Speech Technology: Frontiers and Applications

Sequence-to-Sequence based ASR

Xiangang Li, Guoguo Chen

Outline

1 Sequence modeling for speech recognition

2 Attention/Transformer based speech recognition

- 2.1 Sequence-to-sequence
- 2.2 Listen-Attend-Spell (LAS)
- 2.3 Speech-Transformer

3 Unsupervised pretraining

- 3.1 Pretraining in NLP
- 3.2 Pretraining in ASR

4 Homework

Outline

1 Sequence modeling for speech recognition
2 Attention/Transformer based speech recognition
    2.1 Sequence-to-sequence
    2.2 Listen-Attend-Spell (LAS)
    2.3 Speech-Transformer
3 Unsupervised pretraining
    3.1 Pretraining in NLP
    3.2 Pretraining in ASR
4 Homework

1 Sequence modeling for ASR

- Sequence-to-sequence learning: both input and output are sequences with different lengths
- Sequence-to-sequence, aka seq2seq

Diagram: Jay Alammar. https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

1 Sequence modeling for ASR

- Sequence-to-sequence learning: both input and output are sequences with different lengths
- Typical sequence modeling tasks

ASR

MT

OCR

1 Sequence modeling for ASR

- ASR vs OCR (Optical Character Recognition)

	input	output
OCR	Printed text or handwritten text line images	Transcripts
ASR	Speech	Transcripts

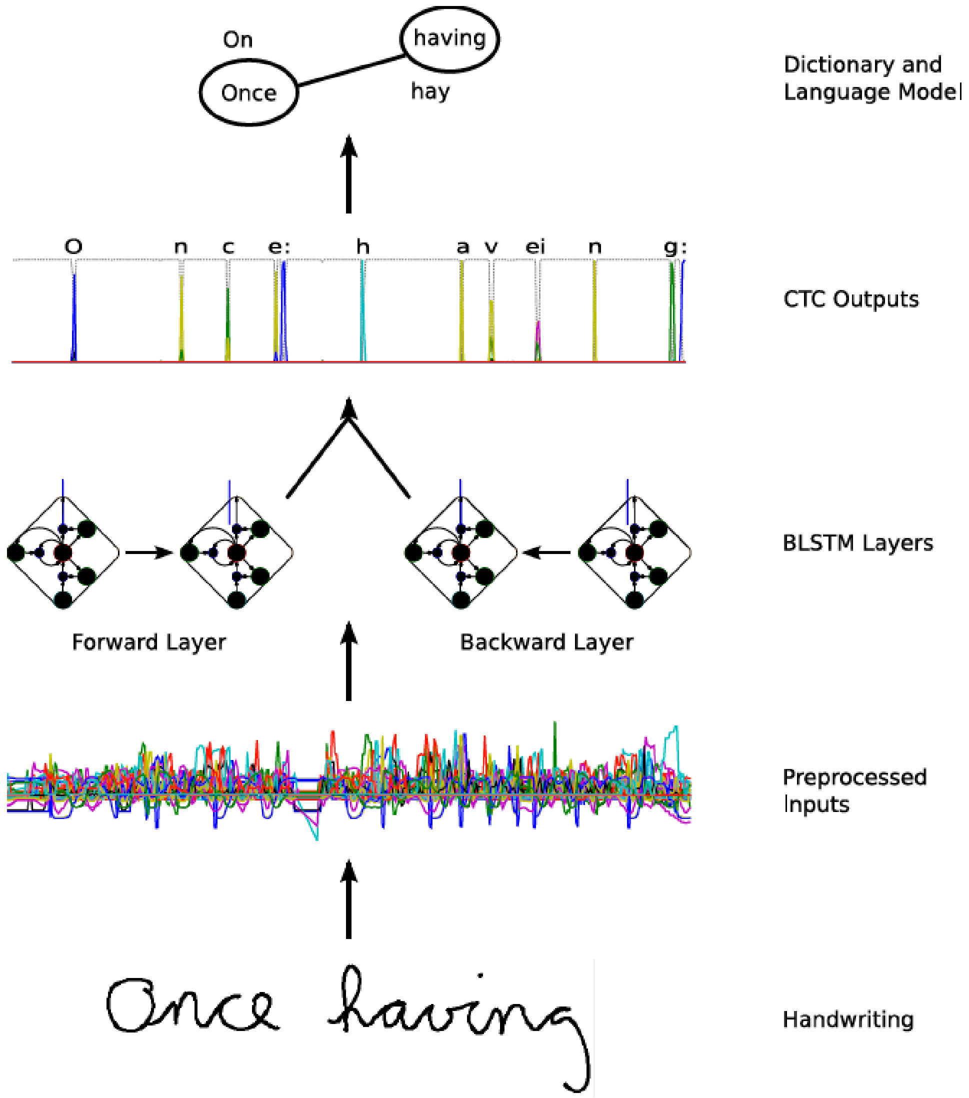
- ASR vs SMT (Statistical Machine Translation)

$$\widehat{W} = \arg\max_{W} p(W|X) = \arg\max_{W} \underbrace{p(X|W)}_{\text{Acoustic Model}} \underbrace{p(W)}_{\text{Language Model}}$$

$$\hat{Y} = \arg\max_{Y} p(Y|X) = \arg\max_{Y} \underbrace{p(X|Y)}_{\text{Translation Model}} \underbrace{p(Y)}_{\text{Language Model}}$$

1 Sequence modeling for ASR

- CTC for ASR & OCR:
  - A Graves, M Liwicki, S Fernández, et.al. A Novel Connectionist System for Unconstrained Handwriting Recognition. PAMI. 2009



1 Sequence modeling for ASR

- Seq2Seq for ASR & SMT
  - Same structure with Bahdanau's neural translation model

$$\alpha_{ts} = \frac{\exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)} \qquad \text{[Attention weights]} \qquad (1)$$

$$\boldsymbol{c}_t = \sum_s \alpha_{ts} \bar{\boldsymbol{h}}_s \qquad \text{[Context vector]} \qquad (2)$$

$$\boldsymbol{a}_t = f(\boldsymbol{c}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{W}_c[\boldsymbol{c}_t; \boldsymbol{h}_t]) \qquad \text{[Attention vector]} \qquad (3)$$

$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \boldsymbol{W} \bar{\boldsymbol{h}}_s & \text{[Luong's multiplicative style]} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_1 \boldsymbol{h}_t + \boldsymbol{W}_2 \bar{\boldsymbol{h}}_s\right) & \text{[Bahdanau's additive style]} \end{cases} \qquad (4)$$

h_t ?

End-to-end Continuous Speech Recognition using Attention-based Recurrent NN: First Results

Jan Chorowski
University of Wrocław, Poland
jan.chorowski@ii.uni.wroc.pl

Dzmitry Bahdanau
Jacobs University Bremen, Germany

Kyunghyun Cho
Université de Montréal

Yoshua Bengio
Université de Montréal
CIFAR Senior Fellow

Abstract

We replace the Hidden Markov Model (HMM) which is traditionally used in in continuous speech recognition with a bi-directional recurrent neural network encoder coupled to a recurrent neural network decoder that directly emits a stream of phonemes. The alignment between the input and output sequences is established using an attention mechanism: the decoder emits each symbol based on a context created with a subset of input symbols selected by the attention mechanism. We report initial results demonstrating that this new approach achieves phoneme error rates that are comparable to the state-of-the-art HMM-based decoders, on the TIMIT dataset.

Outline

深蓝学院
shenlanxueyuan.com

1 Sequence modeling for speech recognition

2 Attention/Transformer based speech recognition

- 2.1 Sequence-to-sequence
- 2.2 Listen-Attend-Spell (LAS)
- 2.3 Speech-Transformer

