Pre-training method in NLP-ELMo

ZhangChenBin

Statistical Machine Intelligence and Learning (SMILE) Lab
University of Electronic Science and Technology of China

outline

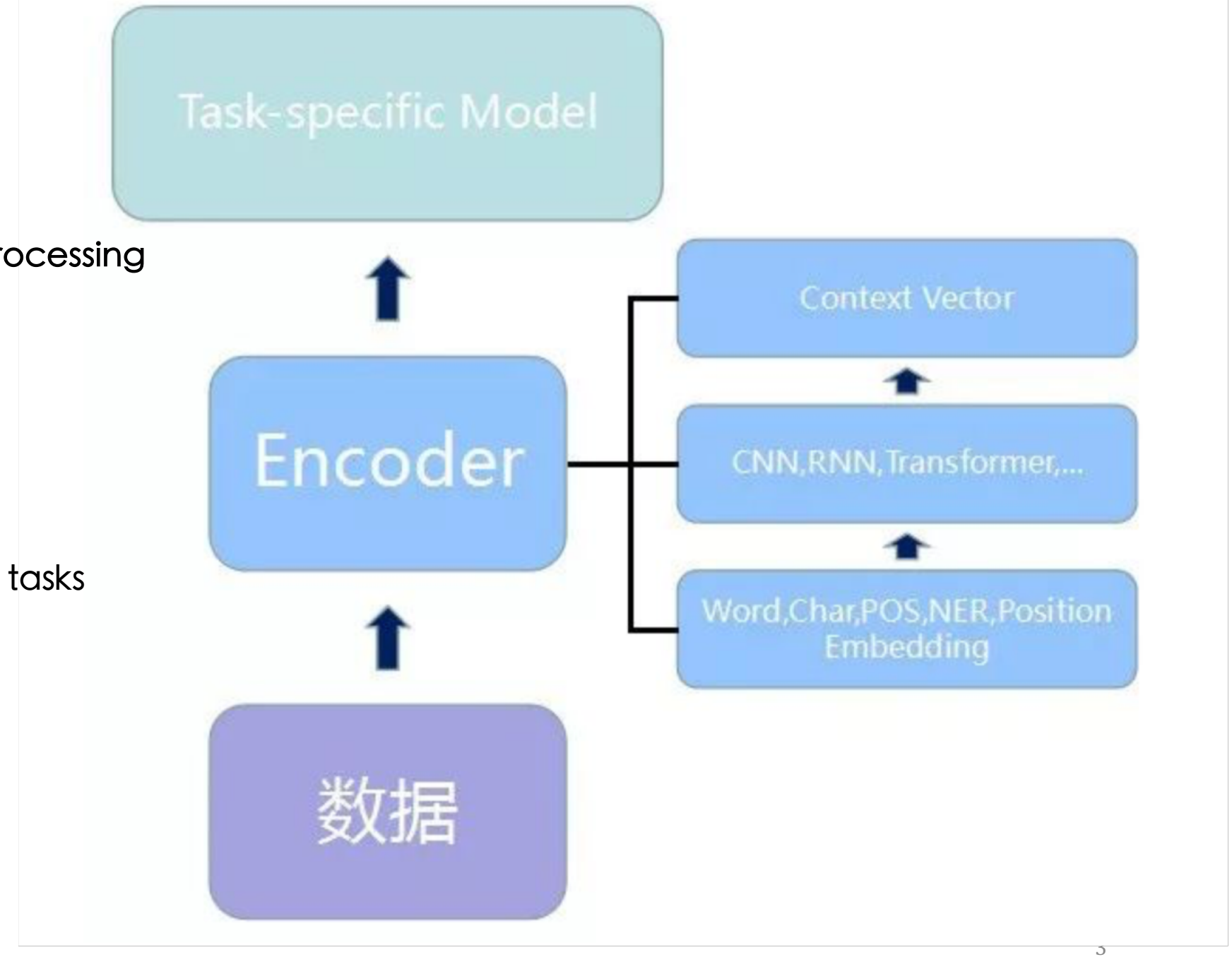
- Overview
  - Language model
  - Neural Net Language Model
  - Word2vec

