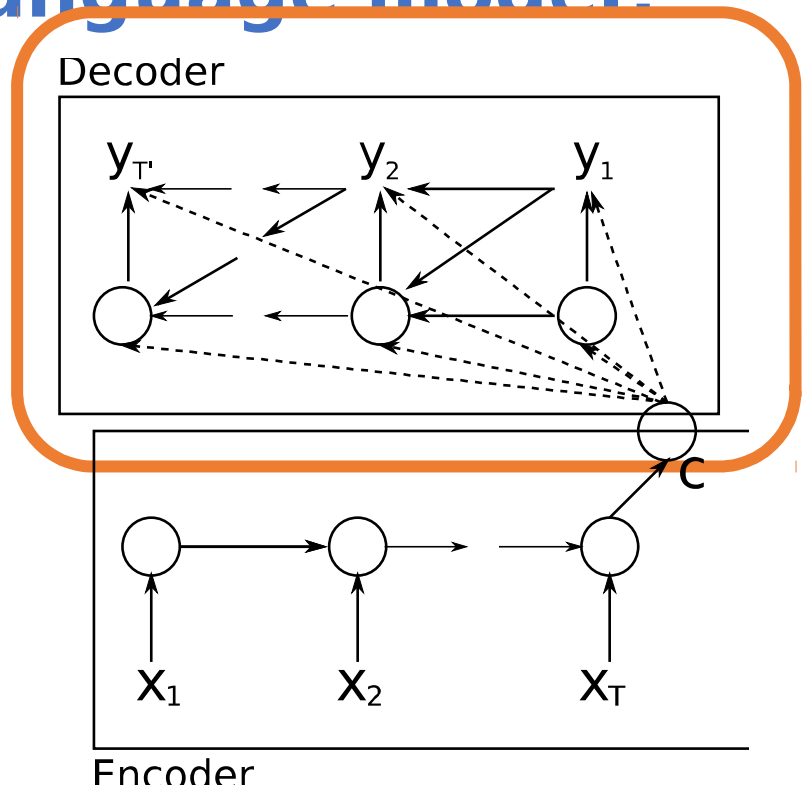
- **Output:**
  - word sequence (target) $\mathrm{y} = (y_1, \cdots, y_{T'})$
- **Decoder Transition**:
  - $s_t = RNN_{dec}(s_{t-1}, y_{t-1}, \boldsymbol{c})$
- **Conditional recurrent language model**
  - $p(\vec{\boldsymbol{y}}|\vec{\boldsymbol{x}}) = \prod_{i=1}^{T'} p(y_t|y_1 \cdots, y_{t-1}; \boldsymbol{c})$
  - $p(y_t|y_1 \cdots, y_{t-1}; \boldsymbol{c}) = \mathrm{g}(\mathrm{s_t}, \mathrm{y_{t-1}}, \mathbf{c})$

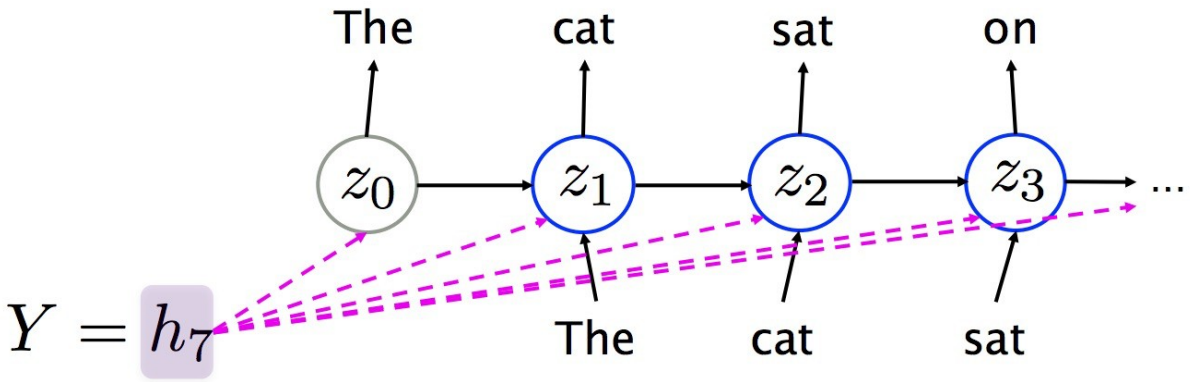


g(·): Function that produce valid probs, e.g. softmax

NMT:Encoder-Decoder Architecture

How to generate word sequence from conditional language model?

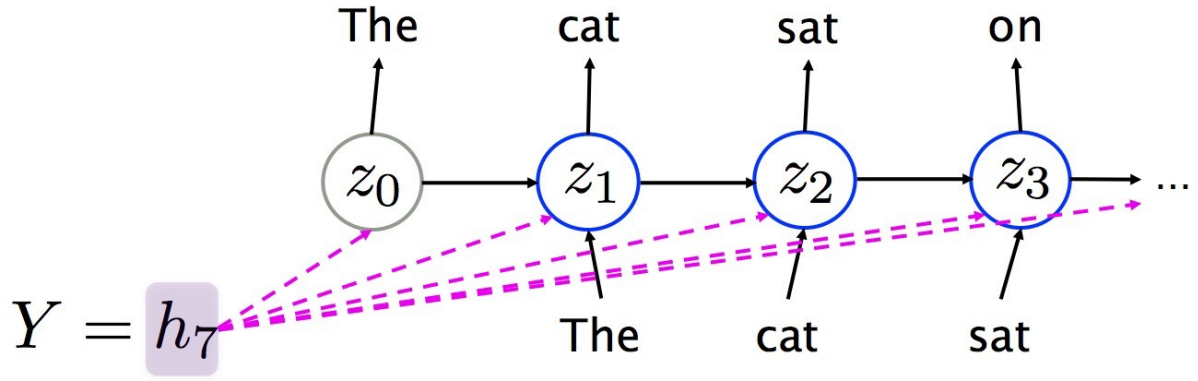




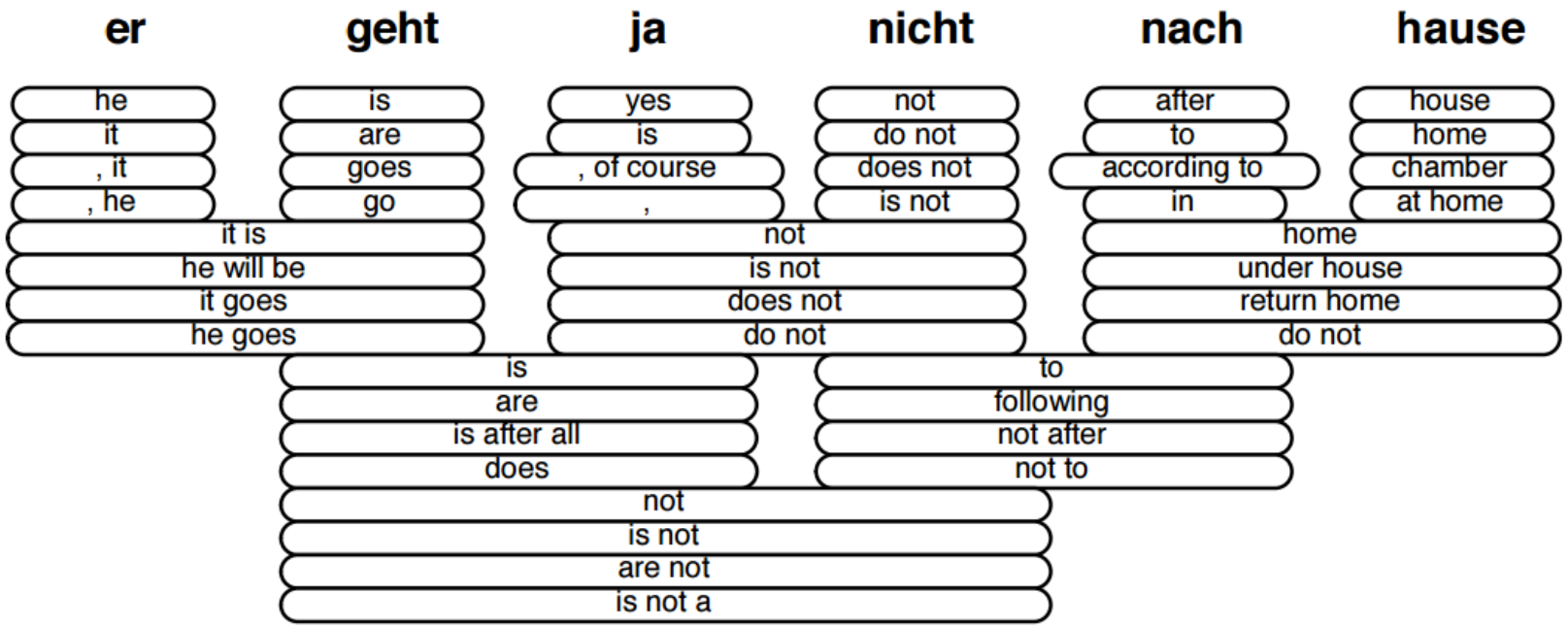
NMT:Encoder-Decoder Architecture:Decoder

How to generate word sequence from conditional language model?

- **Decoding:**
  - Language generation: representation (vector) □ word
- **Decoding Strategies:**
  - Naïve Search
  - Greedy Search
  - Beam Search



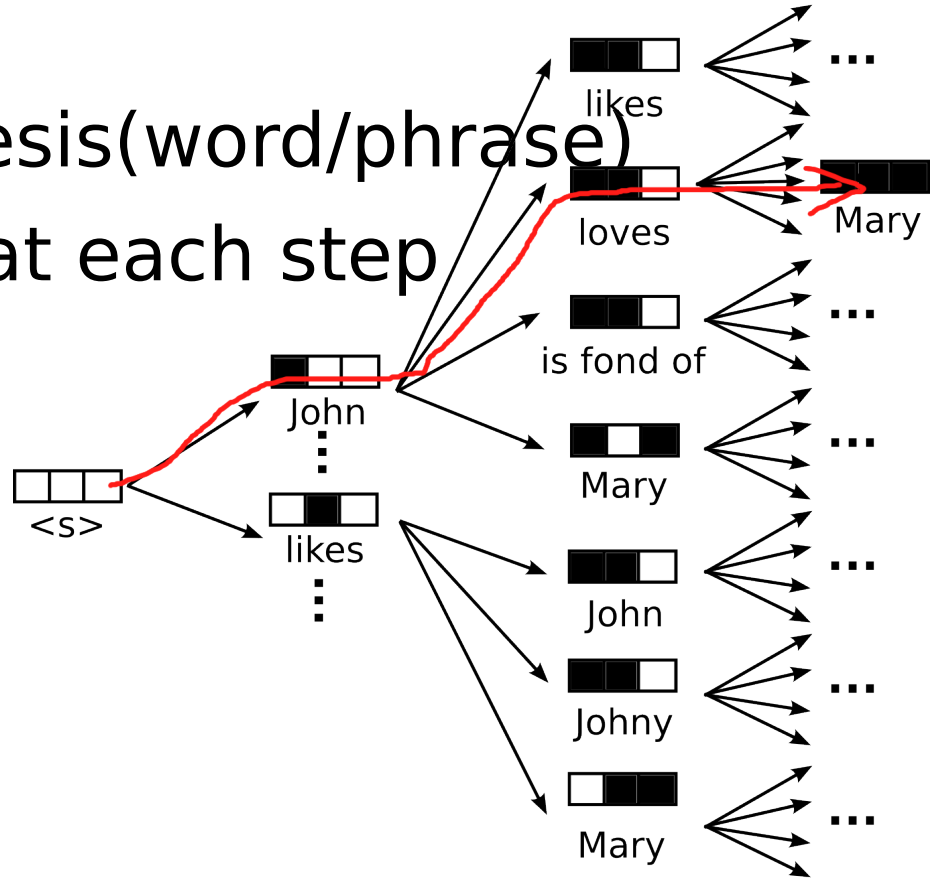
NMT:Encoder-Decoder Architecture:Decoder:Decoding Strategy:Naive(Brute Force)



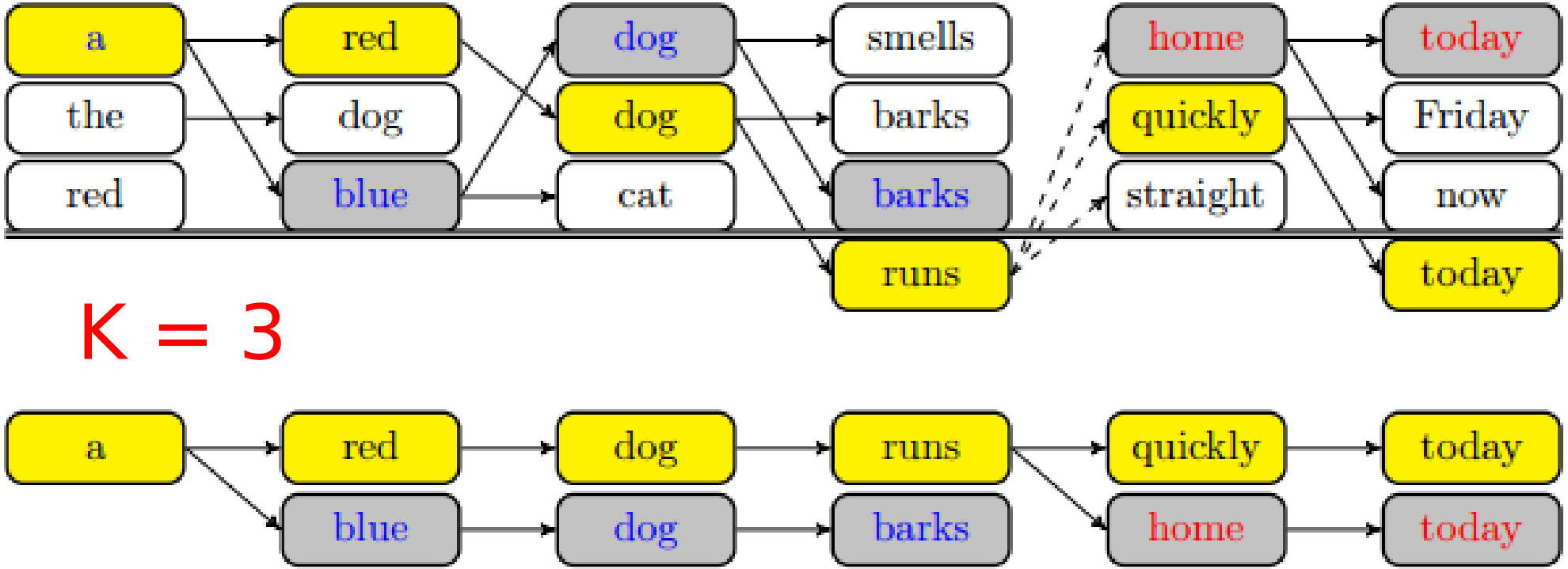
- Align and score every possible translation combinations
- Choose the best combination
- Pros: Simple
- Cons: Exponential in time and space

NMT:Encoder-Decoder Architecture:Decoder:Decoding Strategy:Greedy

- Choose the most likely hypothesis(word/phrase) and maximize the probability at each step
- Pros
  - Efficient
  - Low cost in space and time
- Cons
  - Suboptimal