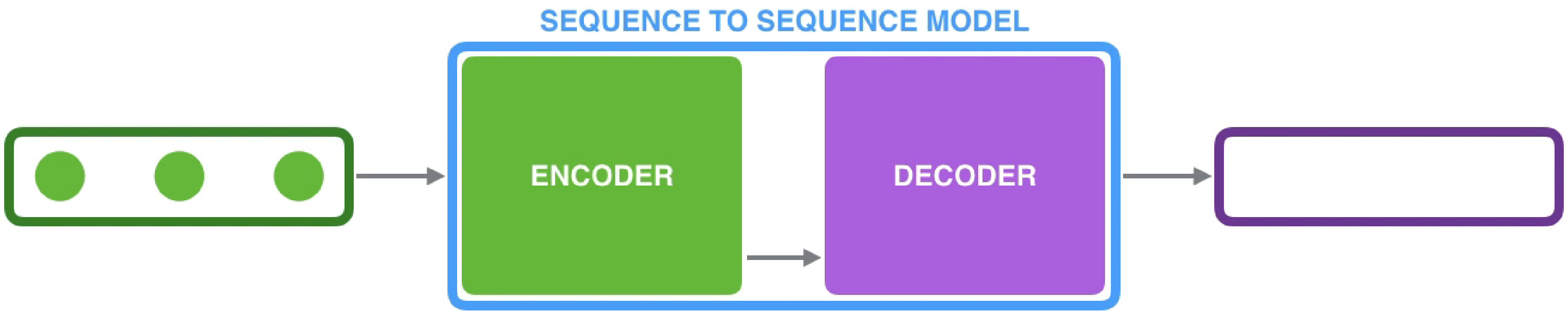
3 Unsupervised pretraining

- 3.1 Pretraining in NLP
- 3.2 Pretraining in ASR

4 Homework

2 Attention/Transformer based ASR

- Sequence-to-sequence learning



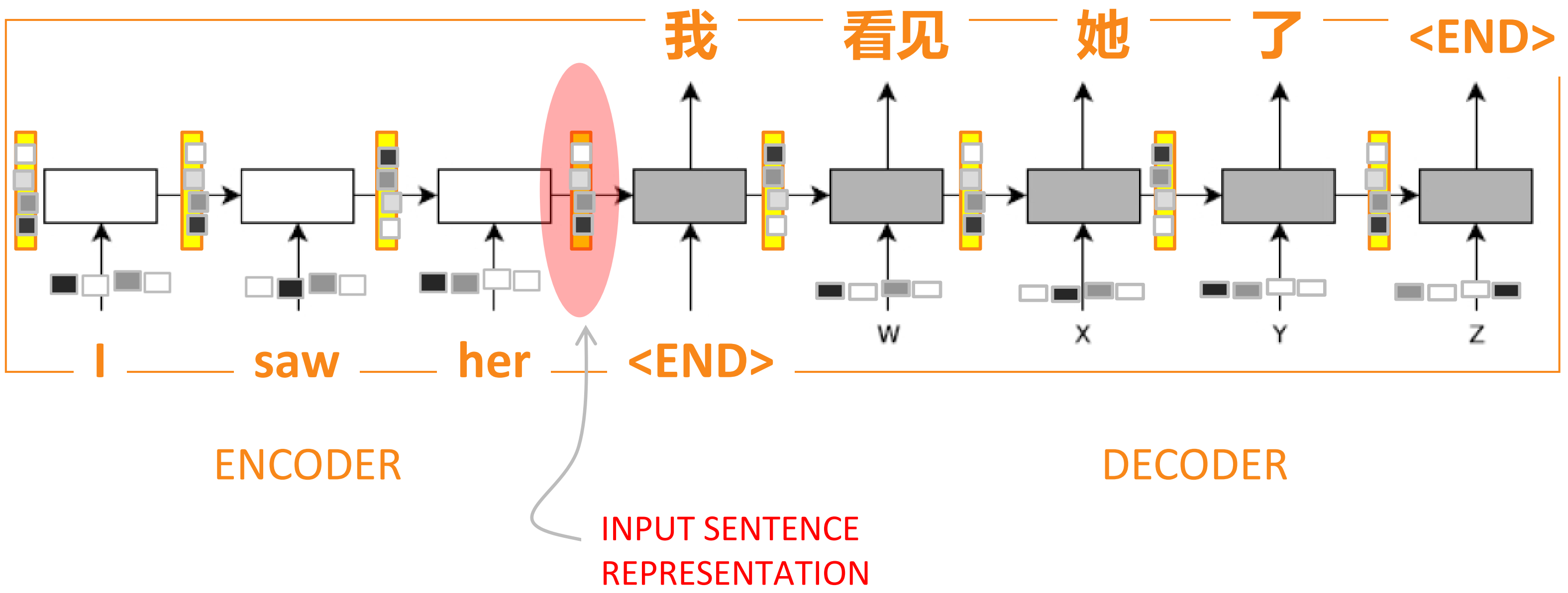
- Three key components： Encoder -> Context -> Decoder

Diagram: Jay Alammar. https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

2.1 Sequence to Sequence RNN

深蓝学院
shenlanxueyuan.com

- Many NLP applications convert one string to another
  - E.g. generating arbitrary-length sequences?
- Sequence-to-sequence, aka seq2seq, aka Encoder-Decoder model:

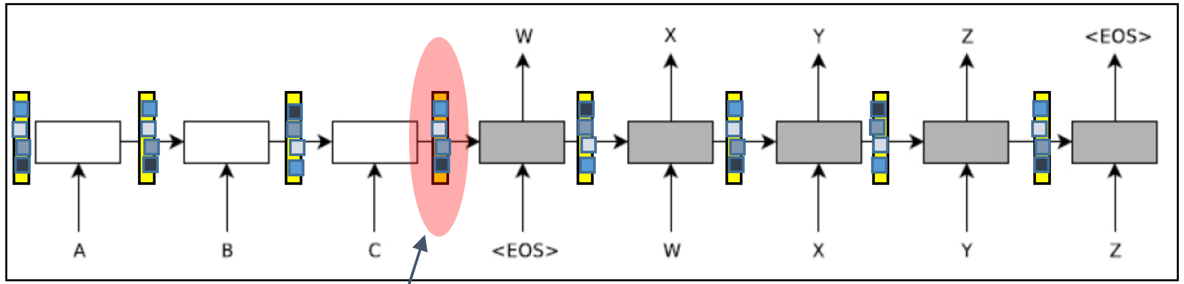


Train on sentence pairs

Maximize P(word | context) for each target-side word.

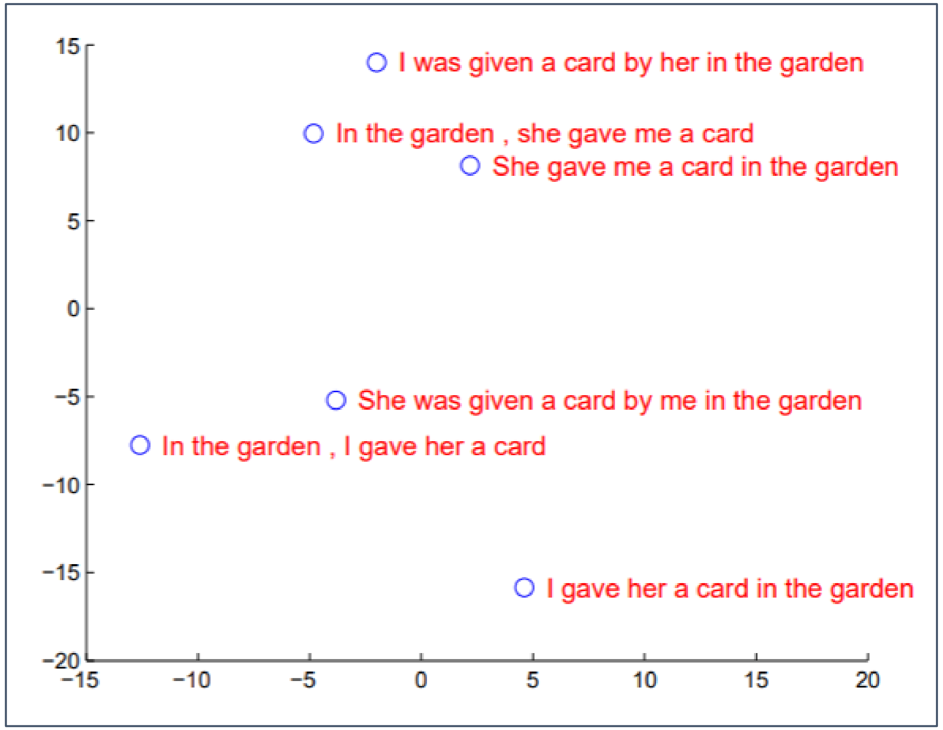
Develop **word vectors**, and also **sentence vectors**.

What's in a Sentence Vector?



INPUT SENTENCE REPRESENTATION

长时间依赖



https://arxiv.org/abs/1409.3215

Sequence-to-sequence with attention

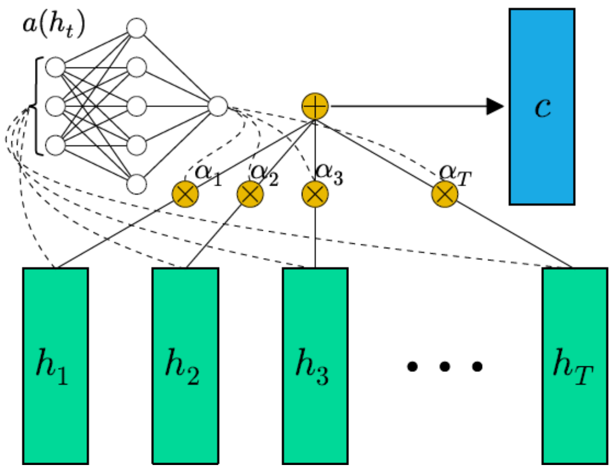
- Sequence-to-sequence learning
  - End-to-end learning
  - Bottleneck problem long-time-dependency
    - Solution: Attention
- General definition of Attention
  - Given a set of vector ***values***, and a vector ***query***, **attention** is a technique to compute a weighted sum of the values, dependent on the query
  - Intuition
    - The weighted sum is a selective summary of the information contained in the values, where the query determines which values to focus on.
    - Attention is a way to obtain a fixed-size representation of an arbitrary set of representations (the values), dependent on some other representation (the query).

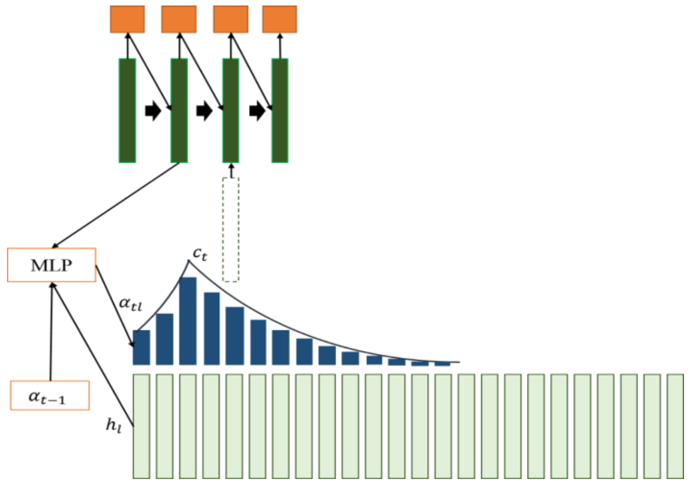
Attention

- Attention always involves:
  - Computing the attention scores $e \in \mathbb{R}^N$
  - Taking softmax to get attention distribution $\alpha = softmax(e) \in \mathbb{R}^N$
  - Using attention distribution to take weighted sum of values:

$$a = \sum_{i=1}^{N} \alpha_i h_i \in \mathbb{R}^N$$

  - Thus obtaining the attention output a (sometimes called the context vector)