- Analysis of ELMo
  - Character-CNN structure
  - Bidirectional language models
  - EMLO
  - Analysis

- Text data collection and preprocessing

- Encode and characterize text

- Design model to solve specific tasks



Language model

Sentence 1:美联储主席本·伯南克昨天告诉媒体7000亿美元的救助资金

Sentence 2:美主席联储本·伯南克告诉昨天媒体7000亿美元的资金救

Sentence 3:美主车席联储本·克告诉昨天公司媒体7000伯南亿美行元

哪个句子更像一个合理的句子？如何量化评估?

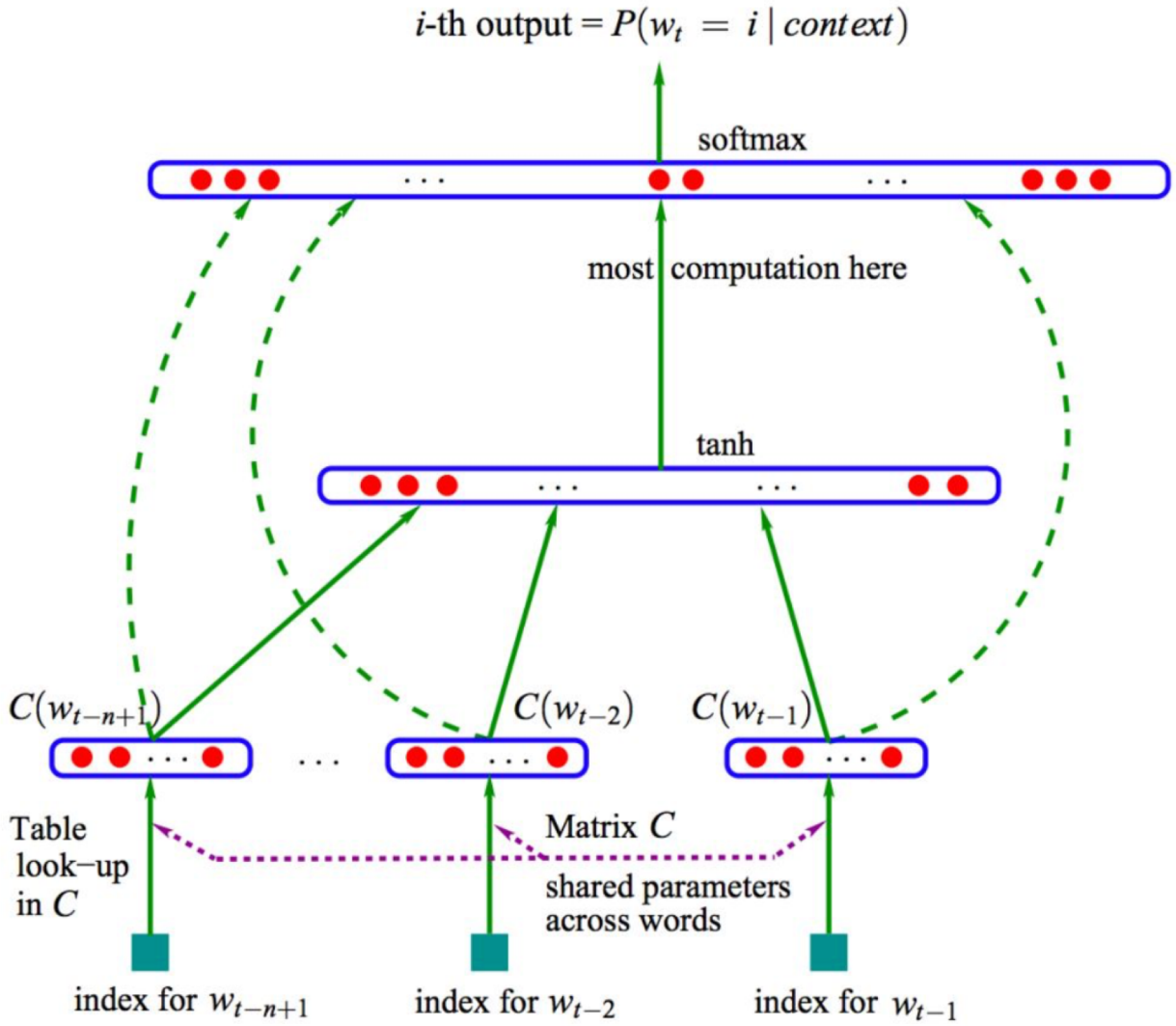