NMT:Encoder-Decoder Architecture:Decoder:Decoding Strategy:Beam

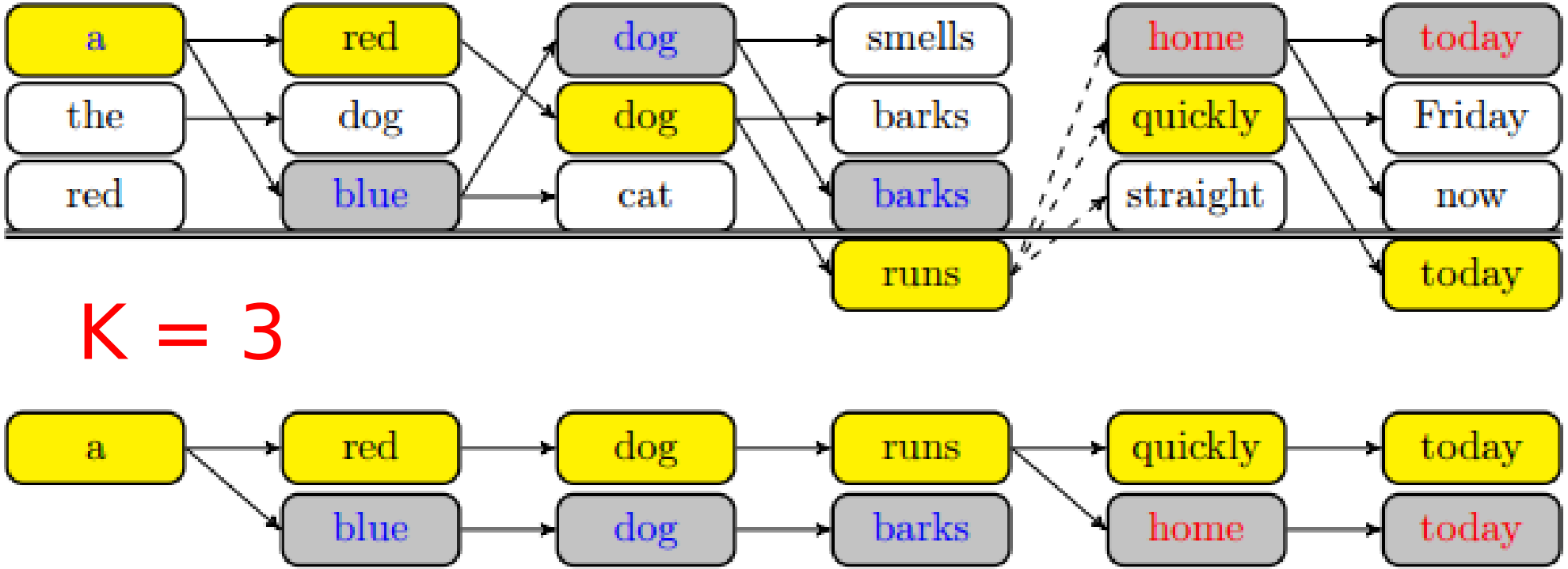


Maintain a set of k(beam size) hypotheses at each step

$$\mathcal{H}_{t-1} = \{(\tilde{x}_1^1, \tilde{x}_2^1, \ldots, \tilde{x}_{t-1}^1), (\tilde{x}_1^2, \tilde{x}_2^2, \ldots, \tilde{x}_{t-1}^2), \ldots, (\tilde{x}_1^K, \tilde{x}_2^K, \ldots, \tilde{x}_{t-1}^K)\}$$

Wiseman, Sam, and Alexander M. Rush. "**Sequence-to-sequence learning as beam-search optimization**." *arXiv preprint arXiv:1606.02960* (2016).

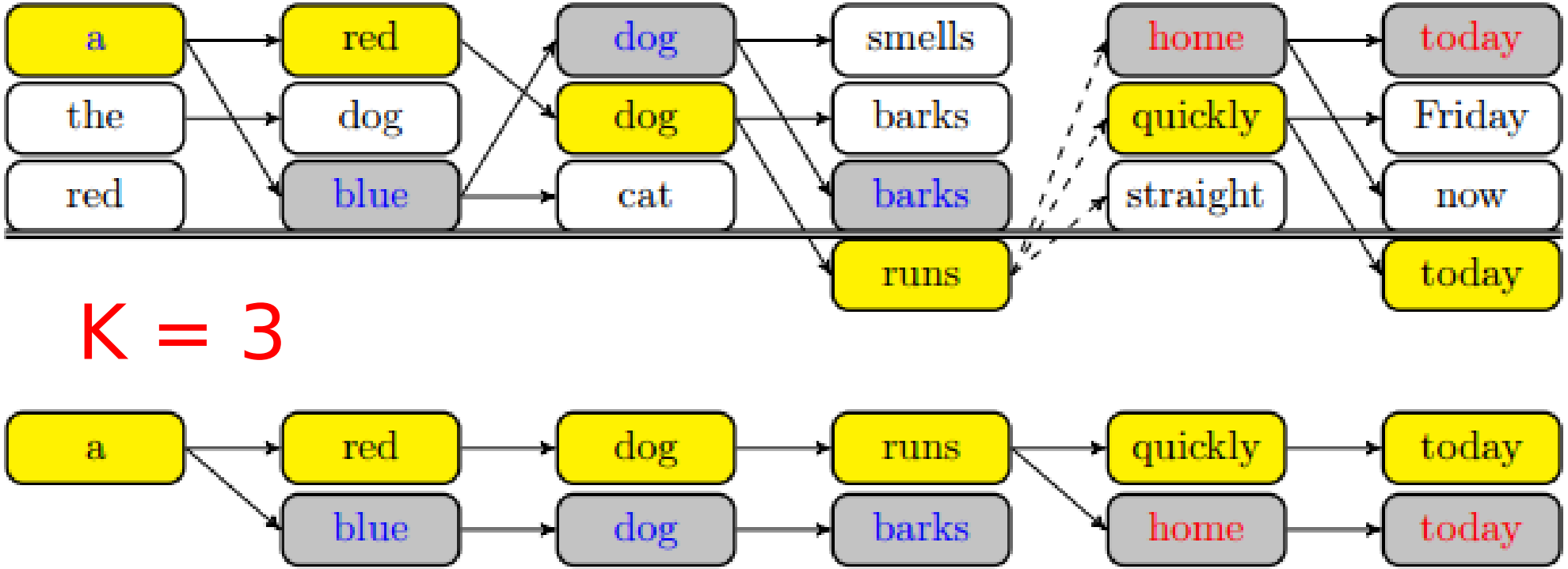
NMT:Encoder-Decoder Architecture:Decoder:Decoding Strategy:Beam



Expand each hypothesis with the best next-word at step t

$$\mathcal{H}_t^k = \{(\tilde{x}_1^k, \tilde{x}_2^k, \ldots, \tilde{x}_{t-1}^k, v_1), (\tilde{x}_1^k, \tilde{x}_2^k, \ldots, \tilde{x}_{t-1}^k, v_2), \ldots, (\tilde{x}_1^k, \tilde{x}_2^k, \ldots, \tilde{x}_{t-1}^k, v_{|V|})\}$$

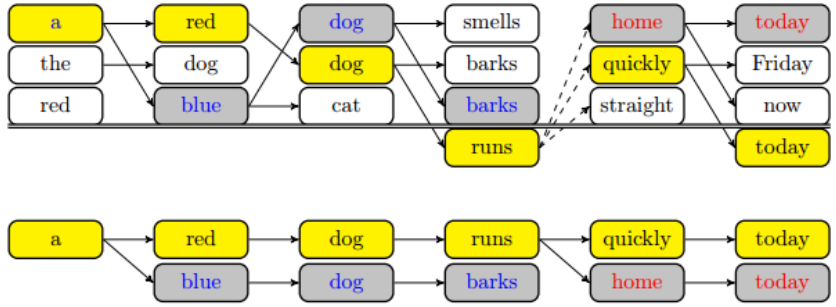
NMT:Encoder-Decoder Architecture:Decoder:Decoding Strategy:Greedy



Choose the top-k hypotheses from th $\mathcal{H}_t = \cup_{k=1}^{K} \mathcal{B}_k$,

$\mathcal{B}_k = \underset{\tilde{X} \in \mathcal{A}_k}{\arg\max} \log p(\tilde{X}|Y),\ \mathcal{A}_k = \mathcal{A}_{k-1} - \mathcal{B}_{k-1},$ and $\mathcal{A}_1 = \cup_{k'=1}^{K} \mathcal{H}_t^{k'}$.

NMT:Encoder-Decoder Architecture:Decoder:Decoding Strategy:Greedy



- Greedy Search is a conditional beam with beam size = 1
- Pros
  - Avoid some suboptimal solutions
  - Computationally cheaper than brute-force
- Cons
  - Hard to parallelize
  - Still possible to fall in suboptimal

NMT:Encoder-Decoder Architecture:Training

- **Jointly** train encoder-decoder with **paralleled corpus** (i.e. source and translated sentence pairs)
- **Objective**: maximize conditional log-likelihood

θ: set of model parameters

$(\vec{x}^{(n)}, \vec{y}^{(n)})$: input, output sequence pair

where $\vec{x}^{(n)}=(x_1^{(n)},\ldots, x_T^{(n)})$, $\vec{y}^{(n)}=(y_1^{(n)},\ldots, y_{T'}^{(n)})$

- **Optimization**: train with SGD (BPTT)

NMT:Encoder-Decoder Architecture:Training

- **Quantitative results** (BLEU Score):
  - NOT beat state-of-the-art
  - First time a pure neural translation system outperforms phrase-based SMT baseline by a sizable margin

Method	test BLEU score (ntst14)
Baseline System [29]	33.30
Cho et al. [5]	34.54
State of the art [9]	**37.0**
Rescoring the baseline 1000-best with a single forward LSTM	35.61
Rescoring the baseline 1000-best with a single reversed LSTM	35.85
Rescoring the baseline 1000-best with an ensemble of 5 reversed LSTMs	**36.5**
Oracle Rescoring of the Baseline 1000-best lists	~45

Phrase-based SMT baseline

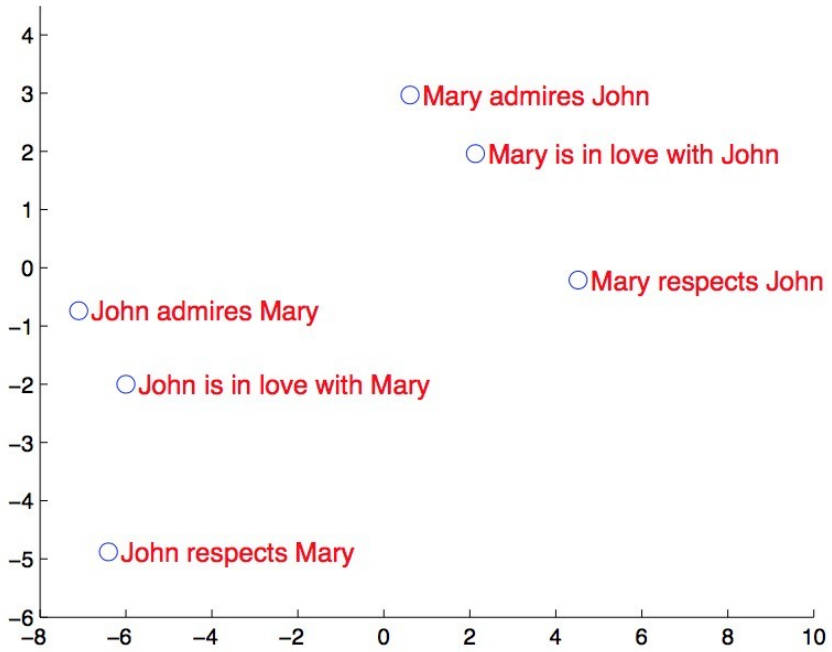
Best NMT

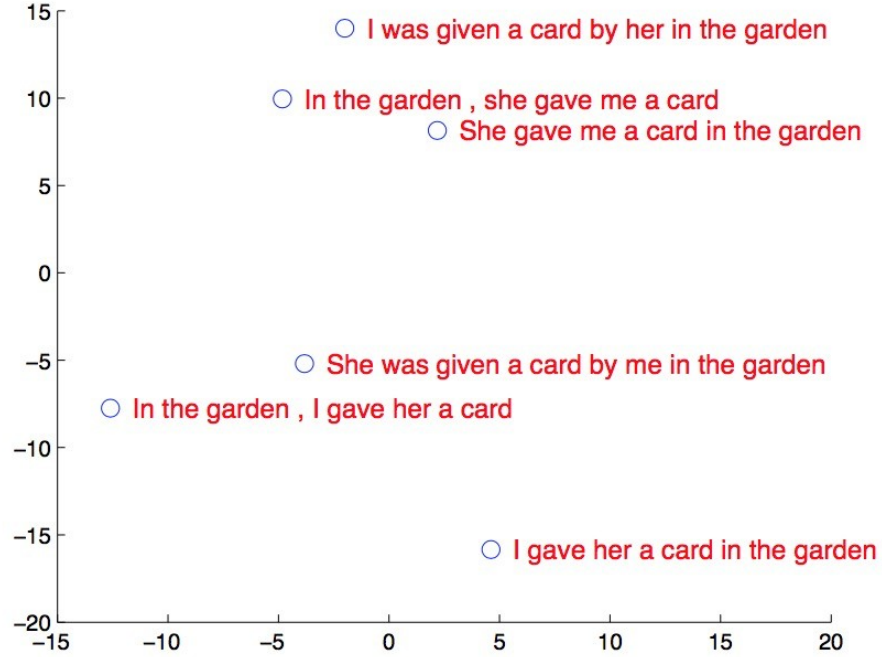
Table 2: Methods that use neural networks together with an SMT system on the WMT'14 English to French test set (ntst14).

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le.
Sequence to sequence learning with neural networks. NIPS 2014

NMT:Encoder-Decoder Architecture:Performance

- **Distributed Representation**
  - Turn a sequence of words into a fixed-dimensional vector
  - Learn representations for phrases





NMT:Encoder-Decoder Architecture:Big wins

- **End-to-end training**
  - **All** parameters are simultaneously optimized by MLE with SGD
- **Distributed representations share strength**
  - Better exploitation of word and phrase similarities
- **Better exploitation of context**
  - NMT can use a much bigger **context** – both source and partial target text – to translate more accurately

NMT:Encoder-Decoder Architecture:Implementation

I	read	.	<PAD>	<PAD>	<PAD>	<PAD>
Je	lis	.	<EOS>	<PAD>	<PAD>	<PAD>
See	you	in	a	little	while	.
A	tout	a	l'heure	<EOS>	<PAD>	<PAD>
			...			

- Naive strategy for padding

Bucket i

I	read	.	<PAD>			
Je	lis	.	<EOS>			
		...				

...

Bucket j

See	you	in	a	little	while	.
A	tout	a	l'heure	<EOS>	<PAD>	<PAD>
		...				

- Padding with bucket

Bucket i

<PAD>	<PAD>	.	read	I		
<GO>	Je	lis	.	<EOS>		
		...				

...

Bucket j

.	while	little	in	a	you	See
<GO>	A	tout	a	l'heure	<EOS>	<PAD>
		...				

- Final, Padding with bucket, reversing the input, and adding the GO token.

NMT:Encoder-Decoder Architecture:Implementation

```
# Build RNN cell
encoder_cell = tf.nn.rnn_cell.BasicLSTMCell(num_units)

# Run Dynamic RNN
#   encoder_outputs: [max_time, batch_size, num_units]
#   encoder_state: [batch_size, num_units]
encoder_outputs, encoder_state = tf.nn.dynamic_rnn(
    encoder_cell, encoder_emb_inp,
    sequence_length=source_sequence_length, time_major=True)
```

- Encoder
- Loaded Embeddings are fed as input to the main network.