Sequence-to-sequence with attention

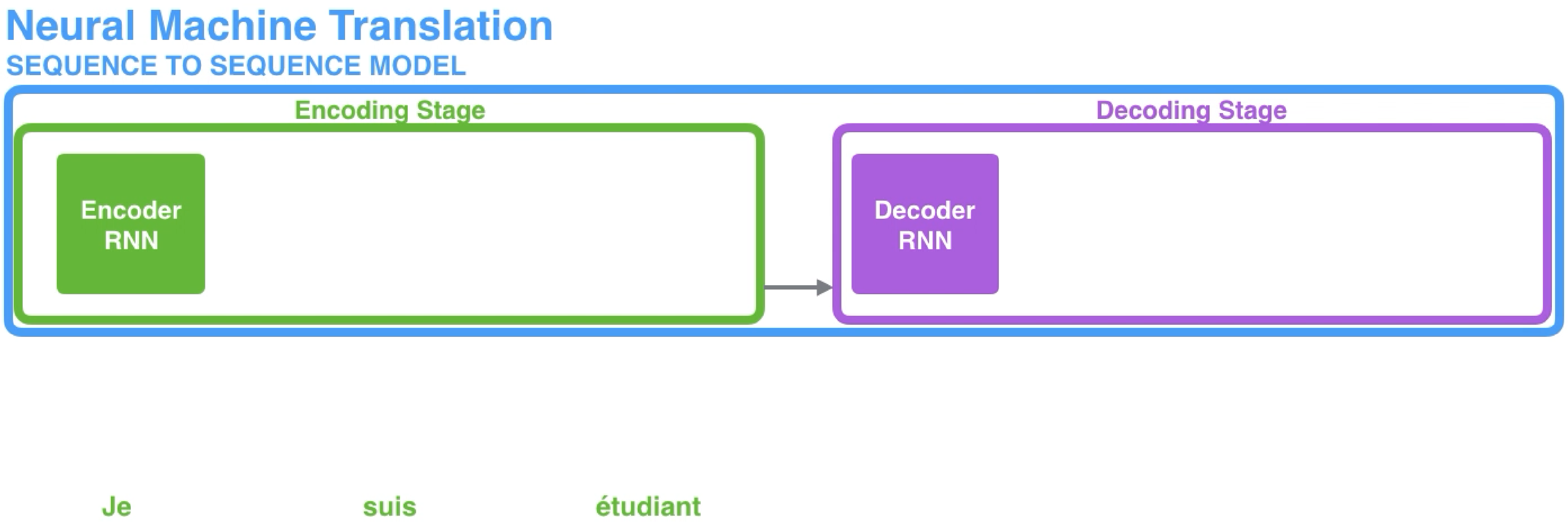


Diagram: Jay Alammar. https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Sequence-to-sequence with attention

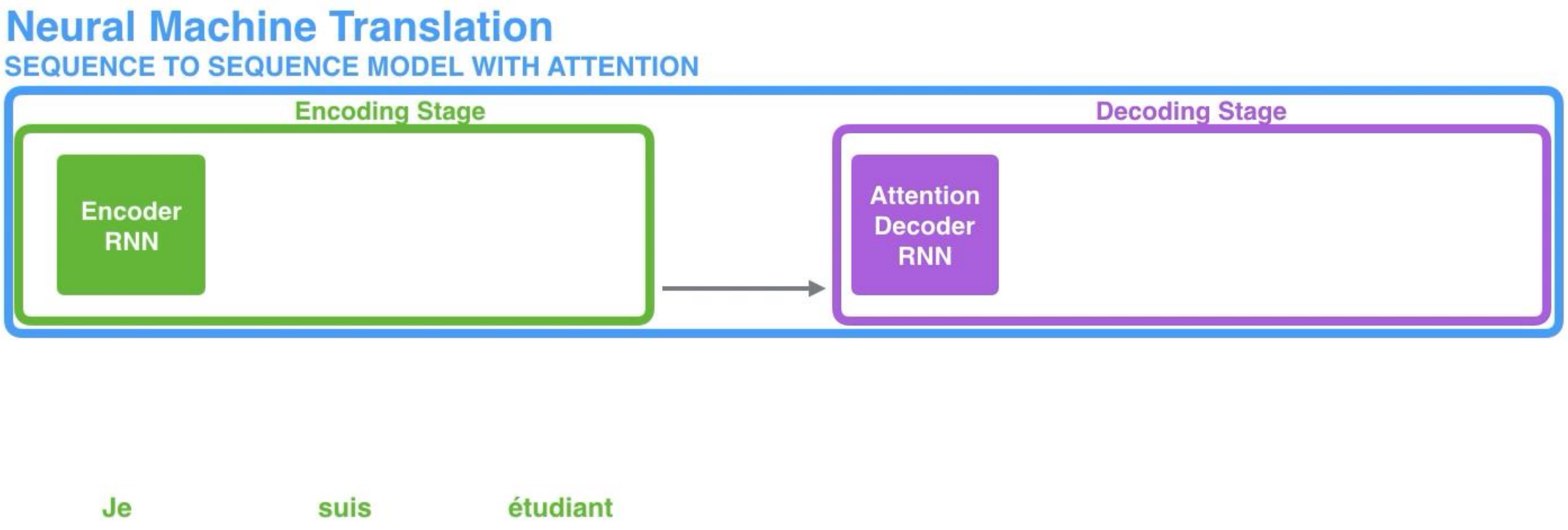


Diagram: Jay Alammar. https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Sequence-to-sequence with attention

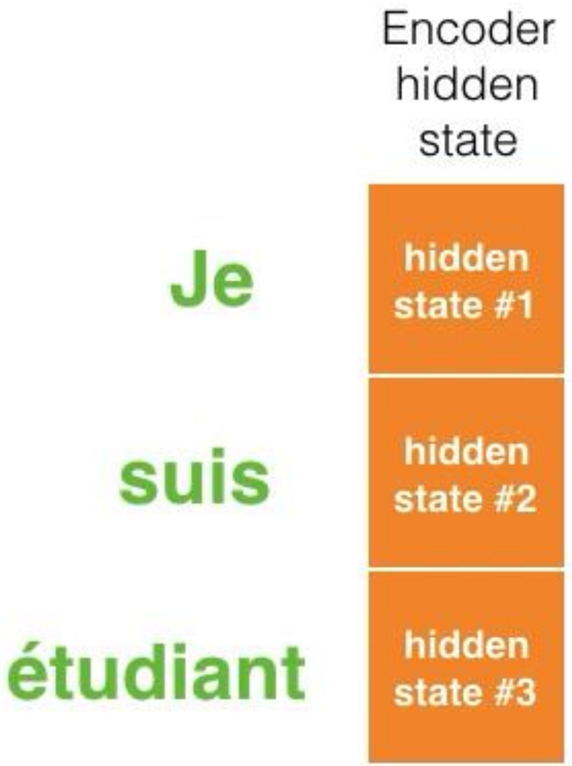


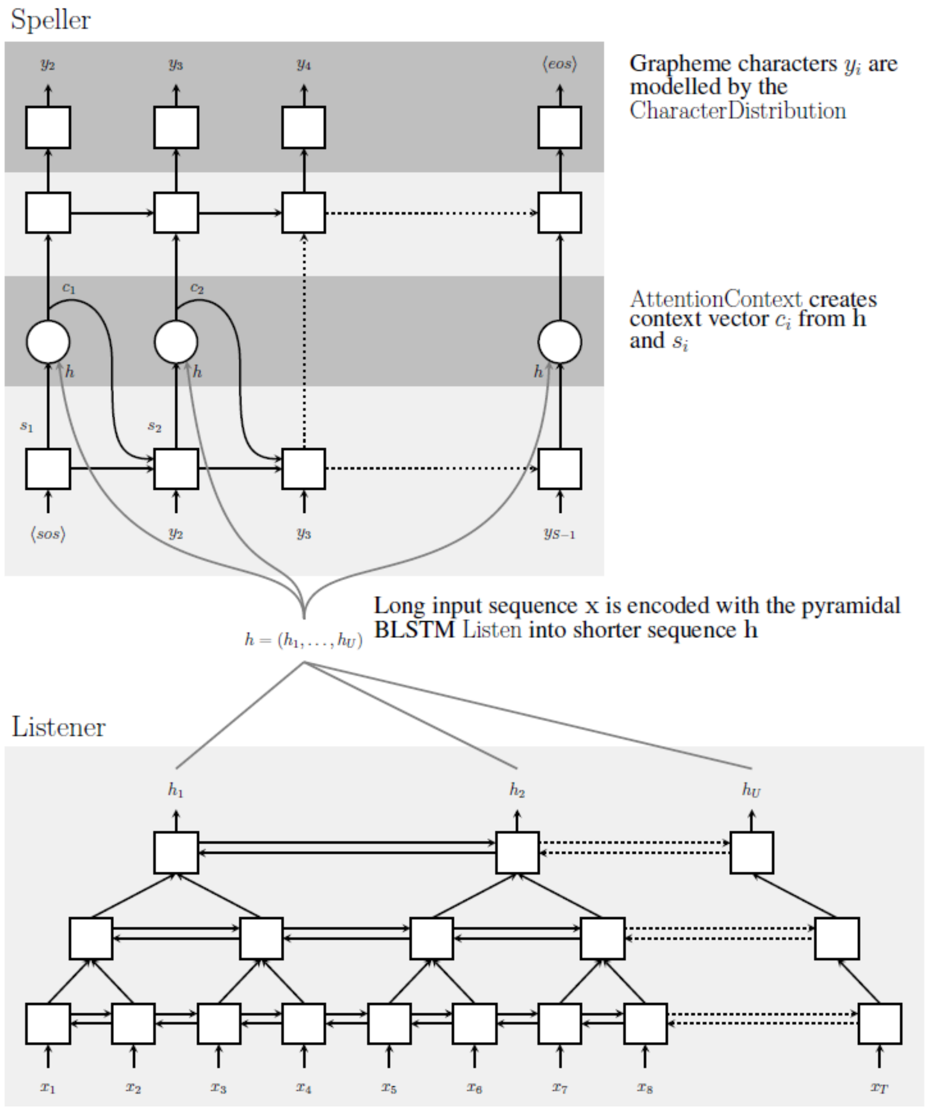
Diagram: Jay Alammar. https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