$$P(S) = P(w_1, w_2, \ldots, w_n)$$

$$P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\cdots P(w_n|w_1, w_2, \cdots w_{n-1})$$

语言模型：

$$L = \sum_{w \in C} logP(w|context(w))$$

Neural Net Language Model



$$L = \frac{1}{T} \sum_t \log P(w_t | w_{t-1}, \ldots, w_{t-n+1}; \theta) + R(\theta)$$

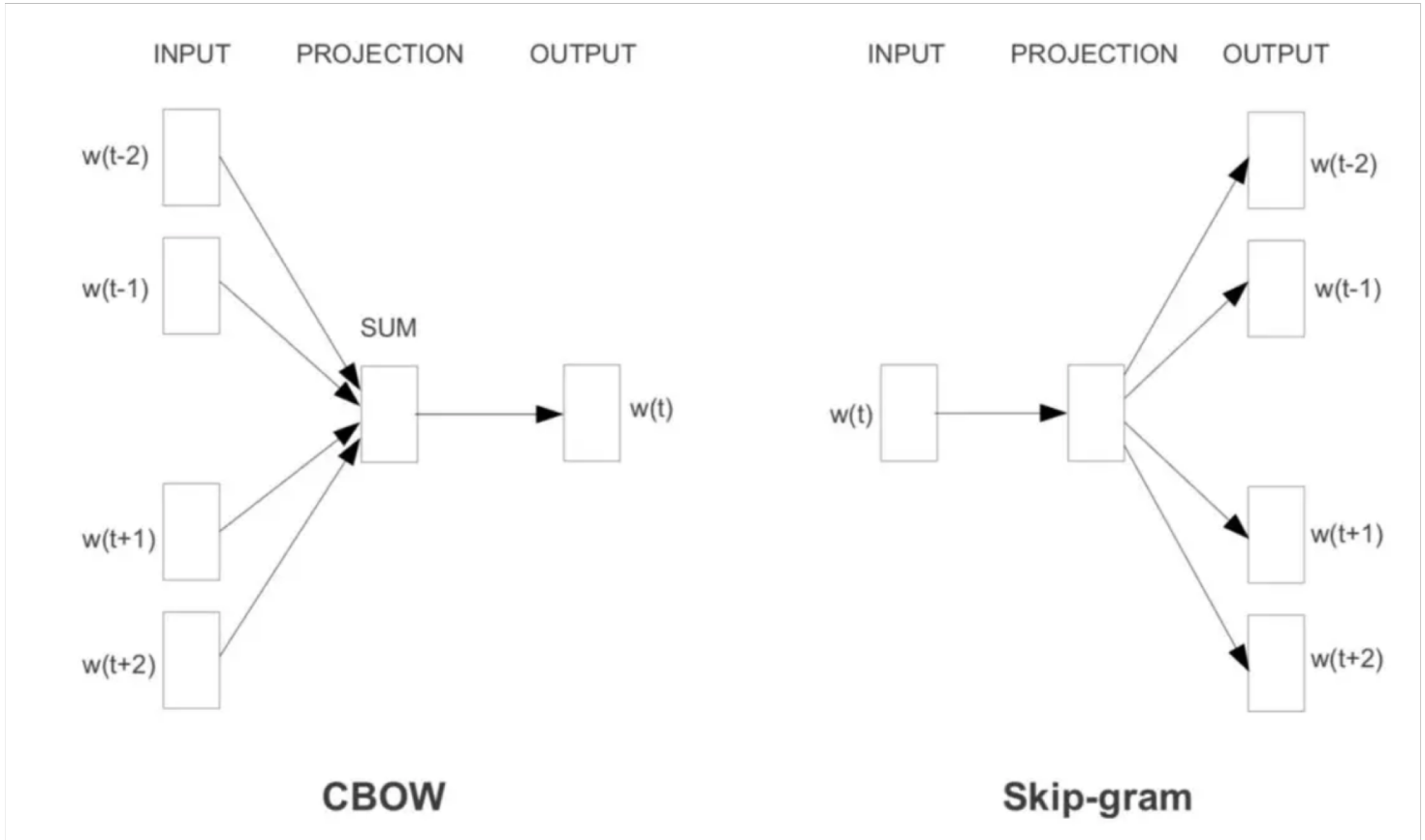
$$P(w_t | w_{t-1}, \ldots, w_{t-n+1}; \theta) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