NMT:Encoder-Decoder Architecture:Implementation

```
# Build RNN cell
decoder_cell = tf.nn.rnn_cell.BasicLSTMCell(num_units)
```

• Decoder

```
# Helper
helper = tf.contrib.seq2seq.TrainingHelper(
    decoder_emb_inp, decoder_lengths, time_major=True)
# Decoder
decoder = tf.contrib.seq2seq.BasicDecoder(
    decoder_cell, helper, encoder_state,
    output_layer=projection_layer)
# Dynamic decoding
outputs, _ = tf.contrib.seq2seq.dynamic_decode(decoder, ...)
logits = outputs.rnn_output
```

NMT:Encoder-Decoder Architecture:Implementation

```
crossent = tf.nn.sparse_softmax_cross_entropy_with_logits(
    labels=decoder_outputs, logits=logits)
train_loss = (tf.reduce_sum(crossent * target_weights) /
    batch_size)
```

- Loss function

```
# Calculate and clip gradients
params = tf.trainable_variables()
gradients = tf.gradients(train_loss, params)
clipped_gradients, _ = tf.clip_by_global_norm(
    gradients, max_gradient_norm)
```

- gradients

```
# Optimization
optimizer = tf.train.AdamOptimizer(learning_rate)
update_step = optimizer.apply_gradients(
    zip(clipped_gradients, params))
```

- optimization

NMT:Encoder-Decoder Architecture:Advanced:Better Performance

- The **attention** aspect
  - Global and local information
- The **vocabulary** aspect
  - Decoding strategies
- The **data** aspect
  - Utilize more data sources

NMT:Encoder-Decoder Architecture:Advanced:Better Performance

- The **attention** aspect
  - Pay more attention to critical information
- The **vocabulary** aspect
  - Ensemble decoding
  - Rare word translation
- The **data** aspect
  - Monolingual data
  - Multilingual data

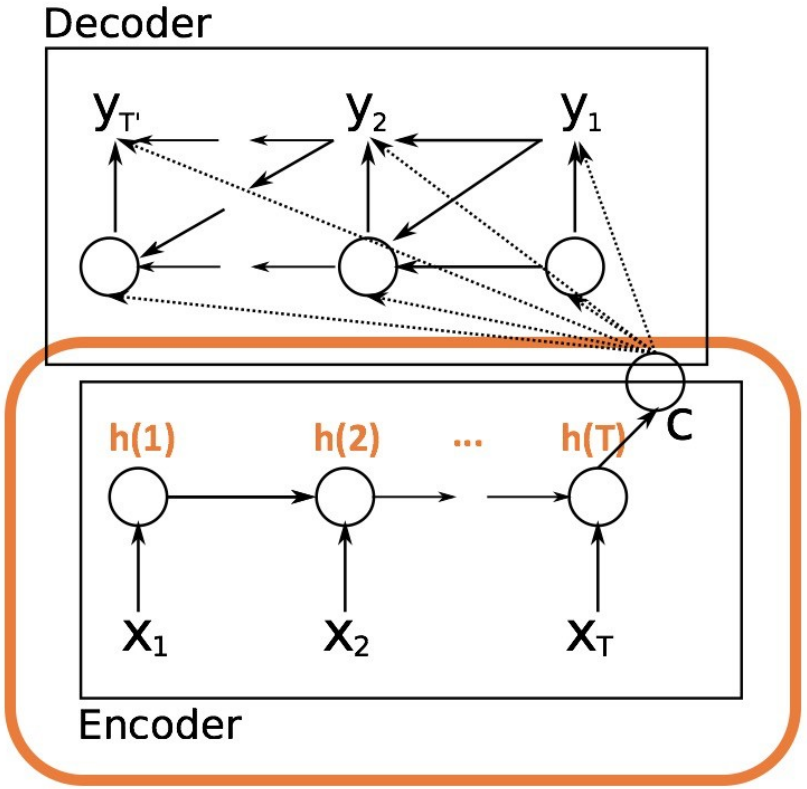
Vanilla structure: what is the bottleneck?

NMT:Encoder-Decoder Architecture:Advanced:Vanilla seq-to-seq problems

Vanilla structure: what is the bottleneck?

Recall the encoder-RNN:

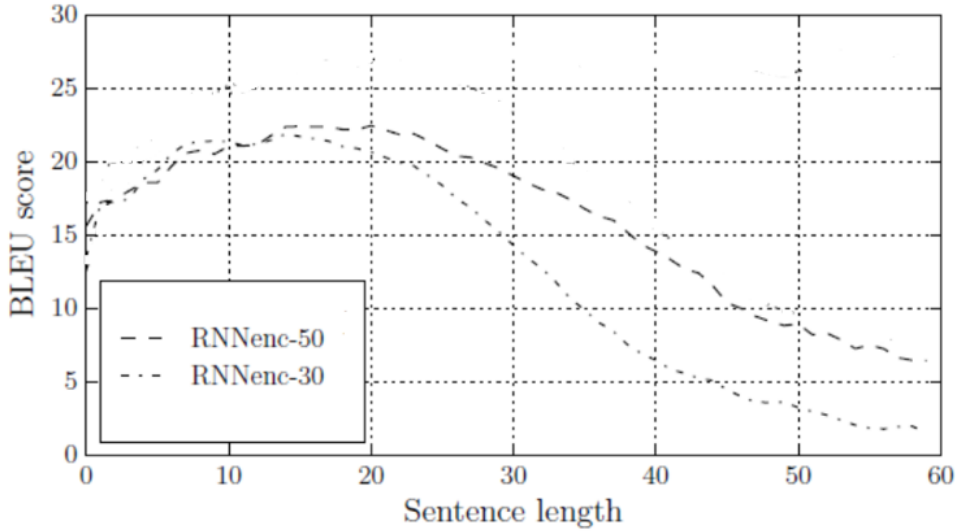
- Encode source sequence into one fixed-length vector:
- is used in decoder at every step to generate the translation



NMT:Encoder-Decoder Architecture:Advanced:Vanilla seq-to-seq problems

Vanilla structure: what is the bottleneck?

- Vanilla model compress everything into a fixed-length
- g sequences



D. Bahdanau, K. Cho, Y. Bengio.
Neural Machine Translation by Jointly Learning to Align and Translate. ICLR 2015.

NMT:Encoder-Decoder Architecture:Advanced:Vanilla seq-to-seq problems:Solution

How to avoid "the curse of length"?

- Memory:
  - Remember important information for future decision making
- **Attention**:
  - Pay attention to critical units for current decision making

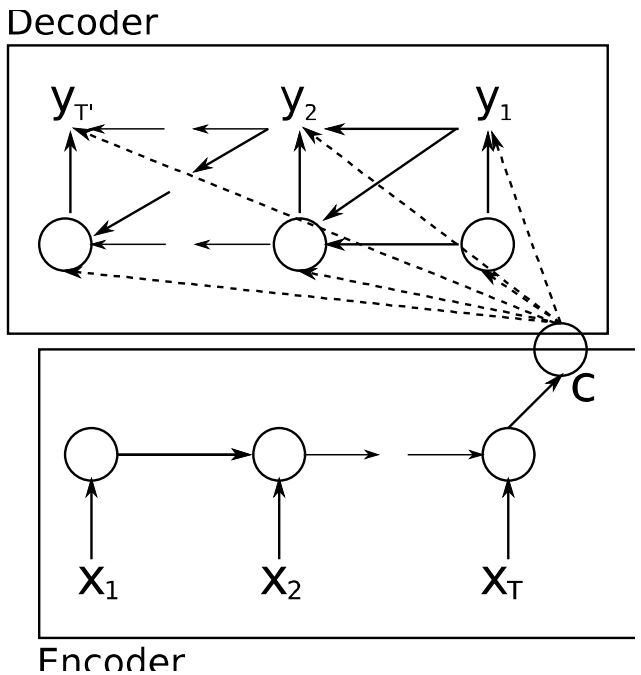
NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism

Attention also used in …

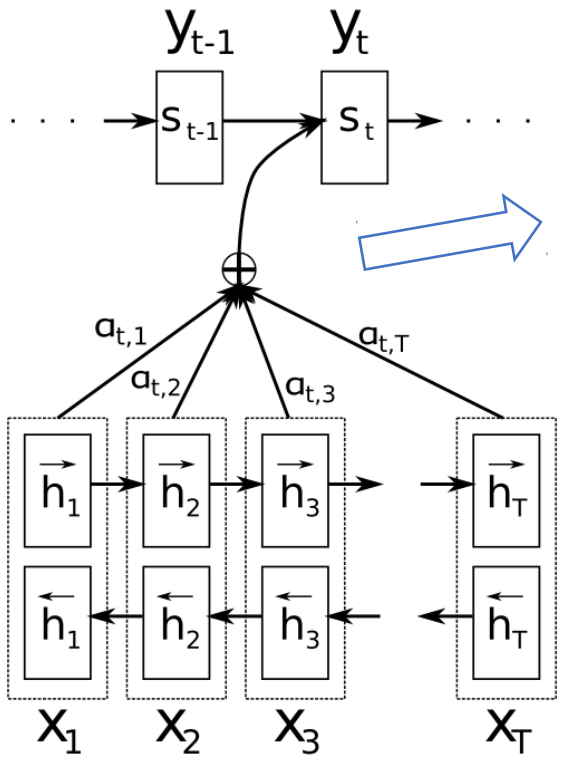
- Image Generation
- Image text embeddings
- Question Answering
- ……

Attention: focus on **critical** information for **current** decision making

NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism



A fixed source representation $c = h_T$ is used for decoding each word



Context vector c_t : (Soft Alignment) Paying attention to **different phrases** in the source sentences when generating words

Vanilla Sequence to Sequence

Attention-based Sequence to Sequence

D. Bahdanau, K. Cho, Y. Bengio.
Neural Machine Translation by Jointly Learning to Align and Translate. ICLR 2015.

NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Aligning encoder decoder

- **Key idea:**

  For different position t in decoder, use a different context vector $\boldsymbol{c_t}$

Without Attention

- $p(y_t|y_1 \cdots, y_{t-1}; c) \propto g(y_t; y_{t-1}, s_t, \boldsymbol{c})$
- RNN hidden state:

$$s_t = RNN_{dec}(s_{t-1}, y_{t-1}, \boldsymbol{c})$$

With Attention

- $p(y_t|y_1 \cdots, y_{t-1}; c) \propto g(y_t; y_{t-1}, s_t, \boldsymbol{c_t})$
- RNN hidden state:

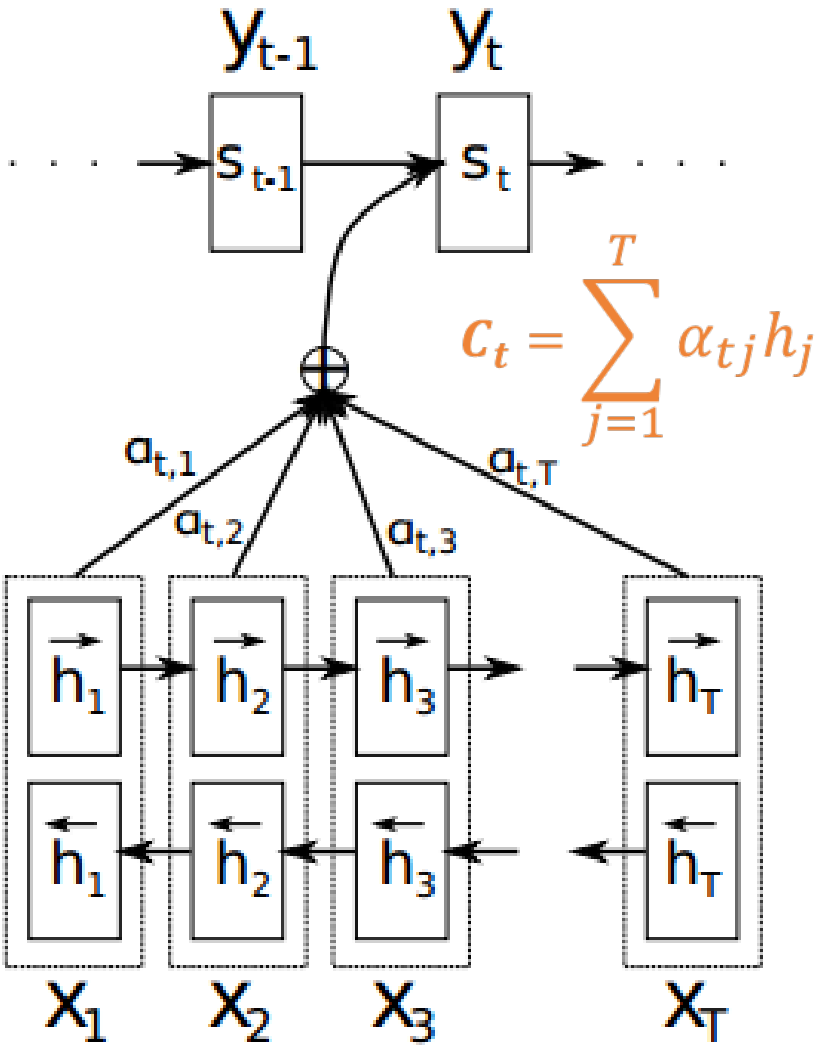
$$s_t = RNN_{dec}(s_{t-1}, y_{t-1}, \boldsymbol{c_t})$$

NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Context Vector

- **Context vector c_t**
  - Not fixed, position-sensitive
  - Weighted sum of different positions in encoder
  - $c_t = \sum_{j=1}^{T} \alpha_{tj} h_j$

- **Encoder Hidden States h_j**
  - Bi-LSTM encoder
  - $h_j = [\mathrm{h}_\mathrm{j}^{\mathrm{forward}}; \mathrm{h}_\mathrm{j}^{\mathrm{backward}}]$

How to weight different position h_j?

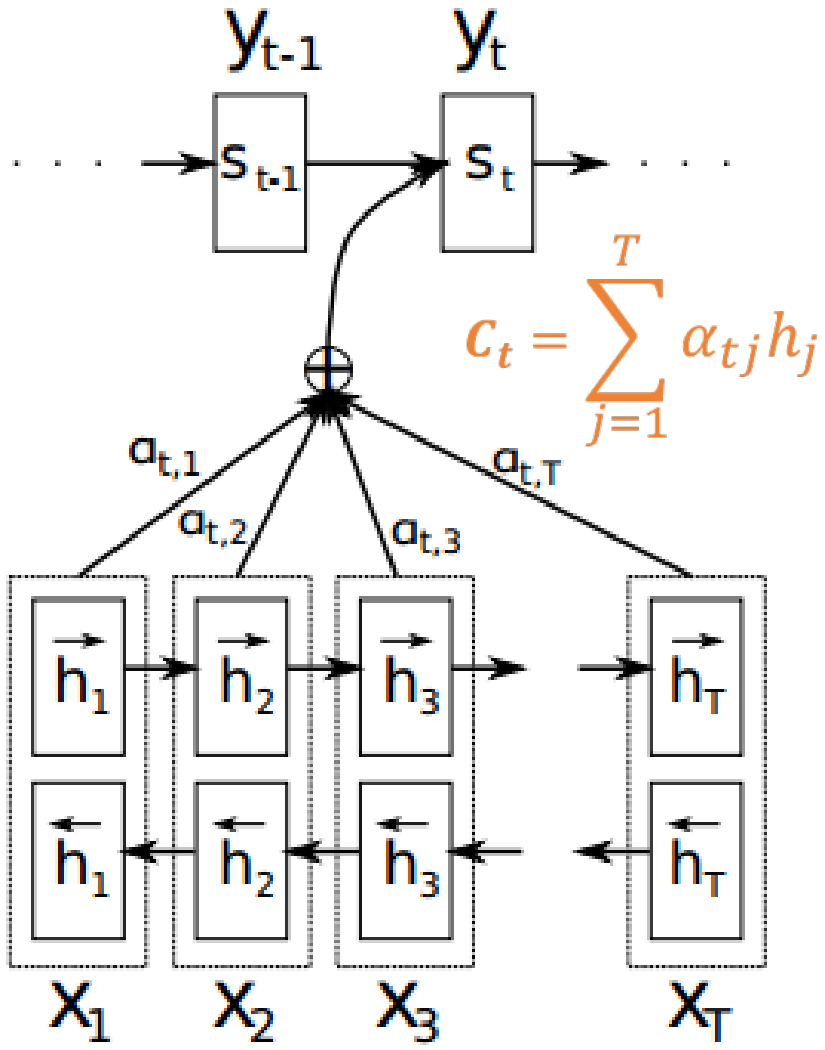


NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Context Vector

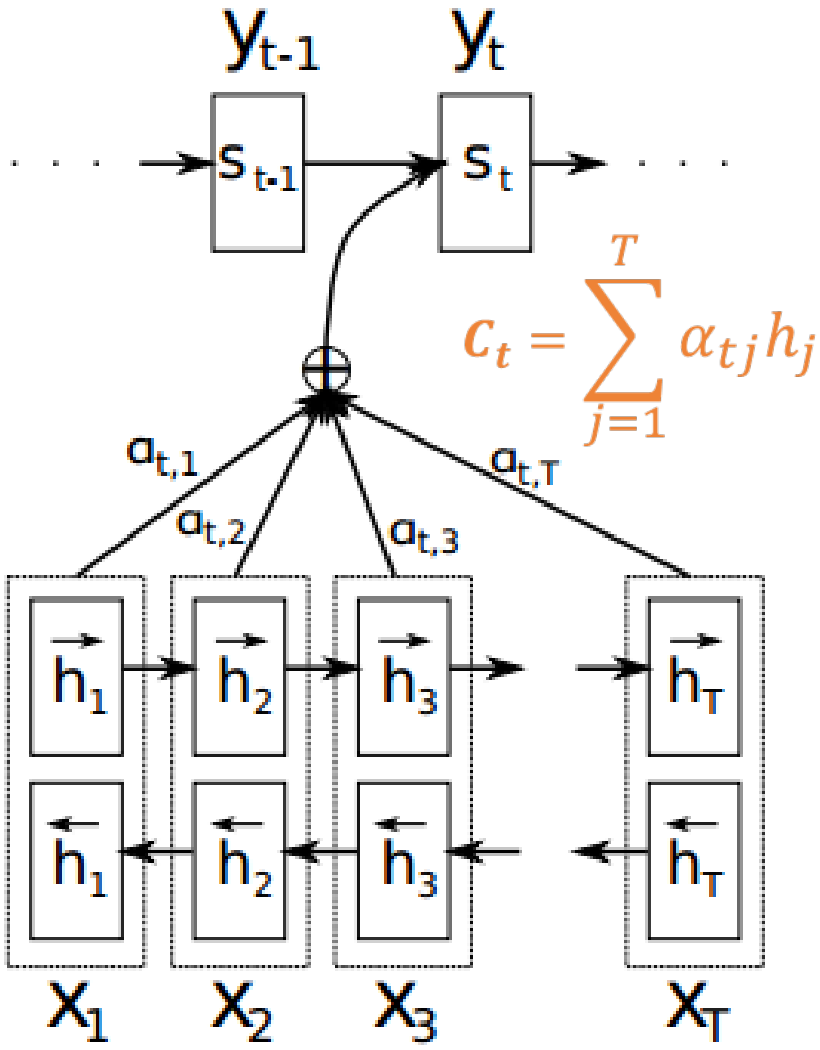
- **Context vector c_t**
  - Weighted sum of different parts in x
  - $c_t = \sum_{j=1}^{T} \alpha_{tj} h_j$

How to weight different parts h_j?