2.2 Listen-Attend-Spell (LAS)

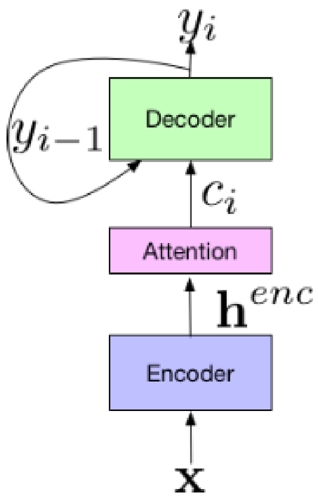
- Apply sequence-to-sequence with attention into ASR

	SMT	ASR
The length difference between inputs and outputs	Nearly the same	The length of input sequence nearly 20 times of the length of output
The input vector	Word vector	Acoustic signal, which is a continuous vector

2.2 Listen-Attend-Spell (LAS)



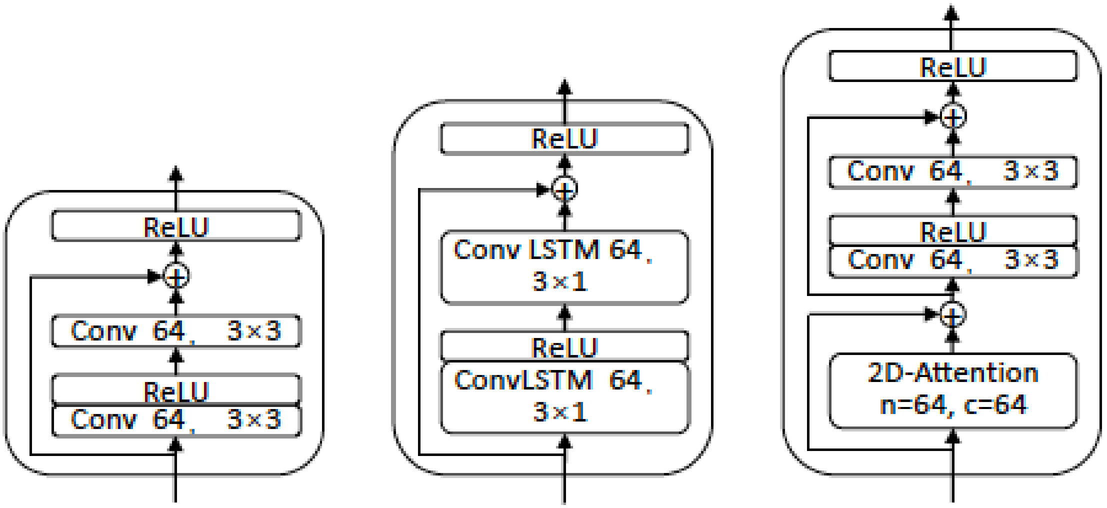
- Encoder
  - Listen, map the input feature sequence to embedding
- Decoder
  - Spell, map the embedding based on the attention information to the output symbols

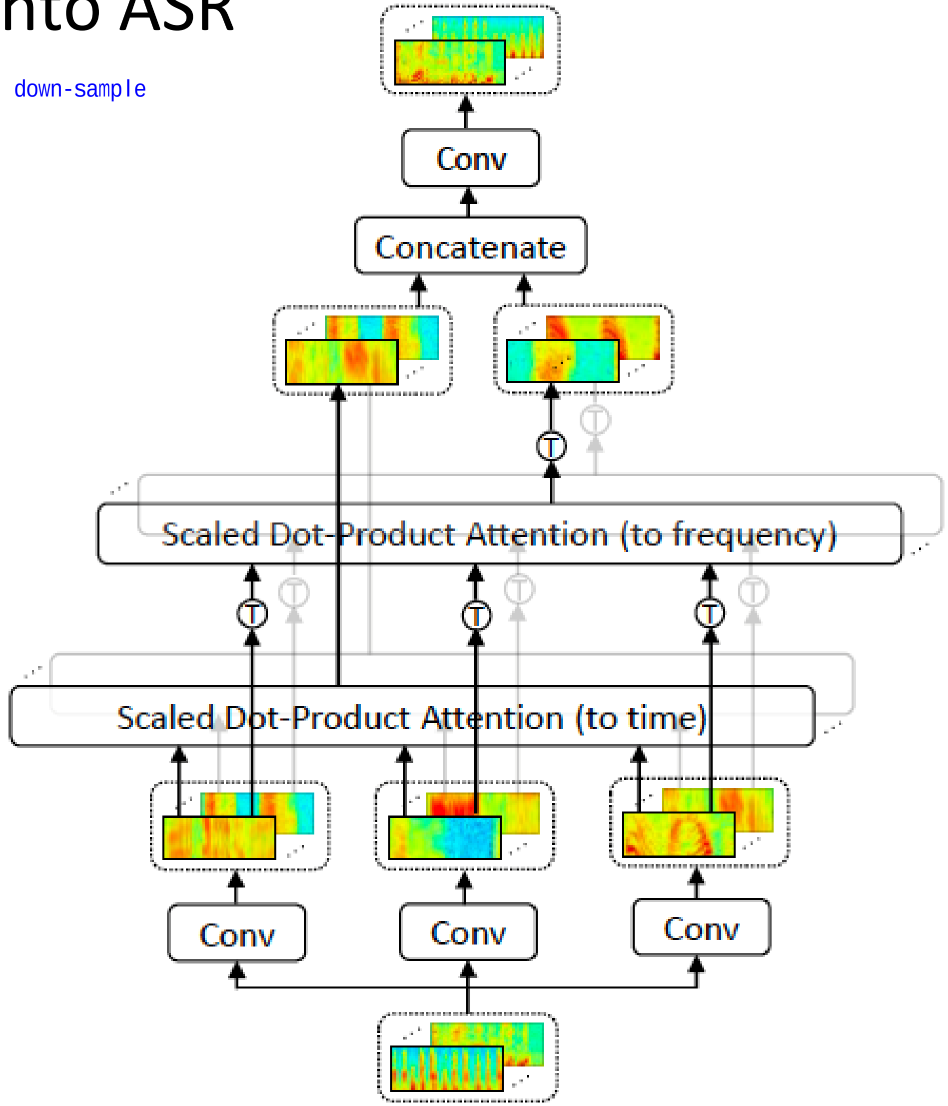


2.2 Listen-Attend-Spell (LAS)

- Apply sequence-to-sequence with attention into ASR
  - Suggested solution: adopting convolution layers

down-sample





A simple implementation of LAS

深蓝学院
shenlanxueyuan.com

- Attention

```python
class BahdanauAttention(tf.keras.Model):
    """ the Bahdanau Attention """

    def __init__(self, units, input_dim=1024):
        super().__init__()
        self.W1 = tf.keras.layers.Dense(units)
        self.W2 = tf.keras.layers.Dense(units)
        self.V = tf.keras.layers.Dense(1)

    def call(self, query, values):
        """ call function """
        hidden_with_time_axis = tf.expand_dims(query, 1)
        score = self.V(tf.nn.tanh(self.W1(values) + self.W2(hidden_with_time_axis)))
        attention_weights = tf.nn.softmax(score, axis=1)
        context_vector = attention_weights * values
        context_vector = tf.reduce_sum(context_vector, axis=1)
      You, 3 months ago • first commit
        return context_vector, attention_weights
```

A simple implementation of LAS

深蓝学院
shenlanxueyuan.com

- AttentionDecoder

```
class AttentionDecoder(tf.keras.layers.Layer):
    """ Used in Encoder-Decoder Models """
    def __init__(self, vocab_size, start, embedding_dim, d_model):
        super().__init__()
        layers = tf.keras.layers
        self.start = start
        self.embedding = layers.Embedding(vocab_size, embedding_dim)
        self.attention = BahdanauAttention(d_model)
        self.rnn = tf.keras.layers.GRU(d_model, return_sequences=True, return_state=True)
        self.dense = layers.Dense(vocab_size)

    def call(self, values, query, y, training=None):
        """ Take in and process target sequences
            values: the output of encoder layers, which is the values
            query: usually the state output of encoder
            y: the sequence of the decoder layers
        """
        input_t = tf.expand_dims([self.start] * y.shape[0], 1)
        query_t = query
        outputs = tf.TensorArray(tf.float32, size=tf.shape(y)[1]-1, dynamic_size=True)
        for t in tf.range(1, tf.shape(y)[1]):
            context, _ = self.attention(query_t, values, training=training)
            output_t, query_t = self.time_propagate(context, query_t, input_t, training=training)
            input_t = tf.expand_dims(y[:, t], 1)
            outputs = outputs.write(t-1, output_t)
        return tf.transpose(outputs.stack(), [1, 0, 2])

    def time_propagate(self, context, query, y, training=None):
        """ Take in and process target sequences only propagate 1 time step
            context: the context computed by attention
            values: the output of encoder layers, the value for attention
            y: the sequence of the decoder layers
        """
        y = self.embedding(y, training=training)
        y = tf.concat([tf.expand_dims(context, 1), y], axis=-1)
        output, query = self.rnn(y, training=training)
        output = tf.reshape(output, (-1, output.shape[2]))
        output = self.dense(output, training=training)
        return output, query
```