$$y = b + Wx + Utanh(d + Hx)$$

$$x = (C(w_{t-1}), C(w_{t-2}), \ldots, C(w_{t-n+1}))$$

Bengio, Yoshua, et al. "A neural probabilistic language model." *Journal of machine learning research* 3.Feb (2003): 1137-1155.

Pre-trained word representations -- word2vec

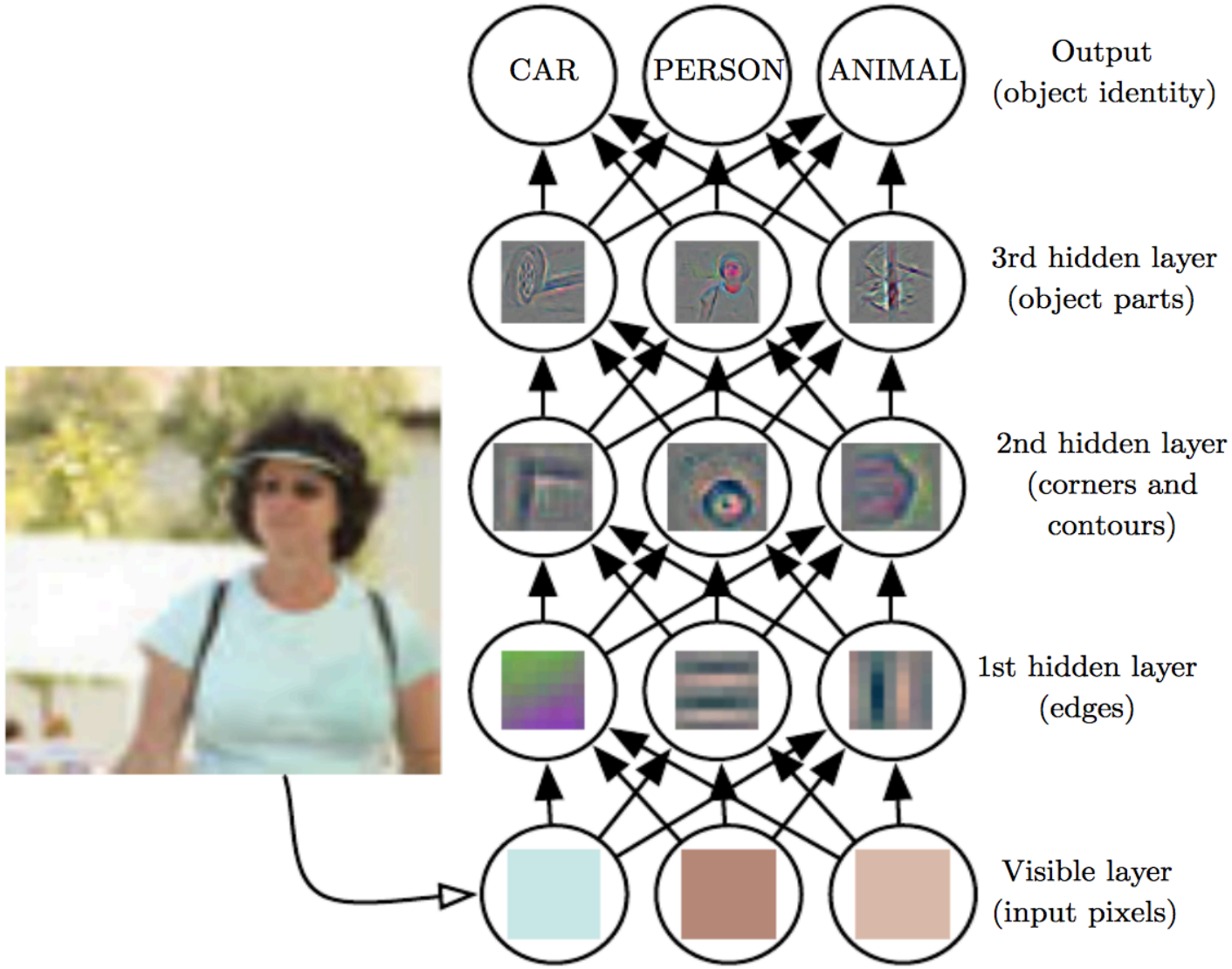


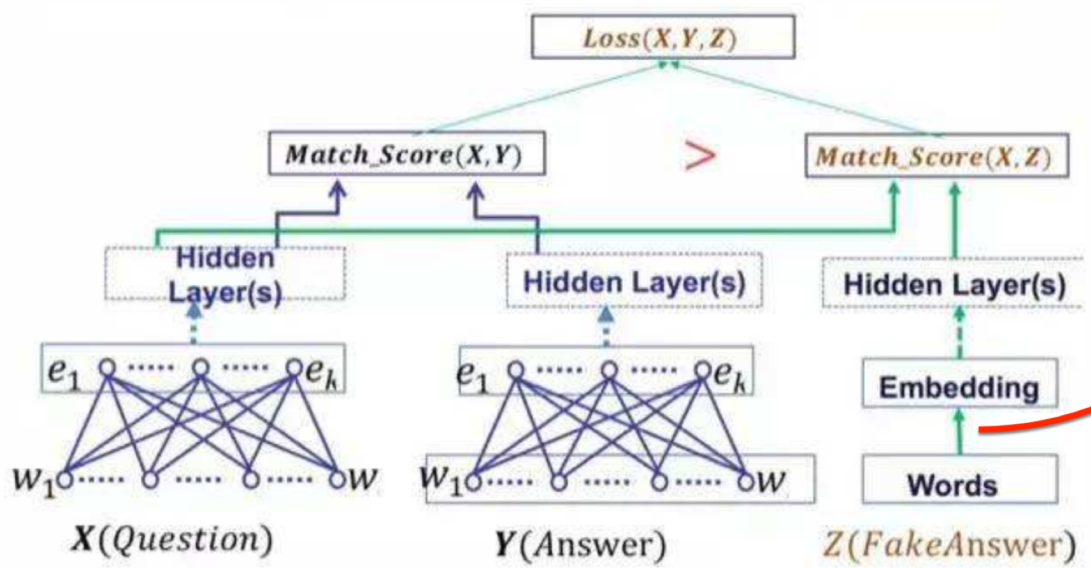
Mikolov T, Chen K, Corrado G, et al. Efficient estimation of word representations in vector space[J]. arXiv preprint arXiv:1301.3781, 2013.

```
In [10]: result = model.most_similar(u"青蛙")

In [11]: for e in result:
    ....:     print e[0], e[1]
    ....:
老鼠 0.559612870216
乌龟 0.489831030369
蜥蜴 0.478990525007
猫 0.46728849411
鳄鱼 0.461885392666
蟾蜍 0.448014199734
猴子 0.436584025621
白雪公主 0.434905380011
蚯蚓 0.433413207531
螃蟹 0.4314712286
```