- Score alignment between target and source hidden states
- Score → alignment weights

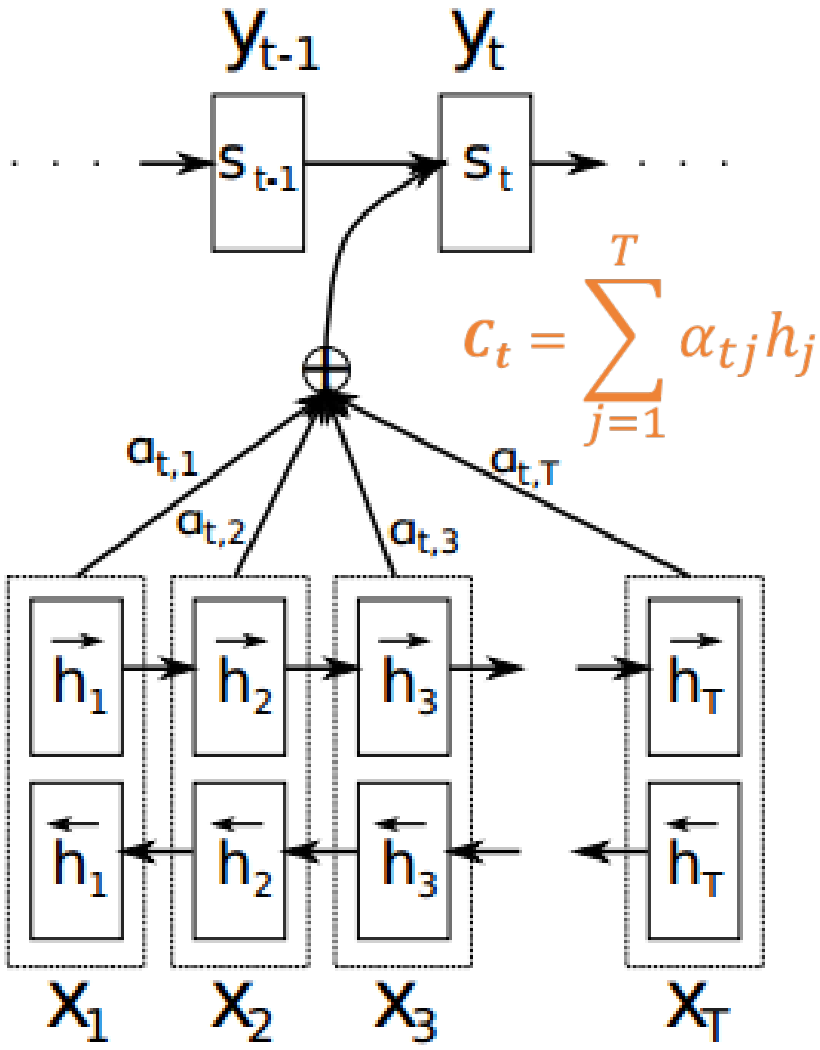


NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Context Vector

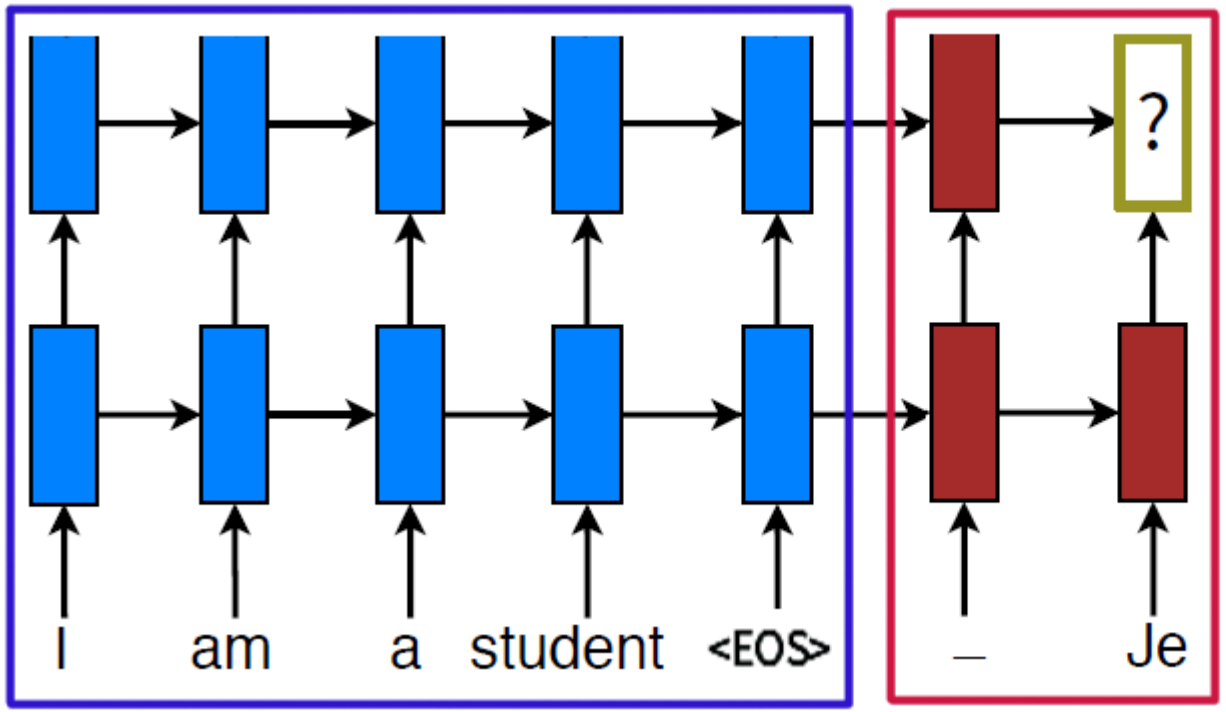


NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Context Vector

- **Context vector c_t**
  - Weighted average of different parts in x
  - $c_t = \sum_{j=1}^{T} \alpha_{tj} h_j$
- **Alignment model:**
  - Score: $e_{tj} = \boldsymbol{a}(s_{t-1}, h_j)$
- **Weight**:
  - $\alpha_{tj} = softmax(e_{tj}) = \frac{\exp(e_{tj})}{\sum_{k=1}^{T} \exp(e_{tk})}$

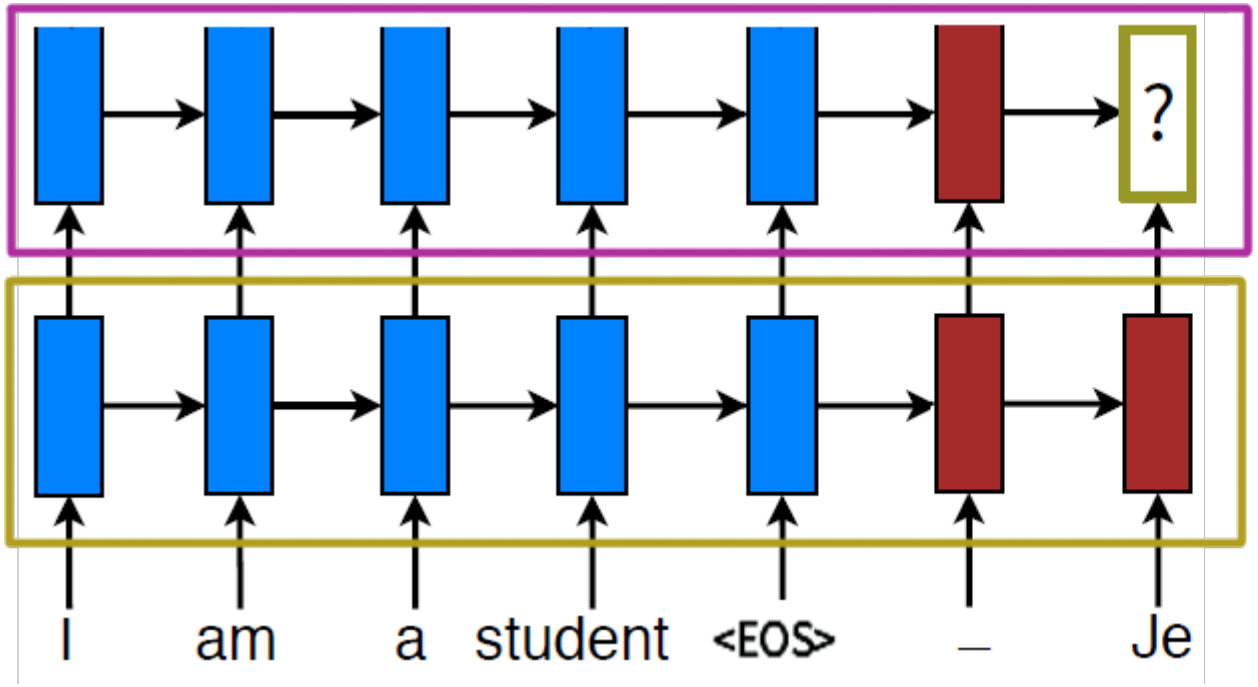


NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Context Vector



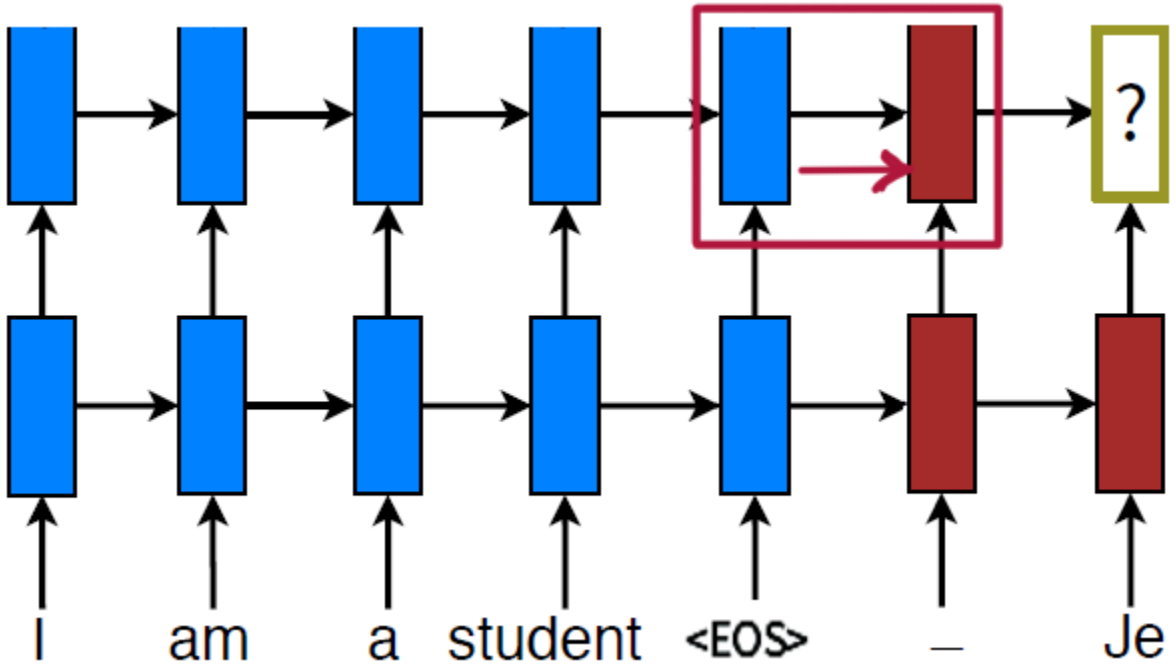
- Encoder
- Decoder

NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Context Vector



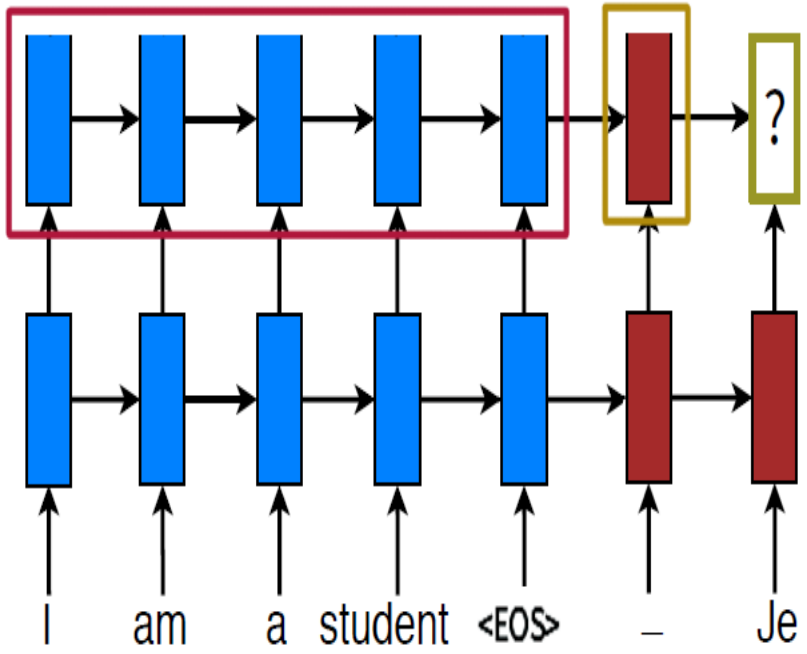
- RNN Layer.
- Word Embedding

NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Context Vector



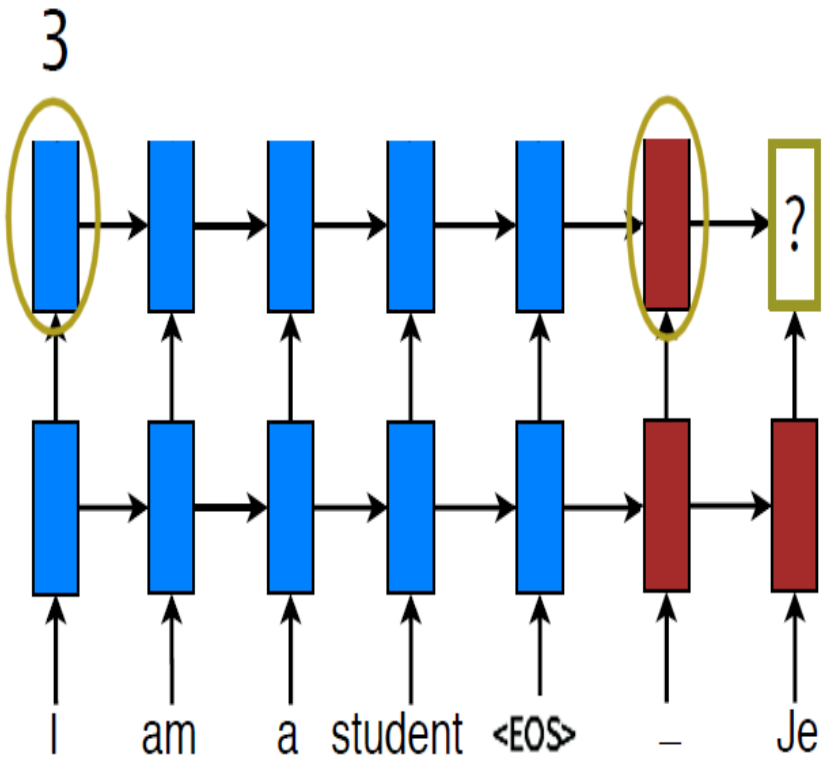
- Before decoding the last state copied over for the decoder.

NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Context Vector



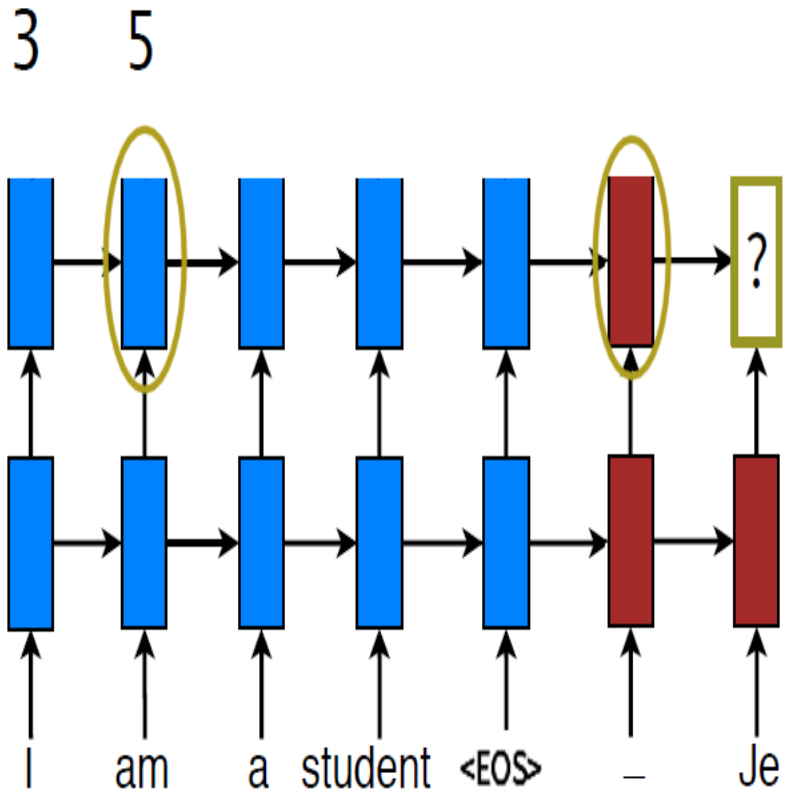

```
def _attend(self, input_vectors, state):
    w1 = dy.parameter(self.attention_w1)
    w2 = dy.parameter(self.attention_w2)
    v = dy.parameter(self.attention_v)
    attention_weights = []

    w2dt = w2 * state.h()[-1]
    for input_vector in input_vectors:
        attention_weight = v * dy.tanh(w1 * input_vector + w2dt)
        attention_weights.append(attention_weight)
    attention_weights = dy.softmax(dy.concatenate(attention_weights))

    output_vectors = dy.esum(
        [vector * attention_weight for vector, attention_weight
        in zip(input_vectors, attention_weights)])
    return output_vectors
```

NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Context Vector



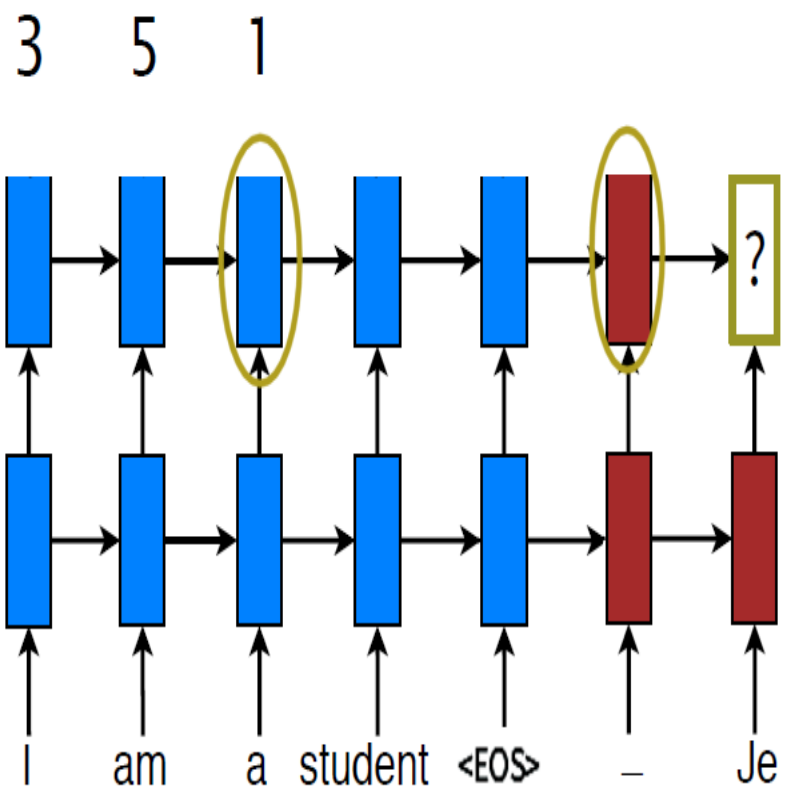

```
def _attend(self, input_vectors, state):
    w1 = dy.parameter(self.attention_w1)
    w2 = dy.parameter(self.attention_w2)
    v = dy.parameter(self.attention_v)
    attention_weights = []

    w2dt = w2 * state.h()[-1]
    for input_vector in input_vectors:
        attention_weight = v * dy.tanh(w1 * input_vector + w2dt)
        attention_weights.append(attention_weight)
    attention_weights = dy.softmax(dy.concatenate(attention_weights))

    output_vectors = dy.esum(
        [vector * attention_weight for vector, attention_weight
        in zip(input_vectors, attention_weights)])
    return output_vectors
```

NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Context Vector



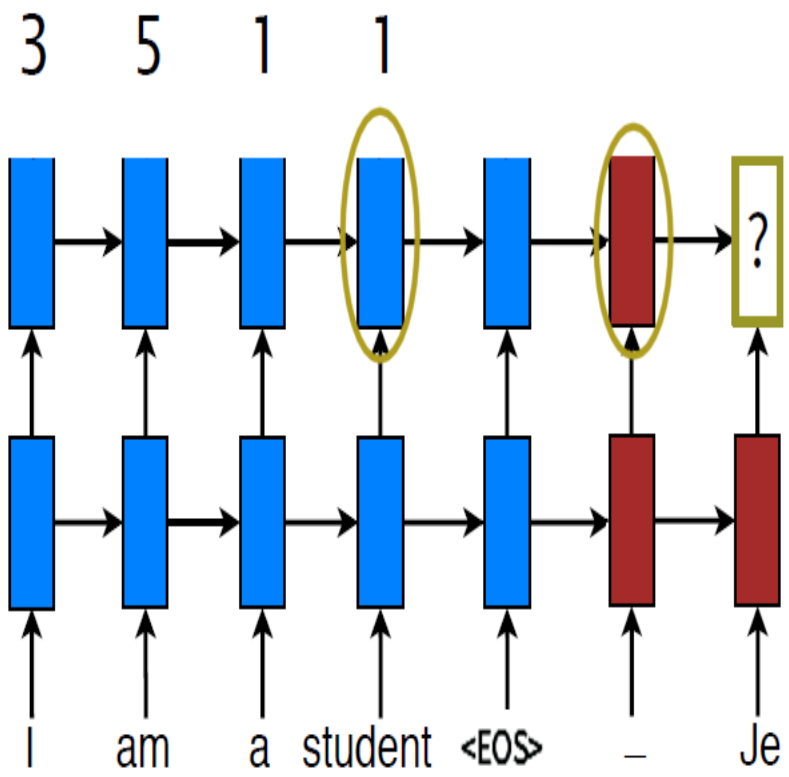

```
def _attend(self, input_vectors, state):
    w1 = dy.parameter(self.attention_w1)
    w2 = dy.parameter(self.attention_w2)
    v = dy.parameter(self.attention_v)
    attention_weights = []

    w2dt = w2 * state.h()[-1]
    for input_vector in input_vectors:
        attention_weight = v * dy.tanh(w1 * input_vector + w2dt)
        attention_weights.append(attention_weight)
    attention_weights = dy.softmax(dy.concatenate(attention_weights))

    output_vectors = dy.esum(
        [vector * attention_weight for vector, attention_weight
        in zip(input_vectors, attention_weights)])
    return output_vectors
```

NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Context Vector



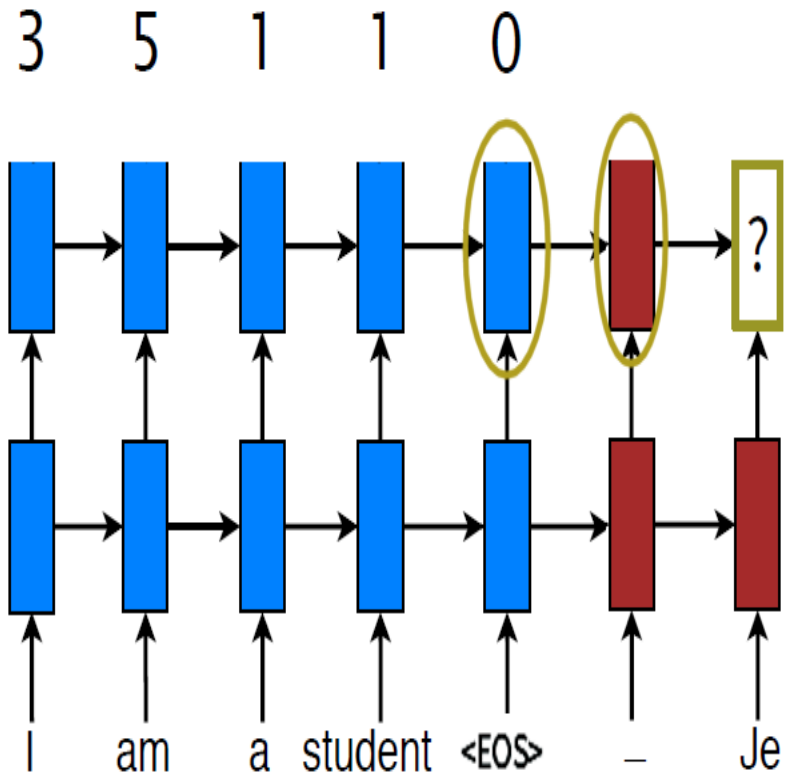

```
def _attend(self, input_vectors, state):
    w1 = dy.parameter(self.attention_w1)
    w2 = dy.parameter(self.attention_w2)
    v = dy.parameter(self.attention_v)
    attention_weights = []

    w2dt = w2 * state.h()[-1]
    for input_vector in input_vectors:
        attention_weight = v * dy.tanh(w1 * input_vector + w2dt)
        attention_weights.append(attention_weight)
    attention_weights = dy.softmax(dy.concatenate(attention_weights))

    output_vectors = dy.esum(
        [vector * attention_weight for vector, attention_weight
        in zip(input_vectors, attention_weights)])
    return output_vectors
```

NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Context Vector



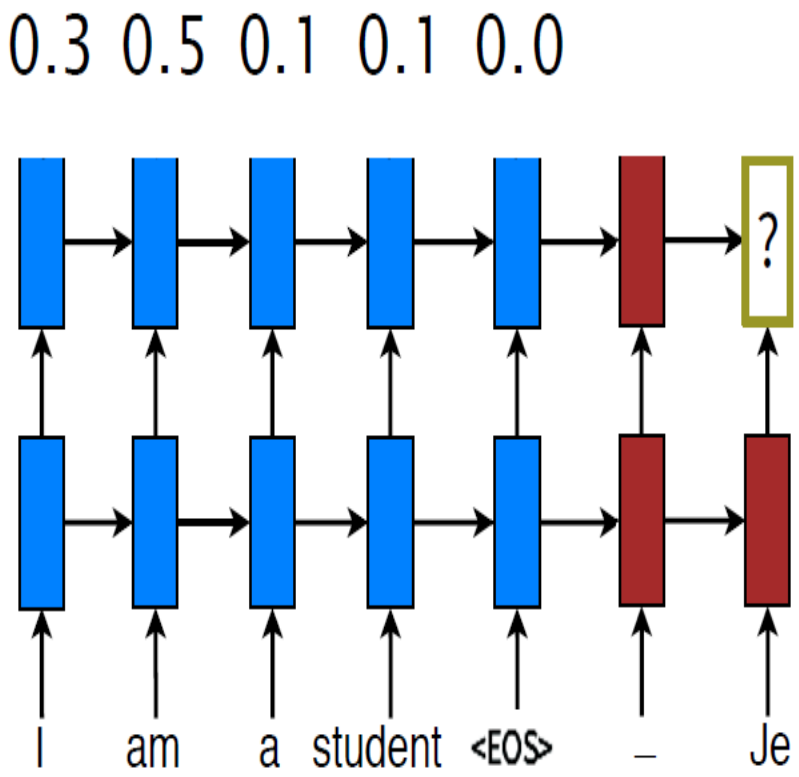

```
def _attend(self, input_vectors, state):
    w1 = dy.parameter(self.attention_w1)
    w2 = dy.parameter(self.attention_w2)
    v = dy.parameter(self.attention_v)
    attention_weights = []

    w2dt = w2 * state.h()[-1]
    for input_vector in input_vectors:
        attention_weight = v * dy.tanh(w1 * input_vector + w2dt)
        attention_weights.append(attention_weight)
    attention_weights = dy.softmax(dy.concatenate(attention_weights))

    output_vectors = dy.esum(
        [vector * attention_weight for vector, attention_weight
        in zip(input_vectors, attention_weights)])
    return output_vectors
```

NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Context Vector




```
def _attend(self, input_vectors, state):
    w1 = dy.parameter(self.attention_w1)
    w2 = dy.parameter(self.attention_w2)
    v = dy.parameter(self.attention_v)
    attention_weights = []

    w2dt = w2 * state.h()[-1]
    for input_vector in input_vectors:
        attention_weight = v * dy.tanh(w1 * input_vector + w2dt)
        attention_weights.append(attention_weight)
    attention_weights = dy.softmax(dy.concatenate(attention_weights))

    output_vectors = dy.esum(
        [vector * attention_weight for vector, attention_weight
        in zip(input_vectors, attention_weights)])
    return output_vectors
```