训练时把标注作为输入,推断时将上一个时刻Decoder的输出作为输入

output:whole_sequence_output
query:final_state

A simple implementation of LAS

深蓝学院 shenlanxueyuan.com

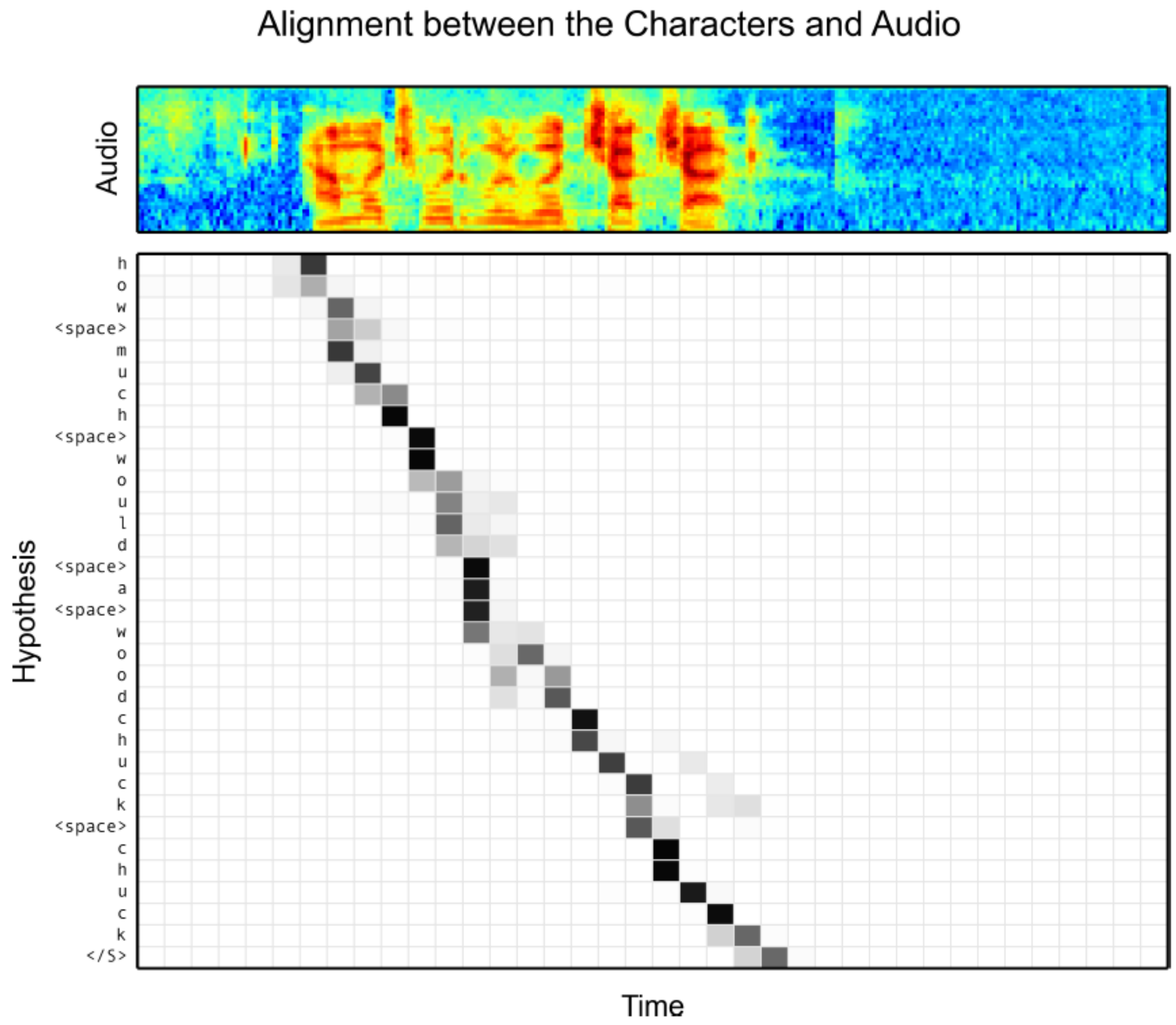
- Loss Function

```
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')

def loss_function(real, pred):
  mask = tf.math.logical_not(tf.math.equal(real, 0))
  loss_ = loss_object(real, pred)

  mask = tf.cast(mask, dtype=loss_.dtype)
  loss_ *= mask

  return tf.reduce_mean(loss_)
```

mask:序列长度不一

Alignment for seq2seq with attention

- In SMT, alignment is the correspondence between particular words in the translated sentence pair
  - Alignment can be many-to-one, one-to-many, or many-to-many
- In ASR, alignment is the correspondence between the characters and audio signals



Decoding for seq2seq with attention

深蓝学院
shenlanxueyuan.com

- Usually using dynamic programming for globally optimal solutions, e.g. Viterbi algorithms

- In practical, we use beam search decoding 输出同步
  - Core idea: On each step of decoder, keep track of the *k* most probable partial translations (which we call *hypotheses*)
    - *k* is the beam size
  - When a hypothesis produces <END>, that hypothesis is complete
    - Different hypothesis may produce <END> on different time-steps

Listen-Attend-Spell (LAS)

深蓝学院
shenlanxueyuan.com

- Compared to CTC or HMM
- Advantages
  - Better performance
  - A single neural network to be optimized end-to-end
  - Requires much less human engineering effort
- Disadvantages
  - Hard to debug
  - Hard to perform streaming decoding

2.3 Speech-Transformer

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

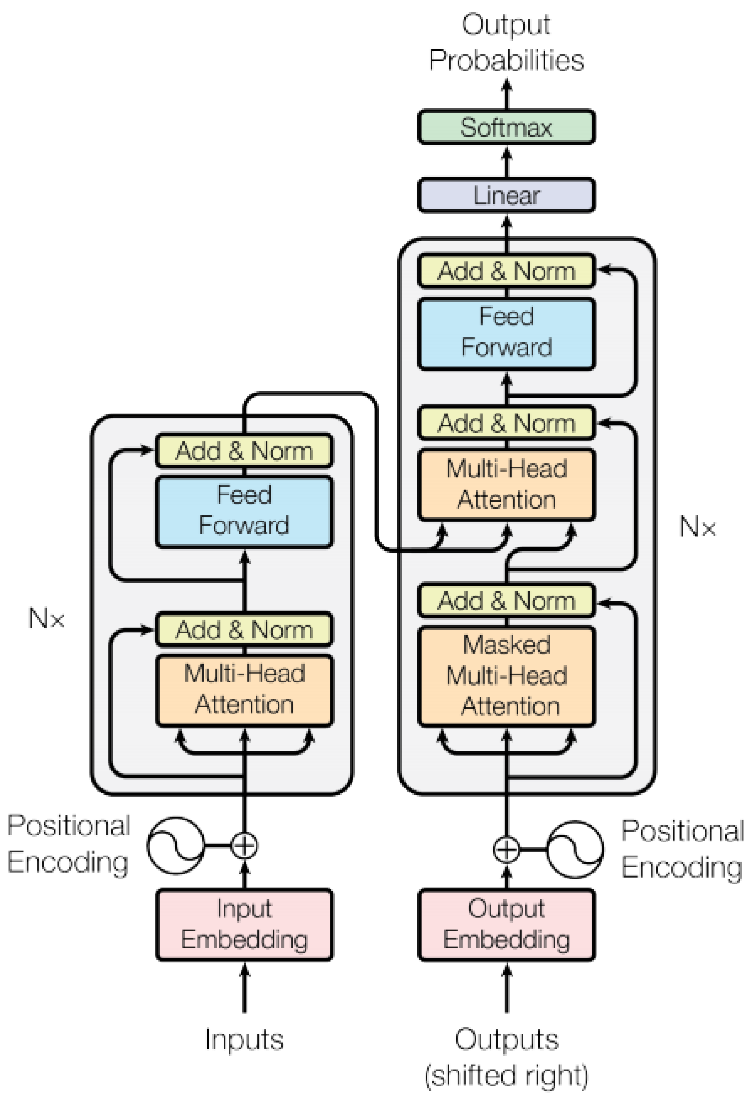
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

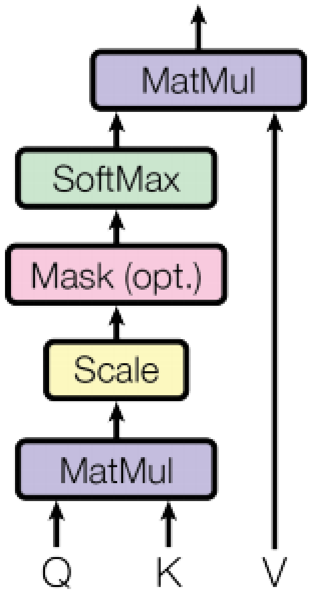
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.



A simple implementation of transformer

深蓝学院
shenlanxueyuan.com

- Scaled Dot-Product Attention

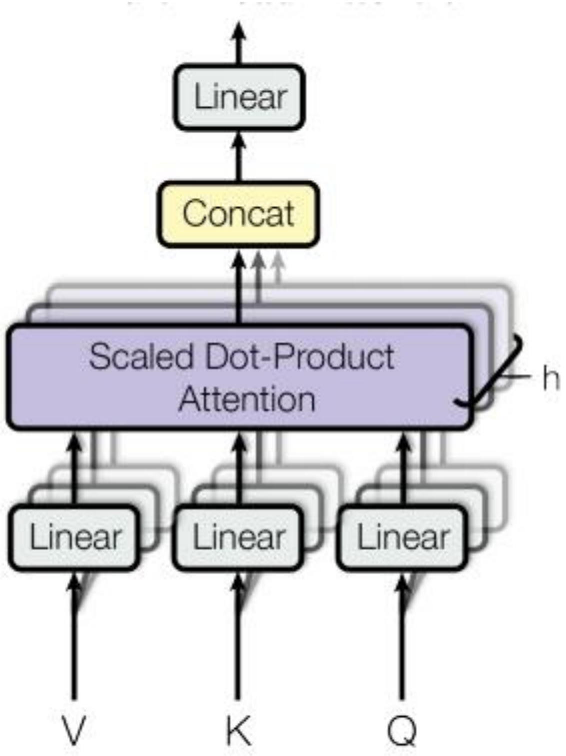


query:同前
key:
value:

```python
class ScaledDotProductAttention(tf.keras.layers.Layer):
    """Calculate the attention weights.
    q, k, v must have matching leading dimensions.
    k, v must have matching penultimate dimension, i.e.: seq_len_k = seq_len_v.
    The mask has different shapes depending on its type(padding or look ahead)
    but it must be broadcastable for addition.

    Args:
        q: query shape == (..., seq_len_q, depth)
        k: key shape == (..., seq_len_k, depth)
        v: value shape == (..., seq_len_v, depth_v)
        mask: Float tensor with shape broadcastable
          to (..., seq_len_q, seq_len_k). Defaults to None.

    Returns:
        output, attention_weights
    """

    def call(self, q, k, v, mask):
        """This is where the layer's logic lives."""
        matmul_qk = tf.matmul(q, k, transpose_b=True)  # (..., seq_len_q, seq_len_k)

        # scale matmul_qk
        dk = tf.cast(tf.shape(k)[-1], tf.float32)
        scaled_attention_logits = matmul_qk / tf.math.sqrt(dk)

        # add the mask to the scaled tensor.
        if mask is not None:
            scaled_attention_logits += mask * -1e9

        # softmax is normalized on the last axis (seq_len_k) so that the scores
        # add up to 1. (..., seq_len_q, seq_len_k)
        attention_weights = tf.nn.softmax(scaled_attention_logits, axis=-1)

        output = tf.matmul(attention_weights, v)  # (..., seq_len_q, depth_v)

        return output, attention_weights
```