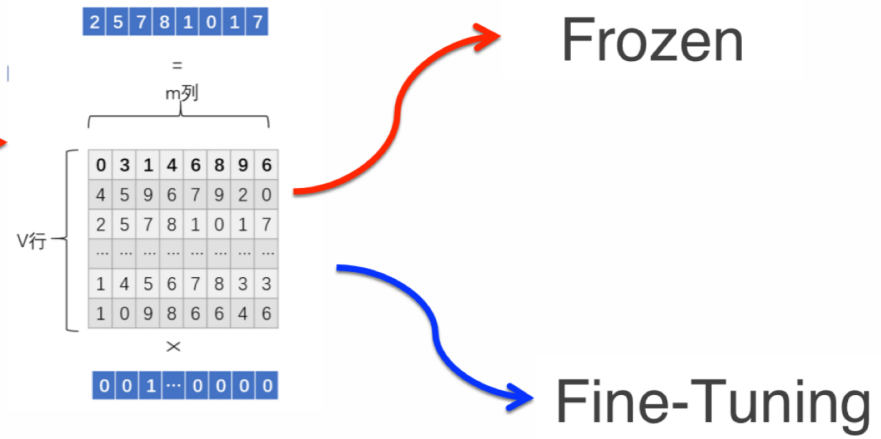
```
In [20]: result = model.most_similar(u"清华大学

In [21]: for e in result:
    ....:     print e[0], e[1]
    ....:
北京大学 0.763922810555
化学系 0.724210739136
物理系 0.694550514221
数学系 0.684280991554
中山大学 0.677202701569
复旦 0.657914161682
师范大学 0.656435549259
哲学系 0.654701948166
生物系 0.654403865337
中文系 0.653147578239
```

Pre-training in computer vison





QA



Word Embedding Matrix

very useful to protect banks or slopes from being washed away by river or raining

the location because it was high, about 100 feet above the bank of river

The bank has plan to branch throughout the country

Do they have the same meaning?

- complex characteristics of word use
  - Syntax
  - Semantics

Question? ?

- how these uses vary across linguistic contexts ?

Deep contextualized word representations

Matthew E. Peters[†], **Mark Neumann**[†], **Mohit Iyyer**[†], **Matt Gardner**[†],
{matthewp,markn,mohiti,mattg}@allenai.org

Christopher Clark[*], **Kenton Lee**[*], **Luke Zettlemoyer**[†*]
{csquared,kentonl,lsz}@cs.washington.edu