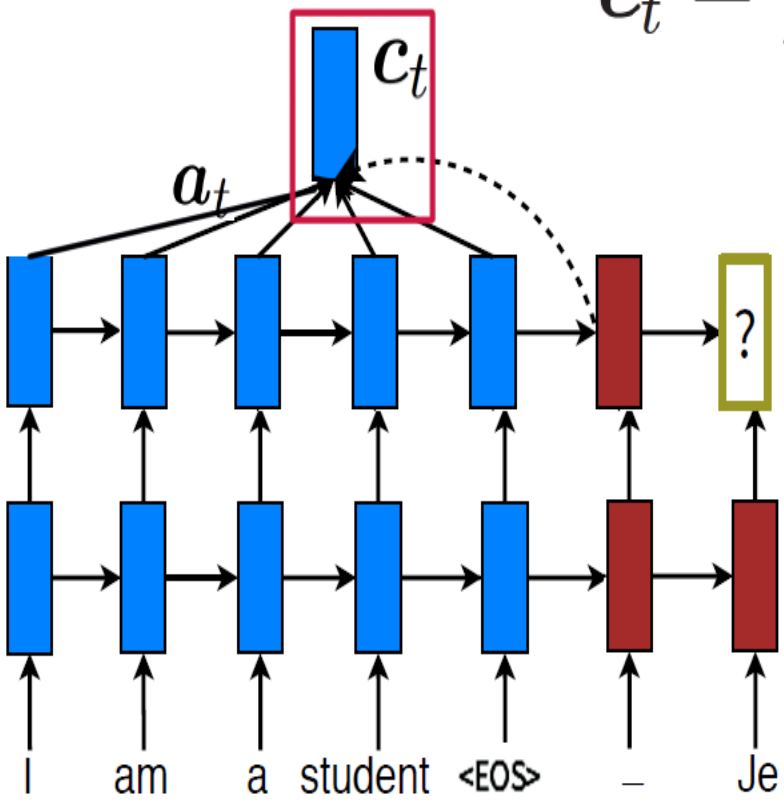
NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Context Vector




```
def _attend(self, input_vectors, state):
    w1 = dy.parameter(self.attention_w1)
    w2 = dy.parameter(self.attention_w2)
    v = dy.parameter(self.attention_v)
    attention_weights = []

    w2dt = w2 * state.h()[-1]
    for input_vector in input_vectors:
        attention_weight = v * dy.tanh(w1 * input_vector + w2dt)
        attention_weights.append(attention_weight)
    attention_weights = dy.softmax(dy.concatenate(attention_weights))

    output_vectors = dy.esum(
        [vector * attention_weight for vector, attention_weight
        in zip(input_vectors, attention_weights)])
    return output_vectors
```

NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Context Vector

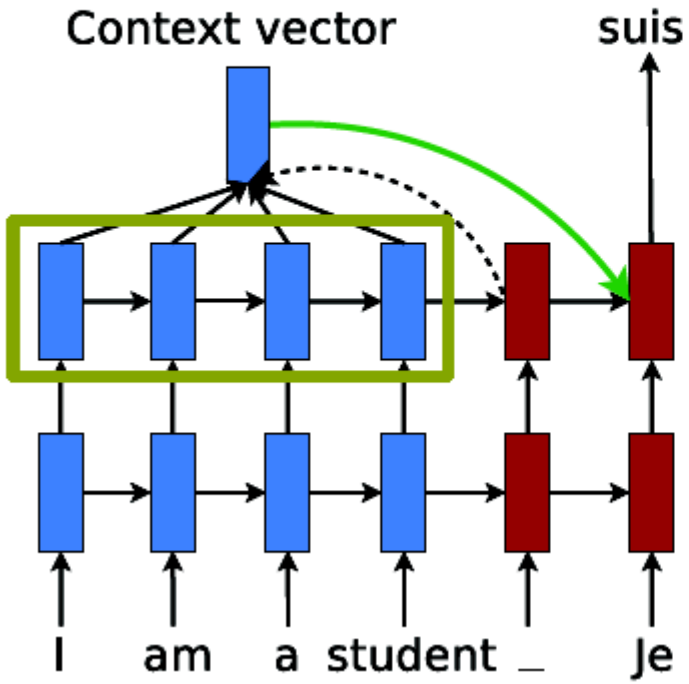
$$c_t = \sum_s a_t(s)\bar{h}_s$$




```
def _attend(self, input_vectors, state):
    w1 = dy.parameter(self.attention_w1)
    w2 = dy.parameter(self.attention_w2)
    v = dy.parameter(self.attention_v)
    attention_weights = []

    w2dt = w2 * state.h()[-1]
    for input_vector in input_vectors:
        attention_weight = v * dy.tanh(w1 * input_vector + w2dt)
        attention_weights.append(attention_weight)
    attention_weights = dy.softmax(dy.concatenate(attention_weights))

    output_vectors = dy.esum(
        [vector * attention_weight for vector, attention_weight
        in zip(input_vectors, attention_weights)])
    return output_vectors
```

NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Context Vector

- Simplified mechanism & more functions:



Bilinear form: well-adopted.

$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s \\ \boldsymbol{h}_t^\top \boldsymbol{W}_a \bar{\boldsymbol{h}}_s \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_a [\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) \end{cases}$$

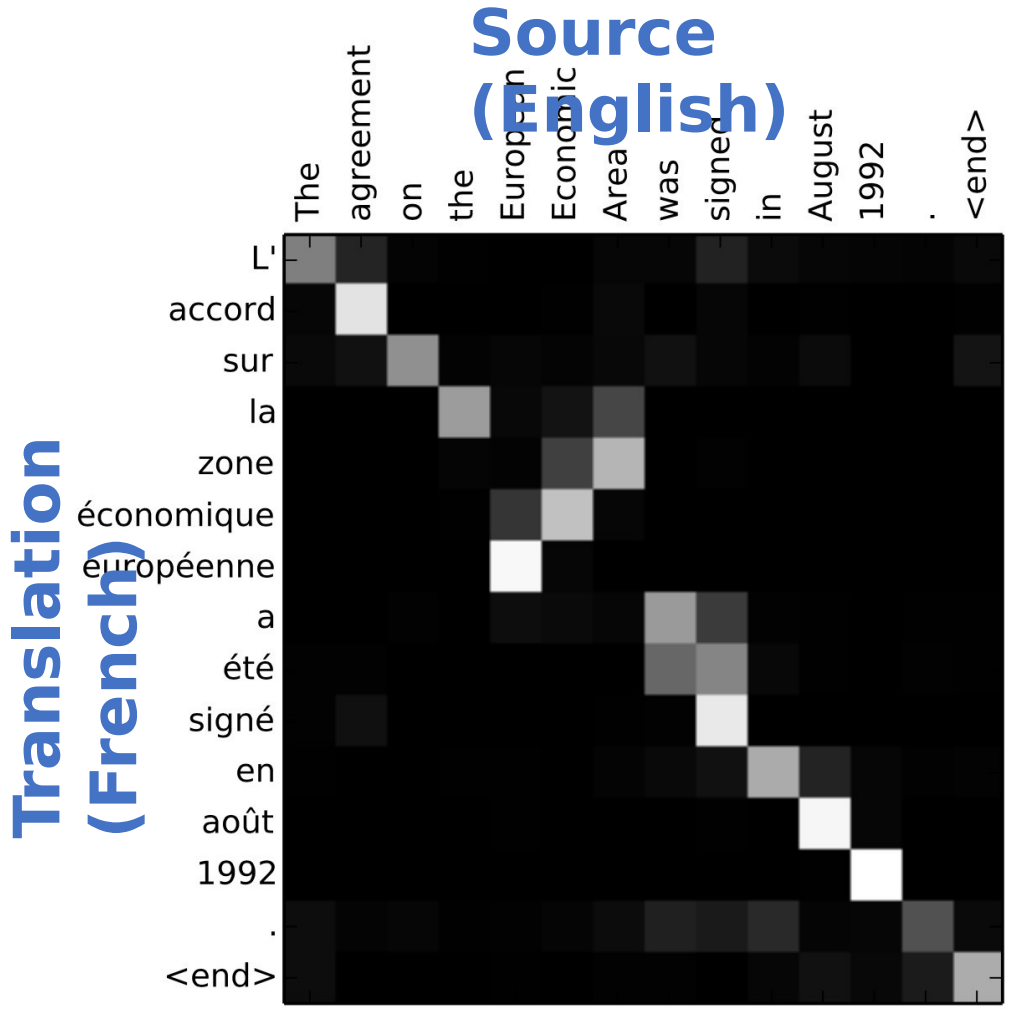
NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Implementation

```
# attention_states: [batch_size, max_time, num_units]
attention_states = tf.transpose(encoder_outputs, [1, 0, 2])

# Create an attention mechanism
attention_mechanism = tf.contrib.seq2seq.LuongAttention(
    num_units, attention_states,
    memory_sequence_length=source_sequence_length)
```

```
decoder_cell = tf.contrib.seq2seq.AttentionWrapper(
    decoder_cell, attention_mechanism,
    attention_layer_size=num_units)
```

- Attention mechanism.

NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Performance



- **Soft Alignment**
  - Heat-map represents soft-alignments
  - Monotonic:
    - Strong weights along the diagonal
  - None-monotonic:
    - Different orders (e.g. for adj. & n.)
    - Phrase of different lengths

D. Bahdanau, K. Cho, Y. Bengio.
Neural Machine Translation by Jointly Learning to Align and Translate. ICLR 2015.

NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Performance

- **Quantitative**: achieve the state-of-the-art from scratch ii

(a) **English→French (WMT-14)**

	NMT(A)	Google	P-SMT
NMT	32.68	30.6*	**37.03**$^\bullet$
+Cand	33.28	–	
+UNK	33.99	32.7°	
+Ens	**36.71**	**36.9**$^\circ$	

(b) **English→German (WMT-15)**

Model	Note
24.8	Neural MT
24.0	U.Edinburgh, Syntactic SMT
23.6	LIMSI/KIT
22.8	U.Edinburgh, Phrase SMT
22.7	KIT, Phrase SMT

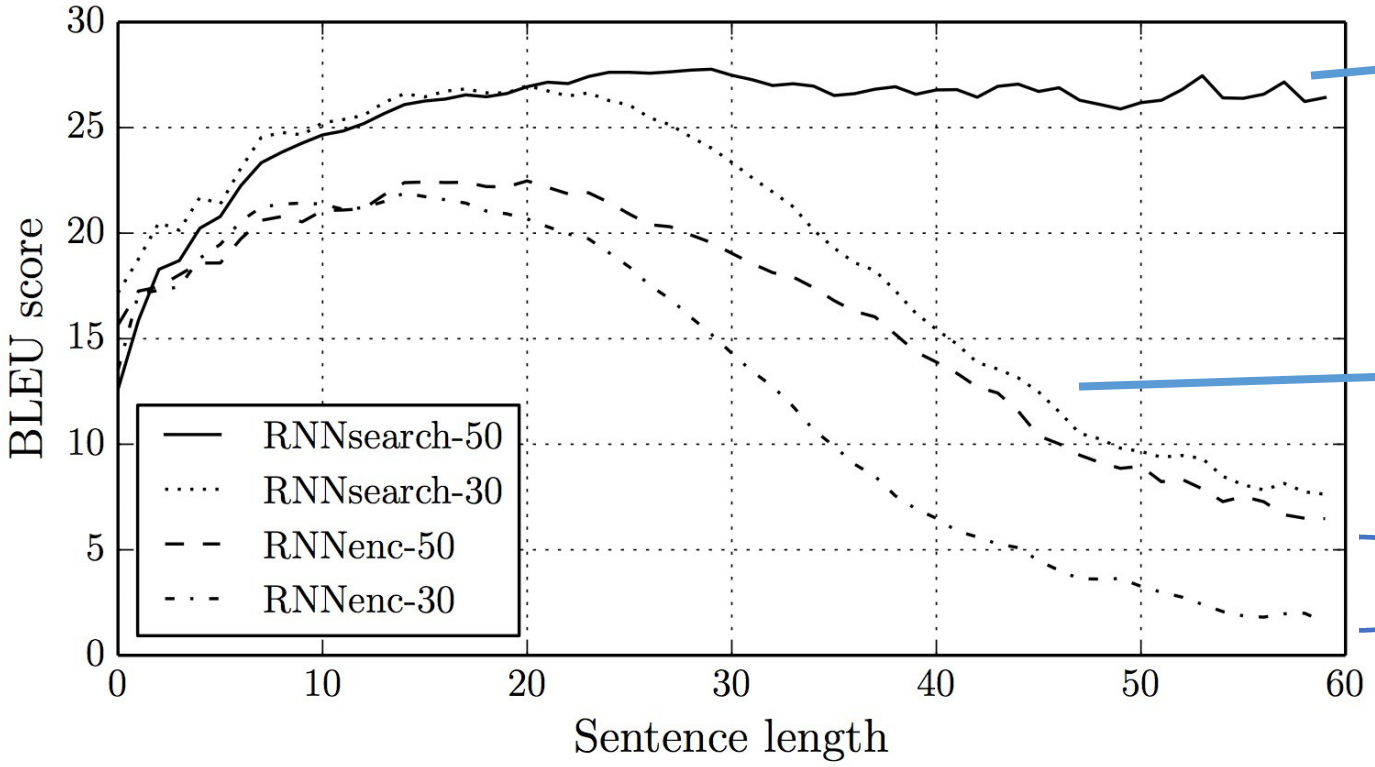
(c) **English→Czech (WMT-15)**

Model	Note
18.3	Neural MT
18.2	JHU, SMT+LM+OSM+Sparse
17.6	CU, Phrase SMT
17.4	U.Edinburgh, Phrase SMT
16.1	U.Edinburgh, Syntactic SMT

D. Bahdanau, K. Cho, Y. Bengio.
Neural Machine Translation by Jointly Learning to Align and Translate. ICLR 2015.

NMT:Encoder-Decoder Architecture:Advanced:Attention Mechanism:Performance

- **“The curse of length” problem**



With attention
(trained with sentence length <= 50)

With attention
(trained with sentence length <= 30)

No attention

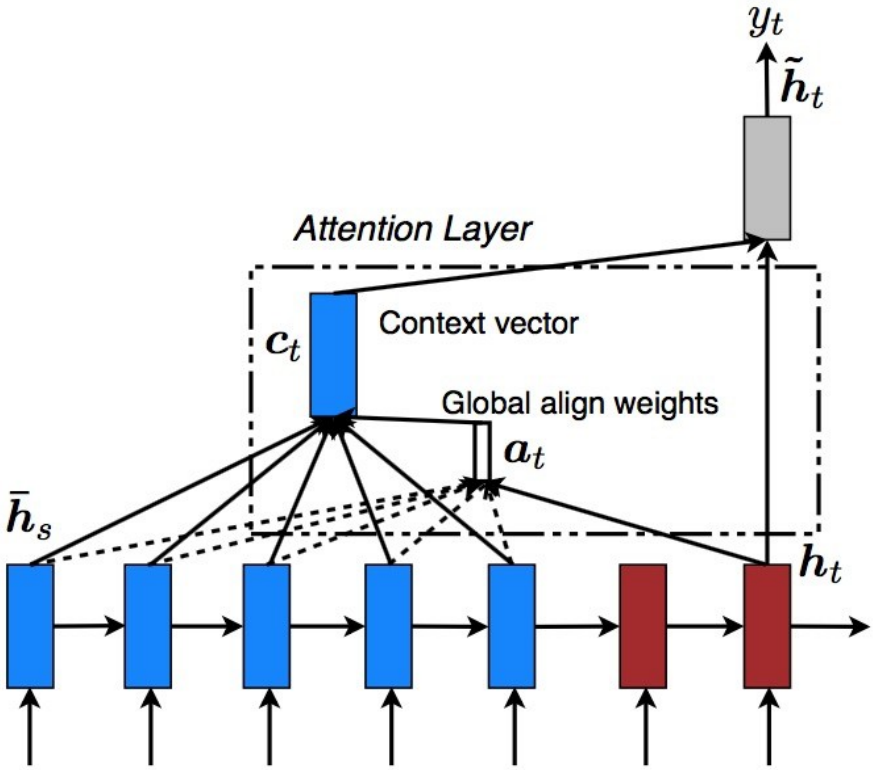
D. Bahdanau, K. Cho, Y. Bengio.
Neural Machine Translation by Jointly Learning to Align and Translate. ICLR 2015.