A simple implementation of transformer

- Multi-head attention



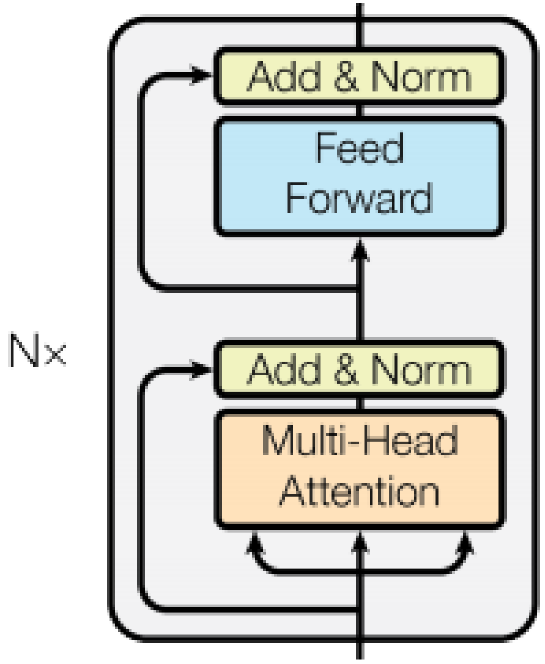

```python
class MultiHeadAttention(tf.keras.layers.Layer):
    def __init__(self, d_model, num_heads):
        super().__init__()
        self.num_heads = num_heads
        self.d_model = d_model
        assert d_model % self.num_heads == 0
        self.depth = d_model // self.num_heads

        self.wq = tf.keras.layers.Dense(d_model)
        self.wk = tf.keras.layers.Dense(d_model)
        self.wv = tf.keras.layers.Dense(d_model)

        self.attention = ScaledDotProductAttention()
        self.dense = tf.keras.layers.Dense(d_model)      You, a few seconds ago • Uncomm

    def split_heads(self, x, batch_size):
        x = tf.reshape(x, (batch_size, -1, self.num_heads, self.depth))
        return tf.transpose(x, perm=[0, 2, 1, 3])

    def call(self, v, k, q, mask):
        batch_size = tf.shape(q)[0]

        q = self.wq(q)  # (batch_size, seq_len, hiddn_dim)
        k = self.wk(k)  # (batch_size, seq_len, hiddn_dim)
        v = self.wv(v)  # (batch_size, seq_len, hiddn_dim)

        q = self.split_heads(q, batch_size)  # (batch_size, num_heads, seq_len_q, depth)
        k = self.split_heads(k, batch_size)  # (batch_size, num_heads, seq_len_k, depth)
        v = self.split_heads(v, batch_size)  # (batch_size, num_heads, seq_len_v, depth)

        scaled_attention, attention_weights = self.attention(q, k, v, mask)

        # (batch_size, seq_len_q, num_heads, depth)
        scaled_attention = tf.transpose(scaled_attention, perm=[0, 2, 1, 3])

        # (batch_size, seq_len_q, d_model)
        concat_attention = tf.reshape(scaled_attention, (batch_size, -1, self.d_model))

        output = self.dense(concat_attention)  # (batch_size, seq_len_q, d_model)

        return output, attention_weights
```

A simple implementation of transformer

深蓝学院
shenlanxueyuan.com

- TransformerEncoderLayer

```
class TransformerEncoderLayer(tf.keras.layers.Layer):
    def __init__(self, d_model, nhead, dim_feedforward=2048, dropout=0.1):
        super().__init__()
        self.self_attn = MultiHeadAttention(d_model, nhead)
        layers = tf.keras.layers
        self.ffn = tf.keras.Sequential([
            layers.Dense(dim_feedforward, activation="relu"),
            layers.Dropout(dropout),
            layers.Dense(d_model),
            layers.Dropout(dropout)]
        )

        self.norm1 = layers.LayerNormalization()
        self.norm2 = layers.LayerNormalization()
        self.dropout = layers.Dropout(dropout)

    def call(self, src, src_mask=None, training=None):
        out = self.self_attn(src, src, src, mask=src_mask)[0]
        out = self.norm1(src + self.dropout(out, training=training))
        out = self.norm2(out + self.ffn(out, training=training))

        return out        You, 3 months ago • first commit
```

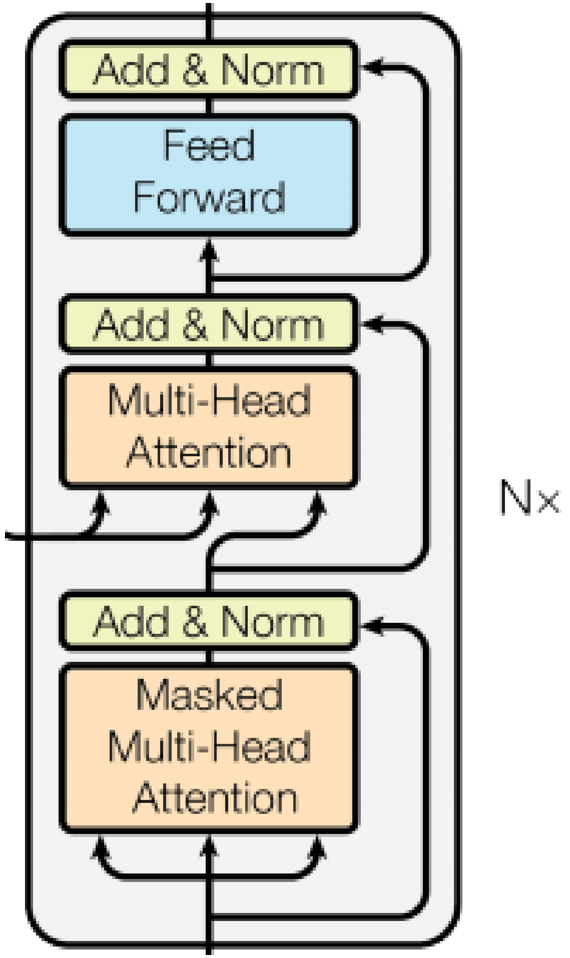


A simple implementation of transformer

深蓝学院 shenlanxueyuan.com

- TransformerDecoderLayer

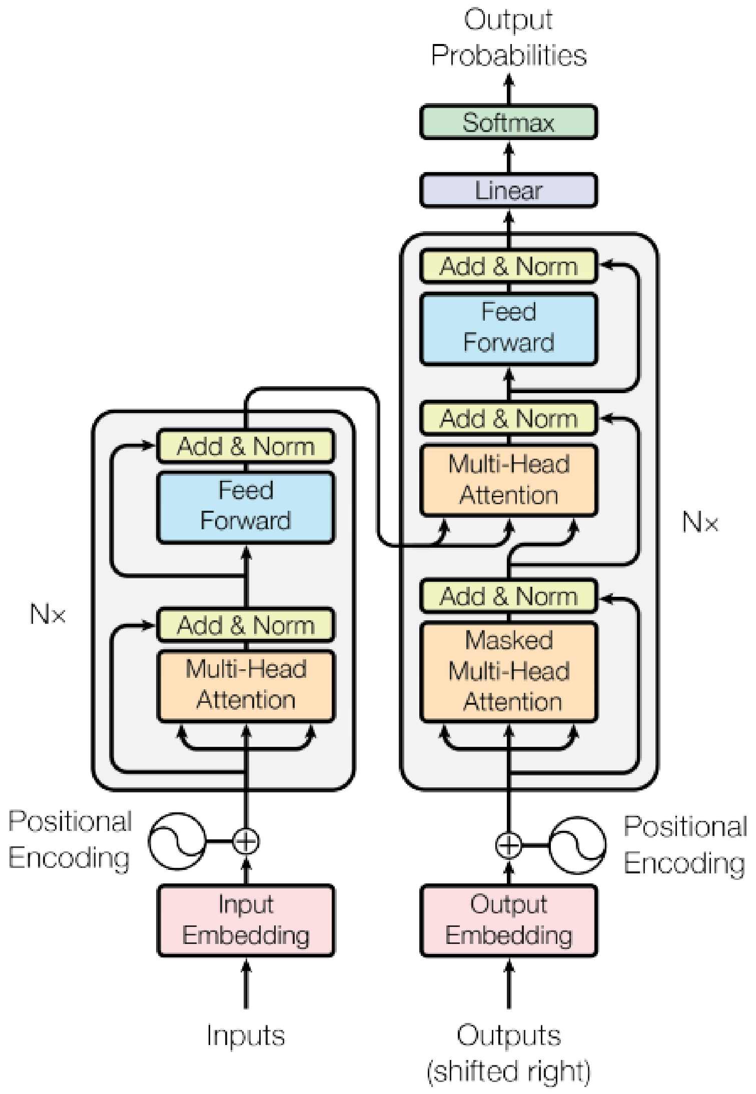
```
class TransformerDecoderLayer(tf.keras.layers.Layer):
    def __init__(self, d_model, nhead, dim_feedforward=2048, dropout=0.1):
        super().__init__()
        self.attn1 = MultiHeadAttention(d_model, nhead)
        self.attn2 = MultiHeadAttention(d_model, nhead)
        layers = tf.keras.layers
        self.ffn = tf.keras.Sequential([
                layers.Dense(dim_feedforward, activation="relu"),
                layers.Dropout(dropout),
                layers.Dense(d_model),
                layers.Dropout(dropout)]
        )

        self.norm1 = layers.LayerNormalization()
        self.norm2 = layers.LayerNormalization()
        self.norm3 = layers.LayerNormalization()
        self.dropout1 = layers.Dropout(dropout)
        self.dropout2 = layers.Dropout(dropout)

    def call(self, tgt, memory, tgt_mask=None, memory_mask=None, training=None):
        out = self.attn1(tgt, tgt, tgt, mask=tgt_mask)[0]
        out = self.norm1(tgt + self.dropout1(out, training=training))
        out2 = self.attn2(memory, memory, out, mask=memory_mask)[0]
        out = self.norm2(out + self.dropout2(out2, training=training))
        out = self.norm3(out + self.ffn(out, training=training))
        return out
```



A simple implementation of transformer

深蓝学院 shenlanxueyuan.com

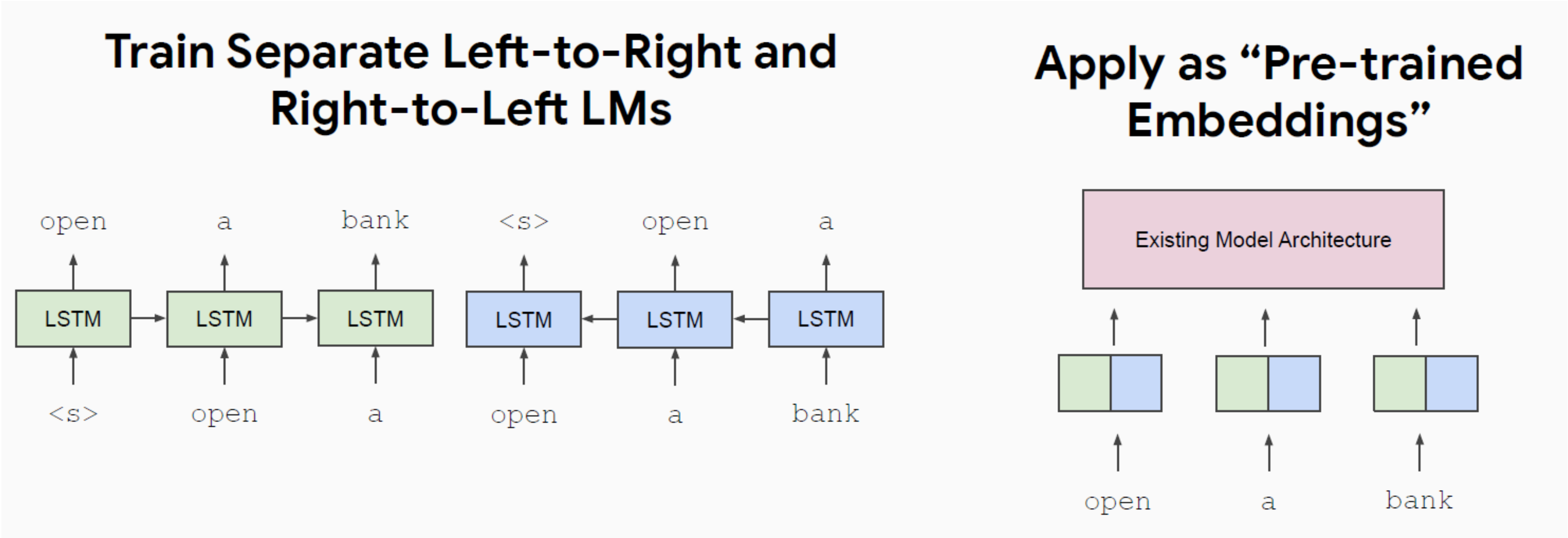
```python
class Transformer(tf.keras.layers.Layer):
    def __init__(self, d_model=512, nhead=8, num_encoder_layers=6, num_decoder_layers=6,
        dim_feedforward=2048, dropout=0.1):
        super().__init__()
        self.encoder = [TransformerEncoderLayer(d_model, nhead, dim_feedforward, dropout)
                        for _ in range(num_encoder_layers)]

        self.decoder = [TransformerDecoderLayer(d_model, nhead, dim_feedforward, dropout)
                        for _ in range(num_decoder_layers)]


    def call(self, src, tgt, src_mask=None, tgt_mask=None, memory_mask=None,
            return_encoder_output=False, training=None):
        memory = src
        for i in range(len(self.encoder)):
            memory = self.layers[i](memory, src_mask=src_mask, training=training)

        output = tgt
        for i in range(len(self.decoder)):
            output = self.layers[i](
                output, memory, tgt_mask=tgt_mask, memory_mask=memory_mask, training=training)

        return output
```



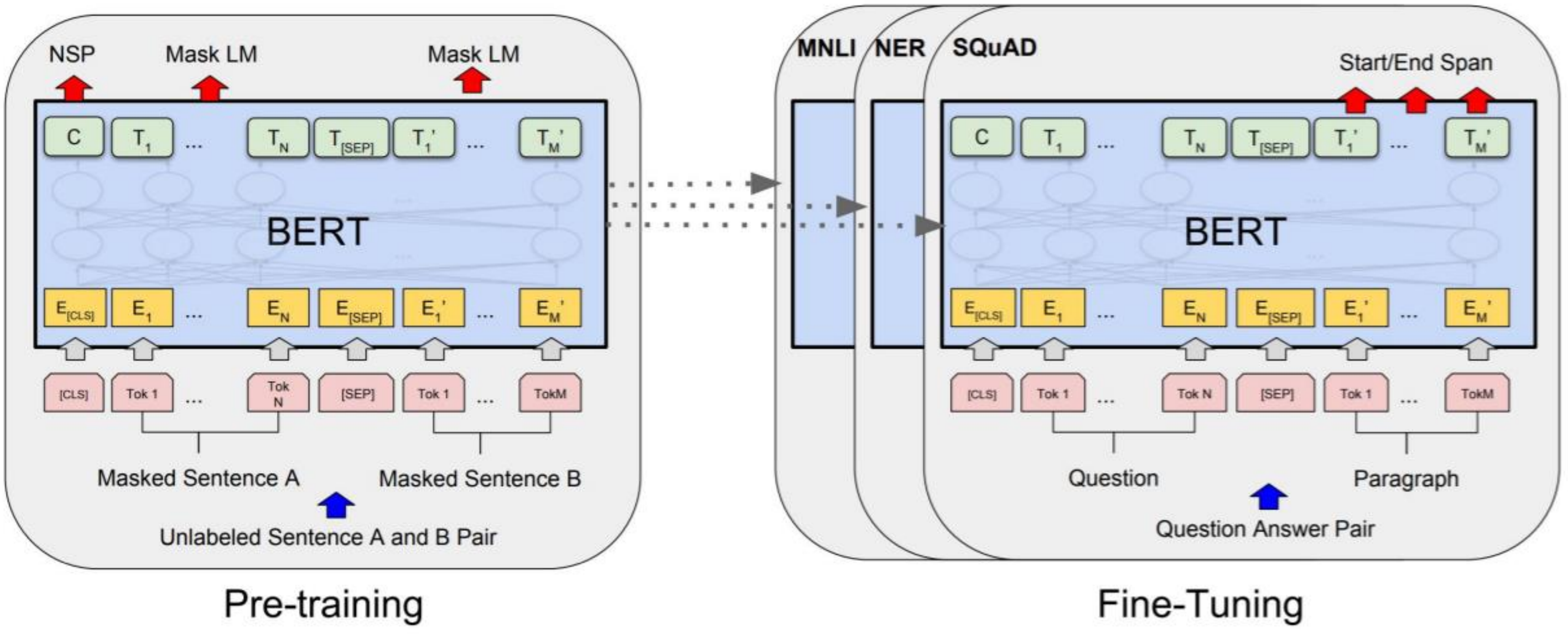
Outline

3.1 Pretraining in NLP

- Word embeddings (like word2vec) are the basis of deep learning for NLP
- ELMo: Deep contextual word embeddings