NMT:Encoder-Decoder Architecture:Advanced:Attention+

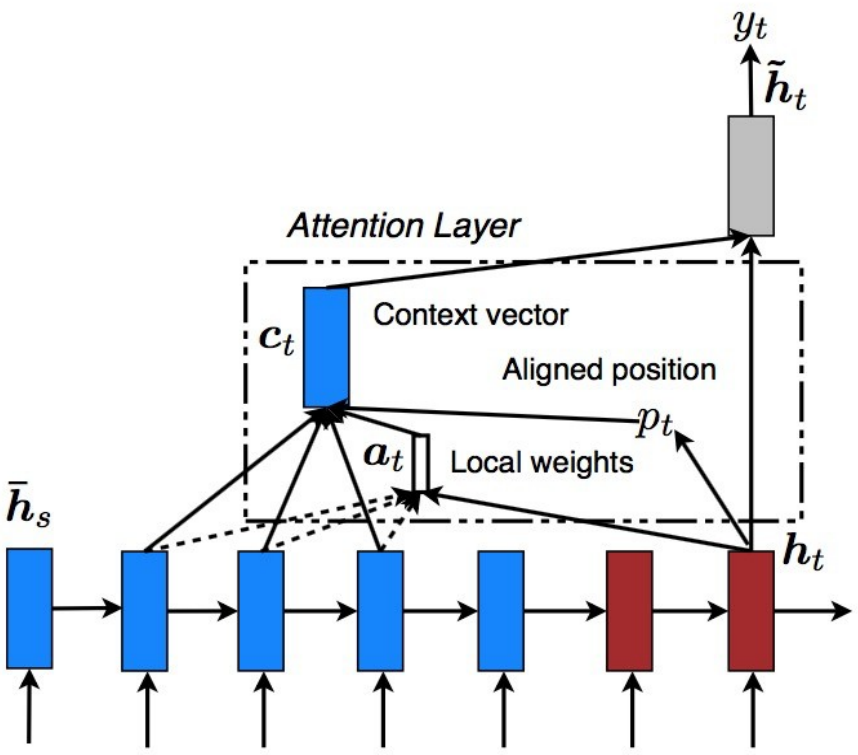
- **Better attention model for NMT tasks?**
  - Global vs local
  - Input-feeding approach
  - Alignment score functions

Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning.
Effective approaches to attention-based neural machine translation. EMNLP 2015

NMT:Encoder-Decoder Architecture:Advanced:Attention+:Global vs Local



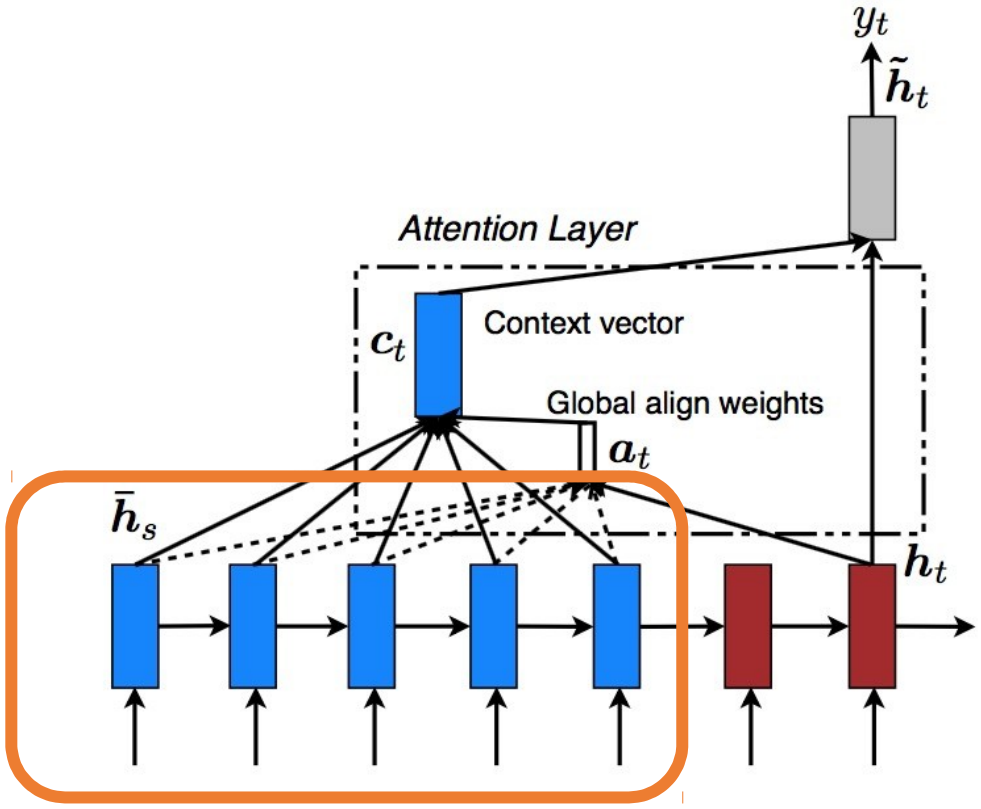
Global



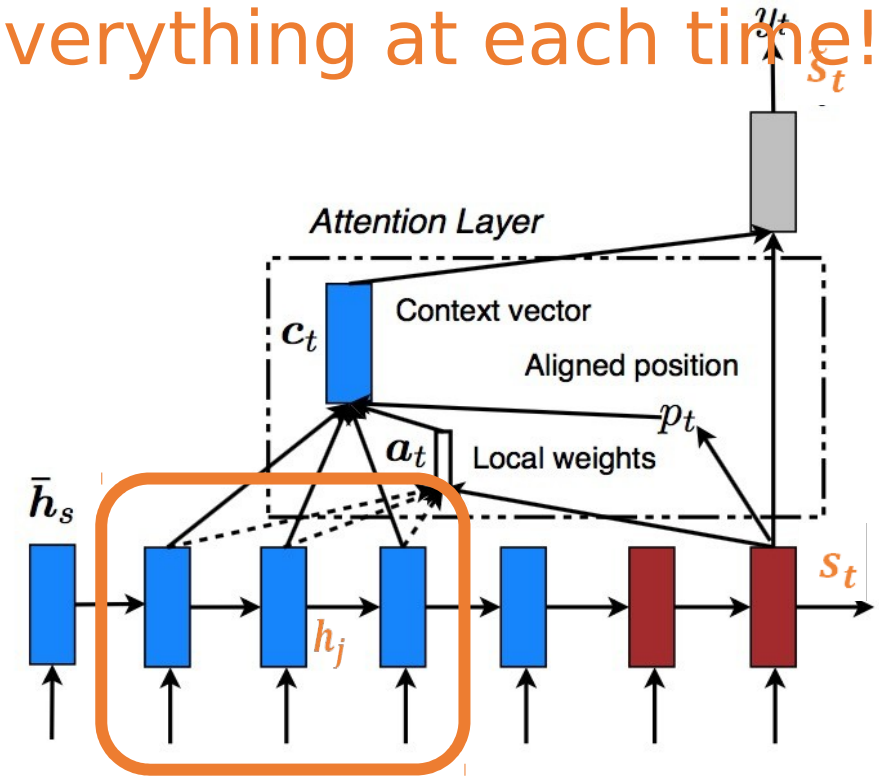
Local

NMT:Encoder-Decoder Architecture:Advanced:Attention+:Global vs Local

Intuition: avoid focus on everything at each time!



Global: use **all** source states

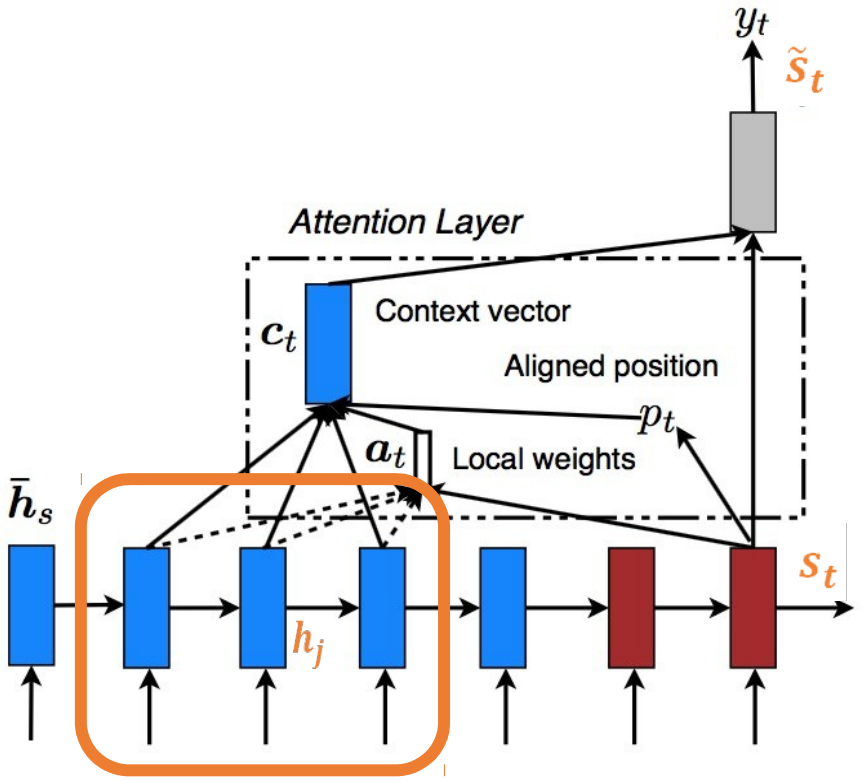


Local: use **subset of** source states

NMT:Encoder-Decoder Architecture:Advanced:Attention+:Local

- **Intuition:**
  - Avoid focus on everything at each time
  - **Tradeoff** between ***soft*** and ***hard*** attenti
- **Recall: soft/hard attention**

	***Soft* attention**	***Hard* attention**
Model	Global attention with weights placed over **all** positions	Select **one** position to attend at a time
Inference	Computationally expensive	Less expensive
Train	Differentiable, jointly trained with other components	Non-differentiable, requires complicated methods to train



Local: use **subset of** source states

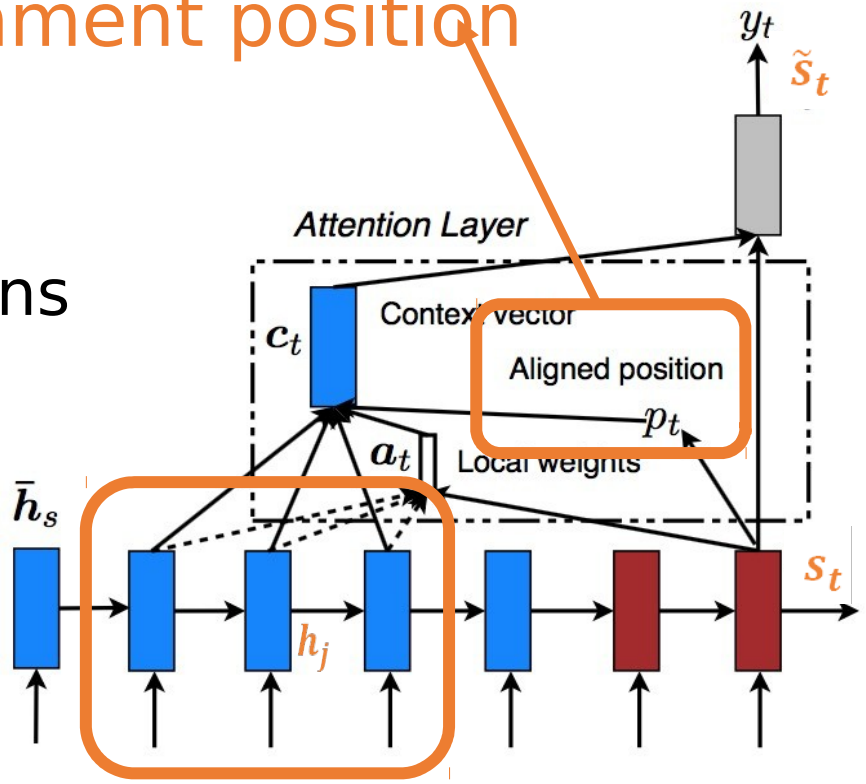
NMT:Encoder-Decoder Architecture:Advanced:Attention+:Local

Problem: find local alignment position

- **Intuition:**
  - Avoid focus on everything at each time
  - Tradeoff between ***soft*** and ***hard*** attentions

- **Recall: soft/hard attention**

	***Soft* attention**	***Hard* attention**
Model	Global attention with weights placed over **all** positions	Select **one** position to attend at a time
Inference	Computationally expensive	Less expensive
Train	Differentiable, jointly trained with other components	Non-differentiable, requires complicated methods to train



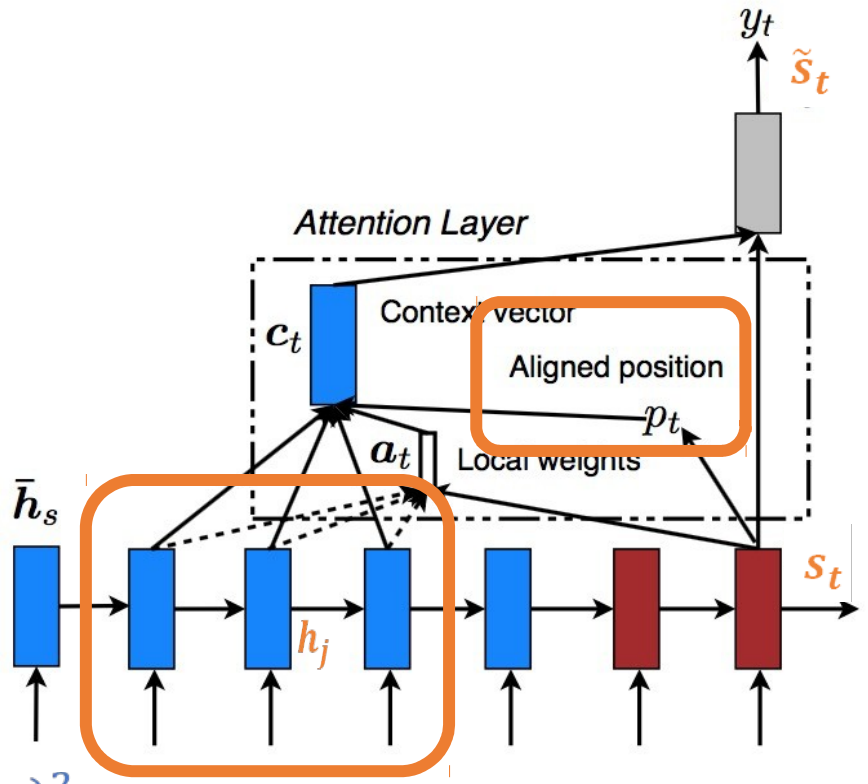
Local: use **subset of** source states

NMT:Encoder-Decoder Architecture:Advanced:Attention+:Local:Alignment

- **Monotonic** (*local-m*):
  - **Position**: $p_t = t$ (Assume monotonically aligned)
  - **Weights**: $\alpha_{tj} = softmax\left(score\left(s_t, h_j\right)\right)$

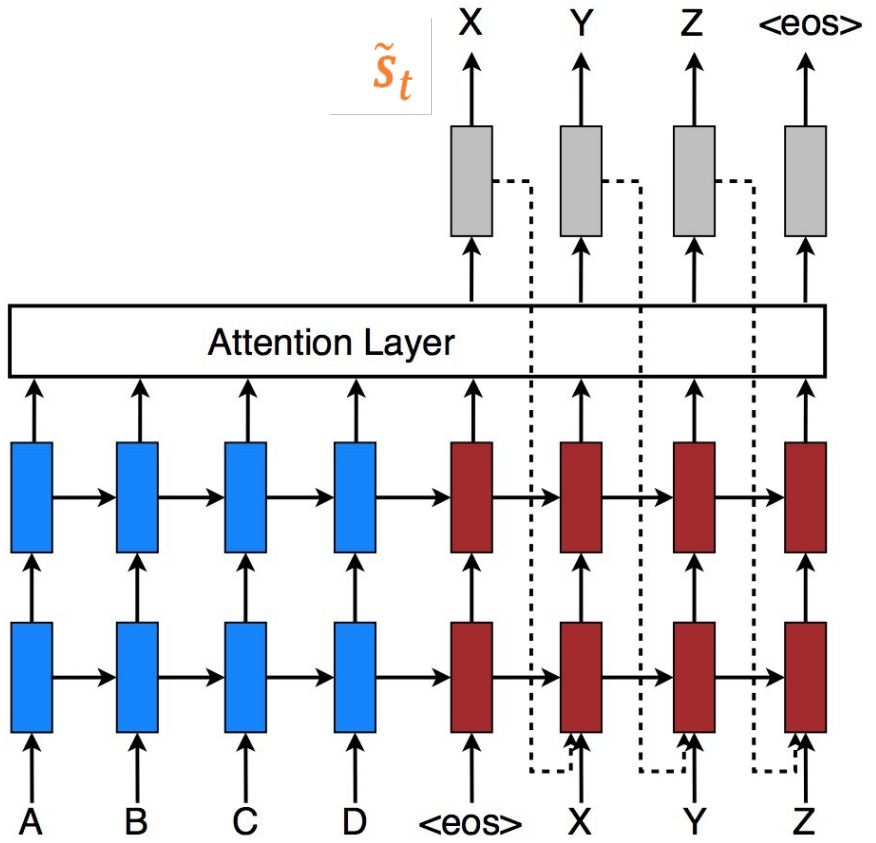
h_j: j[th] hidden state in encoder
s_t: t[th] hidden state in decoder
: t[th] hidden state in decoder

- **Predictive** (*local-p*):
  - **Position**: $p_t = |\vec{s}| \cdot \sigma(v_p^T \tanh(W_p s_t))$
  - **Weights**: $\alpha_{tj} = softmax\left(score\left(s_t, h_j\right)\right) \exp(-\frac{(j-p_t)^2}{2\sigma^2})$
    - Gaussian dist. centered around p_t (favor positions near p_t)



NMT:Encoder-Decoder Architecture:Advanced:Attention+:Local:Input feeding

- **Intuition:**
  - Independent attention is suboptimal
  - Jointly make alignment decisions with past alignment information
- **Approach:**
  - Attentional vector $\tilde{s}_t = \tanh(W_c[c_t; s_t])$
    - Context vector $c_t = \sum_j \alpha_{tj} h_j$
    - Decoder hidden state s_t
  - $\tilde{s}_t$ is also fed as input to the next time steps in decoder-RNN
    - "Inform" model about past alignment information



NMT:Encoder-Decoder Architecture:Advanced:Scoring Methods

- **Content-based** scoring functions:
  -

$$\text{score}(s_t, h_j) = \begin{cases} s_t^T h_j & \textit{dot} \\ s_t^T W_a h_j & \textit{general} \\ v_a^\top \tanh\left(W_a [s_t; h_j]\right) & \textit{concat} \end{cases}$$

h_j: encoder hidden states at position j

s_t: decoder hidden states at position t

: decoder hidden states at position

$$a_t = \text{softmax}(W_a s_t) \qquad \textit{location}$$

- **Location-based** scoring functions:

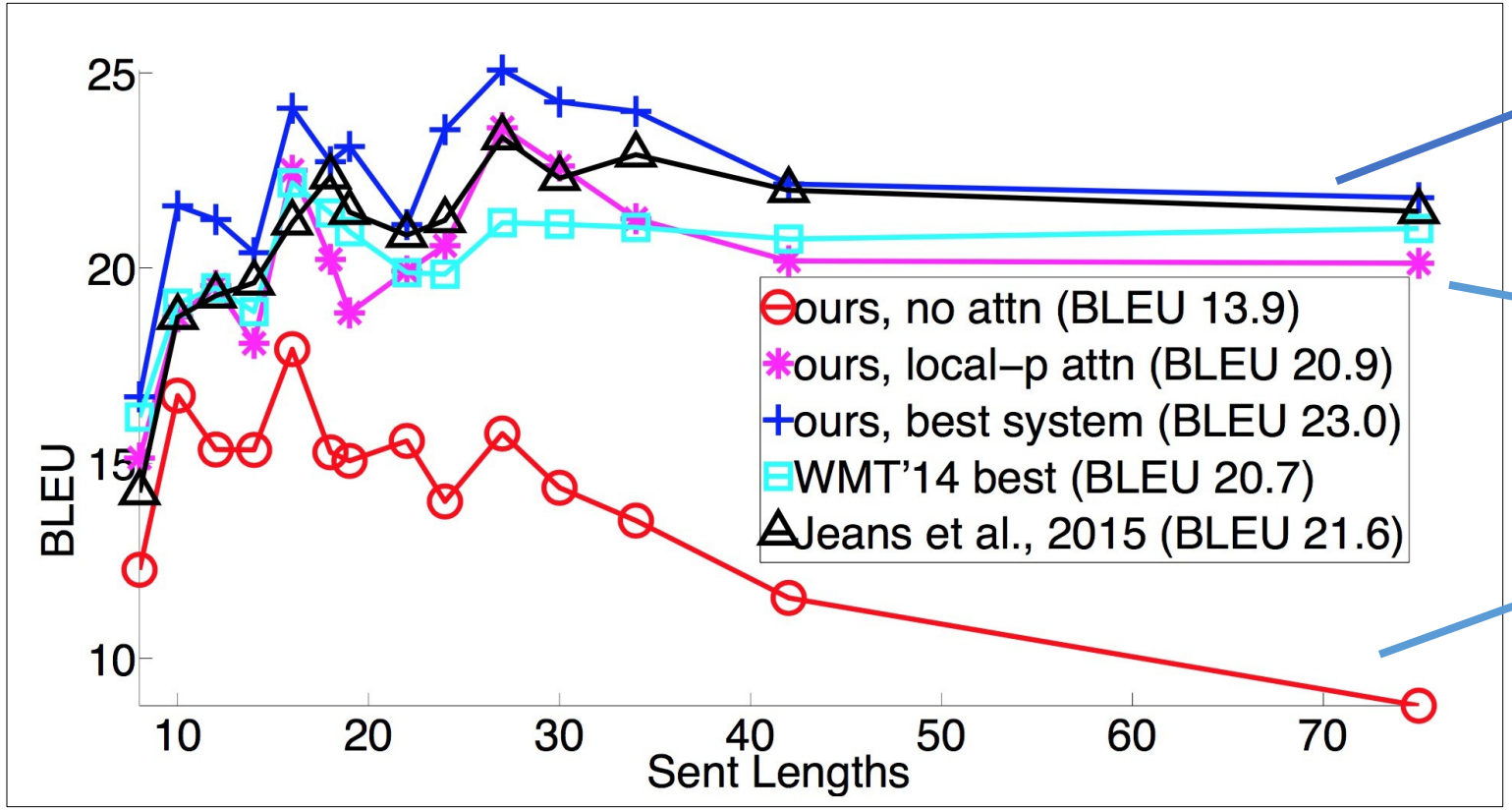
NMT:Encoder-Decoder Architecture:Performance

System	Ppl	BLEU	
		Before	After unk
global (location)	6.4	18.1	19.3 (+1.2)
global (dot)	6.1	18.6	20.5 (+1.9)
global (general)	6.1	17.3	19.1 (+1.8)
local-m (dot)	>7.0	x	x
local-m (general)	6.2	18.6	20.4 (+1.8)
local-p (dot)	6.6	18.0	19.6 (+1.9)
local-p (general)	**5.9**	**19**	**20.9 (+1.9)**

- ***Location-based*** function does NOT learn good alignments
- ***Content-based*** functions:
  - *Concat* does NOT perform well
  - *Dot* works well for *Global* attention
  - *General* works well for *Local* attention

NMT:Encoder-Decoder Architecture:Performance

- Better translation quality for long sentences



Attention
(Ensemble model)

Local attention
(with predictive alignments)

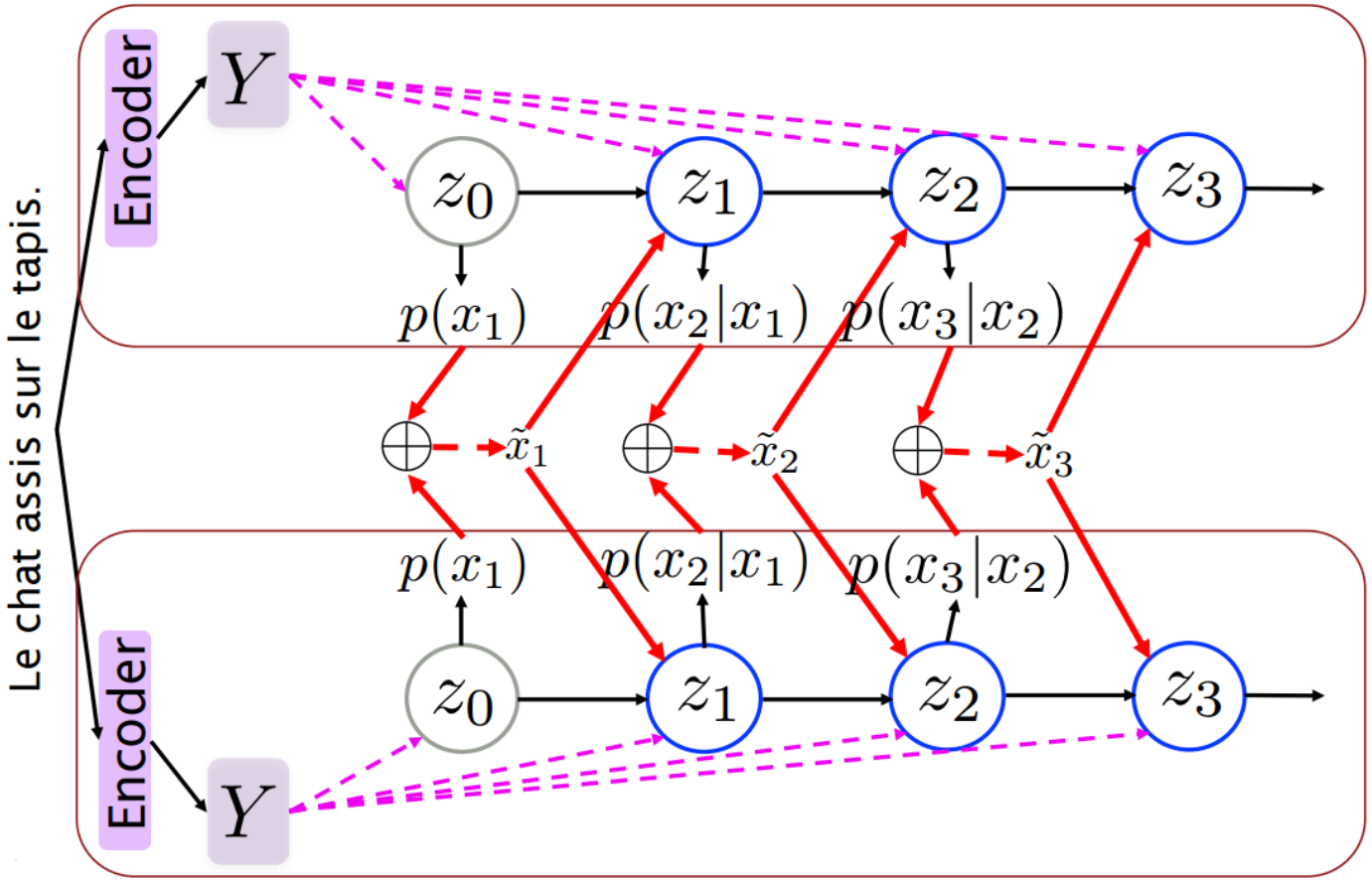
No attention

NMT:Encoder-Decoder Architecture:How to better performance?

- The **attention** aspect
  - Global and local information
- The **vocabulary** aspect
  - **Ensemble decoding**
  - Rare word translations
- The **data** aspect
  - Monolingual data
  - Multilingual data

NMT:Encoder-Decoder Architecture:How to better performance?Ensemble of Conditional LMs

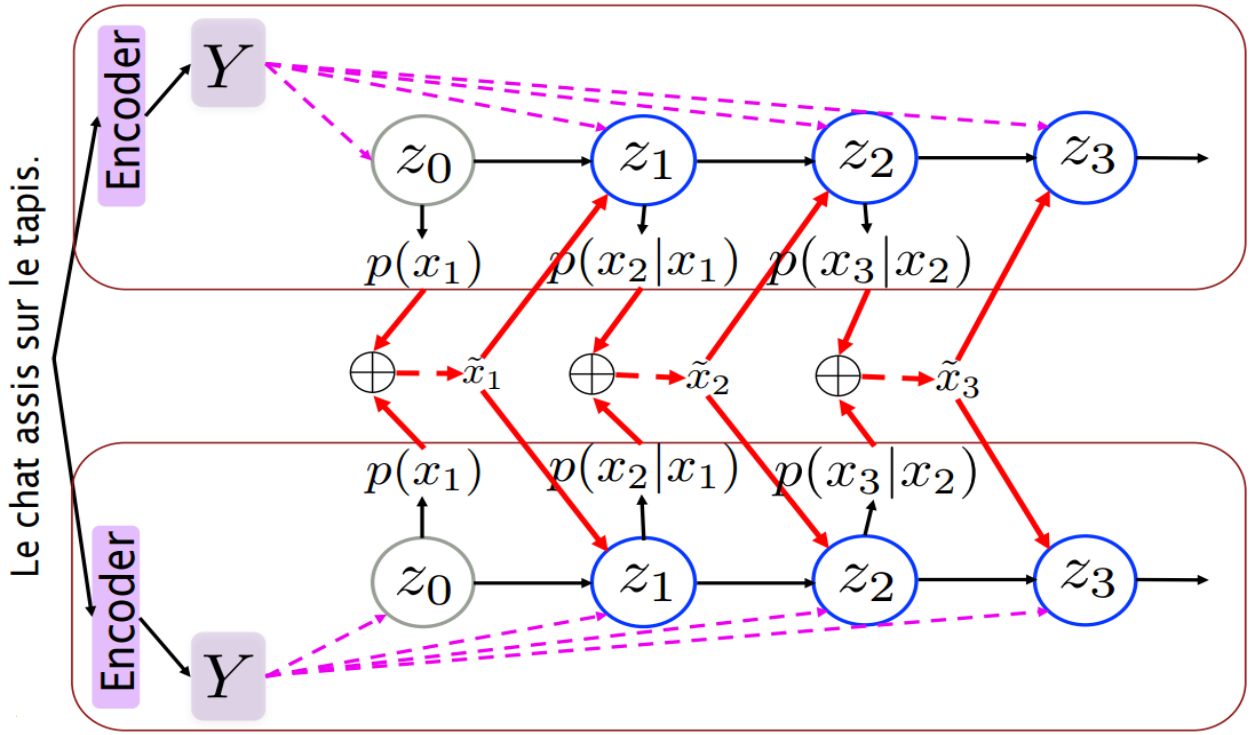
- Training different models from different points of performance
- Fix word embeddings and continue training



Jean, Sébastien et al. "**On Using Very Large Target Vocabulary for Neural Machine Translation**." *ACL* (2015).

NMT:Encoder-Decoder Architecture:How to better performance?Ensemble of Conditional LMs

- At each step, choose the word/phrase with highest conditional ensemble probability
- Ensemble operator $\oplus$ implementations:
  - Majority voting(OR)
  - Consensus building (



NMT:Encoder-Decoder Architecture:Performance

	RNNsearch	RNNsearch-LV	Google	Phrase-based SMT	
Basic NMT	29.97 (26.58)	32.68 (28.76)	30.6*		
+Candidate List	–	33.36 (29.32)	–		
+UNK Replace	33.08 (29.08)	34.11 (29.98)	33.1°	33.3*	37.03$^\bullet$
+Reshuffle (τ=50k)	–	34.60 (30.53)	–		
+Ensemble	–	37.19 (31.98)	37.5°		

(a) English→French

	RNNsearch	RNNsearch-LV	Phrase-based SMT
Basic NMT	16.46 (17.13)	16.95 (17.85)	
+Candidate List	–	17.46 (18.00)	
+UNK Replace	18.97 (19.16)	18.89 (19.03)	20.67$^\diamond$
+Reshuffle	–	19.40 (19.37)	
+Ensemble	–	21.59 (21.06)	

(b) English→German

Ensemble Model outperforms both translation tasks

Jean, Sébastien et al. "On Using Very Large Target Vocabulary for Neural Machine Translation." *ACL* (2015).