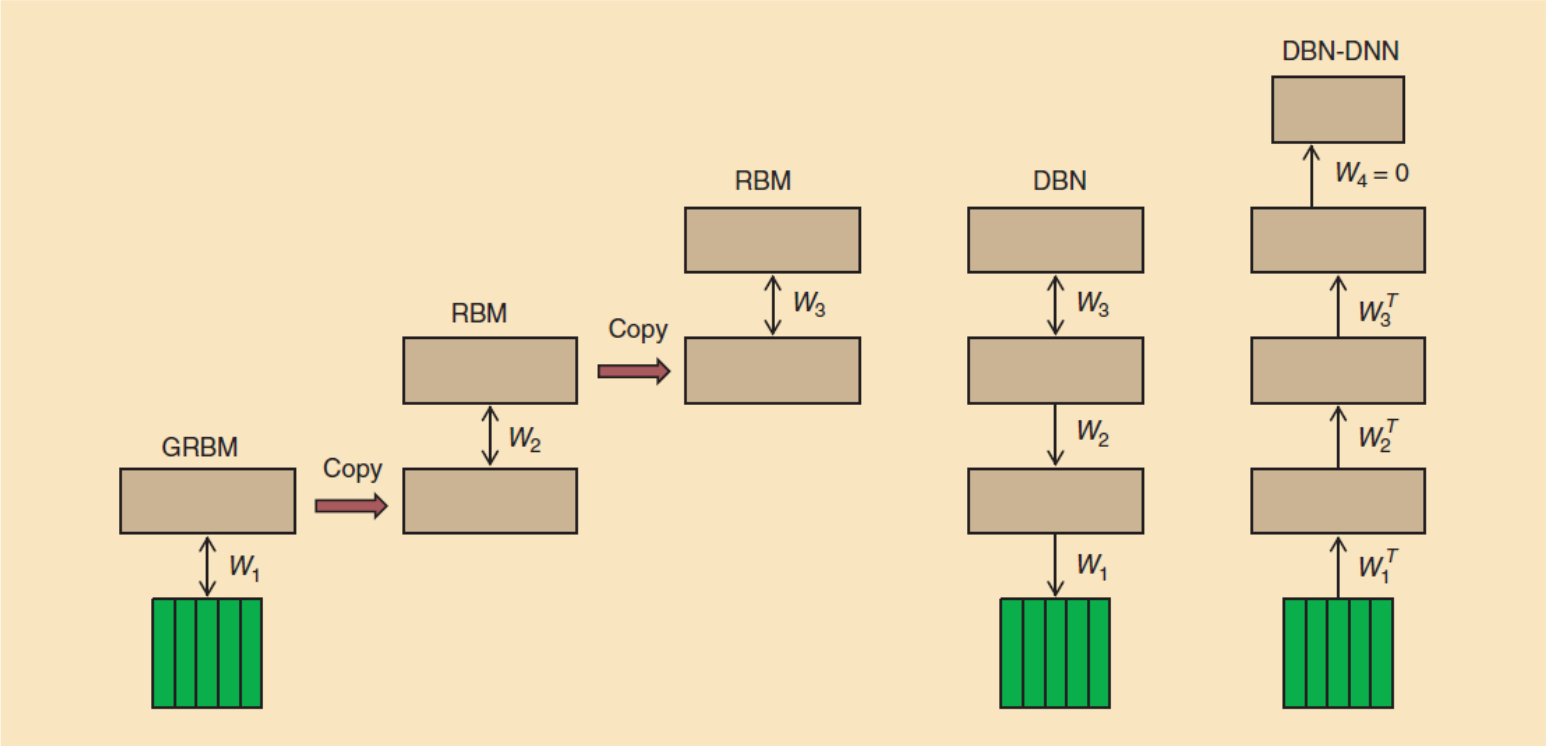


Using BERT


NSP
Mask LM
Mask LM
BERT
C
T1
TN
T[SEP]
T1'
TM'
E[CLS]
E1
EN
E[SEP]
E1'
EM'
[CLS]
Tok 1
Tok N
[SEP]
TokM
Masked Sentence A
Masked Sentence B
Unlabeled Sentence A and B Pair
Pre-training
MNLI
NER
SQuAD
Start/End Span
Question
Paragraph
Question Answer Pair
Fine-Tuning

3.2 Pretraining in ASR

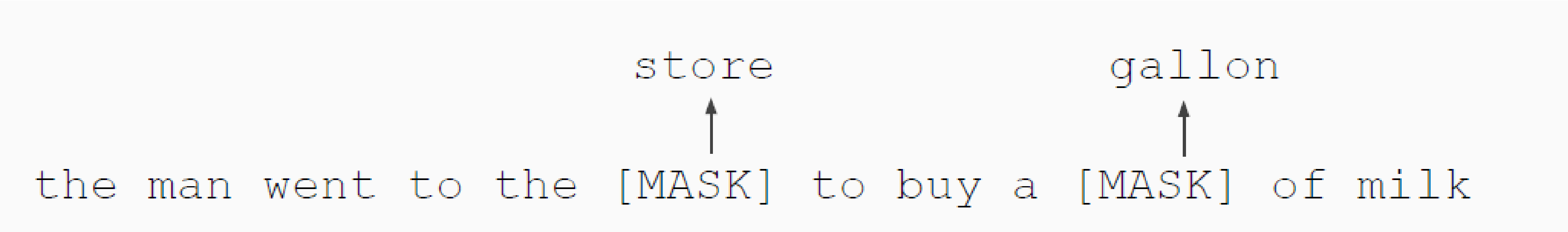
- Generative pretraining



[FIG1] The sequence of operations used to create a DBN with three hidden layers and to convert it to a pretrained DBN-DNN. First, a GRBM is trained to model a window of frames of real-valued acoustic coefficients. Then the states of the binary hidden units of the GRBM are used as data for training an RBM. This is repeated to create as many hidden layers as desired. Then the stack of RBMs is converted to a single generative model, a DBN, by replacing the undirected connections of the lower level RBMs by top-down, directed connections. Finally, a pretrained DBN-DNN is created by adding a "softmax" output layer that contains one unit for each possible state of each HMM. The DBN-DNN is then discriminatively trained to predict the HMM state corresponding to the central frame of the input window in a forced alignment.

BERT-like pretraining for Speech Transformer

- Key idea of BERT:
  - Mask out *k*% of the input words, and then predict the masked words



- In Speech
  - The input is acoustic spectrum, and time shift is 10 ms
  - Solution:
    - Using MSE loss function
    - using convolution layer to perform downscale, for example 1/8

BERT-like pretraining for Speech Transformer

- Some reference papers:

[1] D Jiang, X Lei, W Li. et.al. Improving transformer-based speech recognition using unsupervised pre-training. arxiv.1910.09932

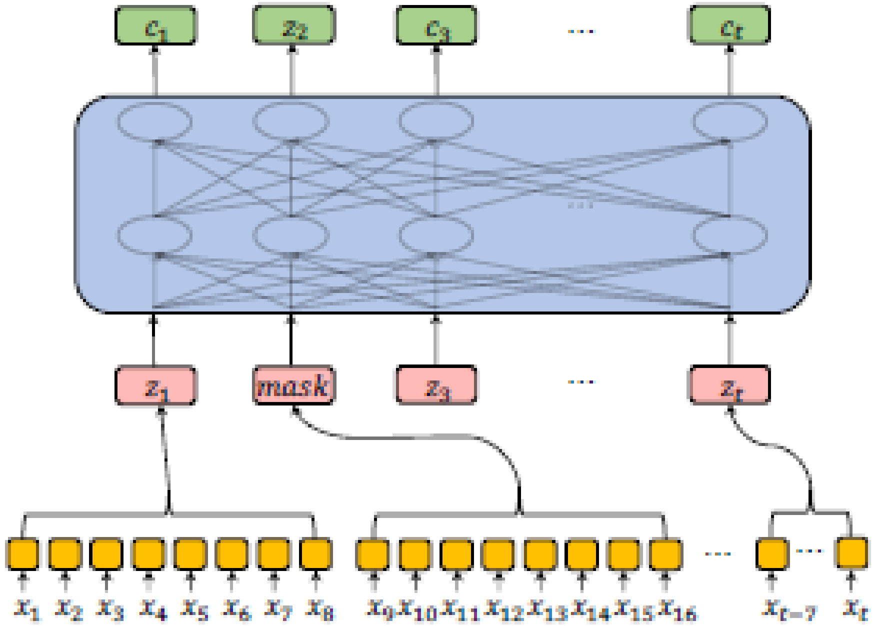


Fig. 2. Masked Predictive Coding with eight-fold downsample.

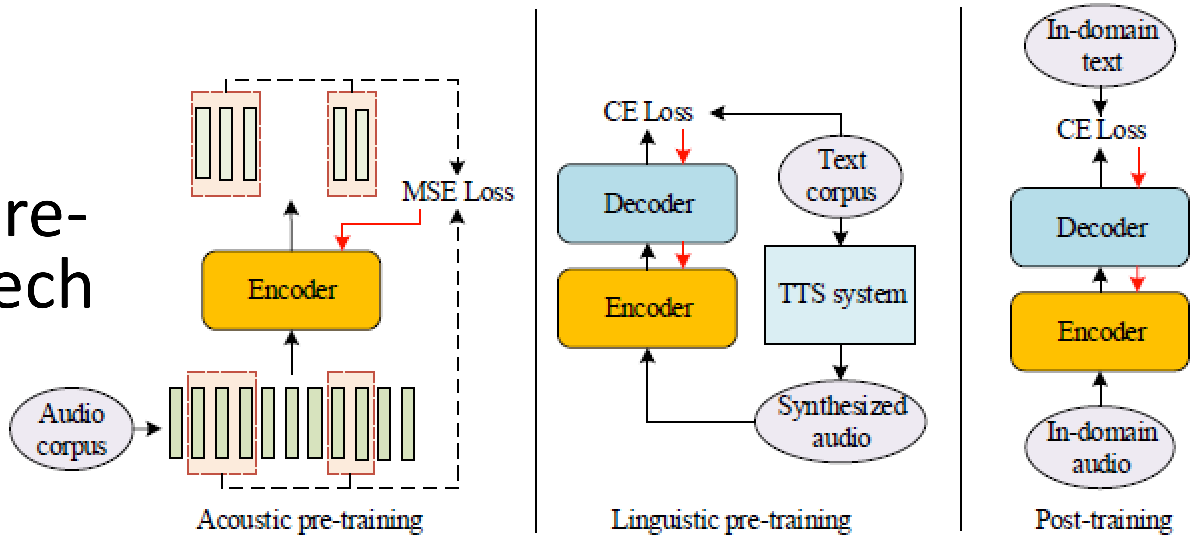
BERT-like pretraining for Speech Transformer

深蓝学院
shenlanxueyuan.com

- Some reference papers:

[1] D Jiang, X Lei, W Li. et.al. Improving transformer-based speech recognition using unsupervised pre-training. arxiv.1910.09932

[2] Z Fan, S Zhou, B Xu. unsupervised pre-training for sequence to sequence speech recognition. arxiv. 1910.12418



Outline

深蓝学院
shenlanxueyuan.com

4 Homework

- Select one dataset and build the scripts just like those in athena examples
  - TIMIT
  - THCHS-30: http://openslr.org/18
  - Free ST Chinese Mandarin Corpus: http://openslr.org/38
  - Primewords Chinese Corpus Set 1: http://openslr.org/47
  - aidatatang_200zh: http://openslr.org/62
  - ...
- Better performance, better homework score

4 Homework

- How to build a project on athena
  - Step 1: prepare the dataset csv file

```
wav_filename    wav_length_ms    transcript
/dataset/train-clean-100-wav/374-180298-0000.wav        465004   chapter sixteen i might have told you of the b
/dataset/train-clean-100-wav/374-180298-0001.wav        514764   marguerite to be unable to live apart from me
/dataset/train-clean-100-wav/374-180298-0002.wav        425484   i wished above all not to leave myself time tc
/dataset/train-clean-100-wav/374-180298-0003.wav        356044   assumed all at once an appearance of noise anc
```

  - Step 2： prepare the model configuration json file
  - Step 3： just train and test

Thanks!