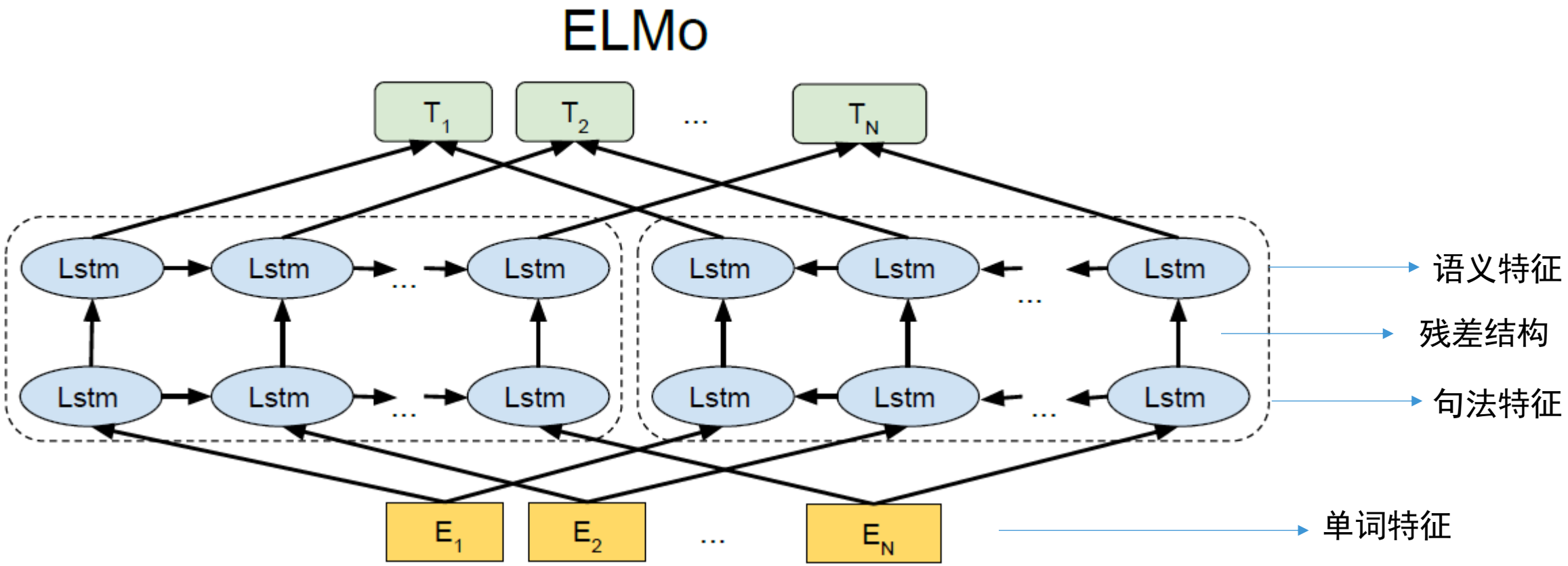
[†]Allen Institute for Artificial Intelligence
[*]Paul G. Allen School of Computer Science & Engineering, University of Washington

Contextualized Word Representation

- word vectors are context-independent. the same word is always the same word vector in different contexts, which obviously leads to the lack of word-disambiguation (WSD) in the word vector model

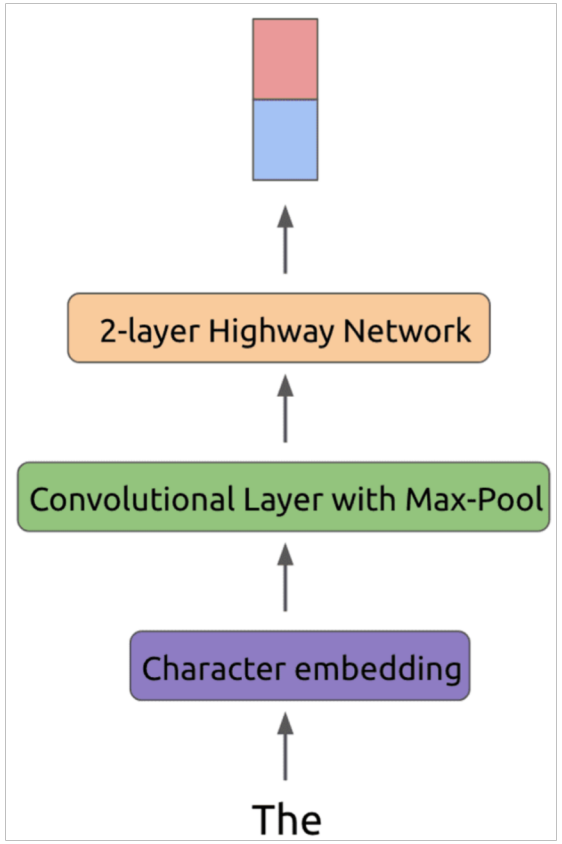
- People also realized that the char-level text also contains some patterns that are difficult to describe by word-level text.

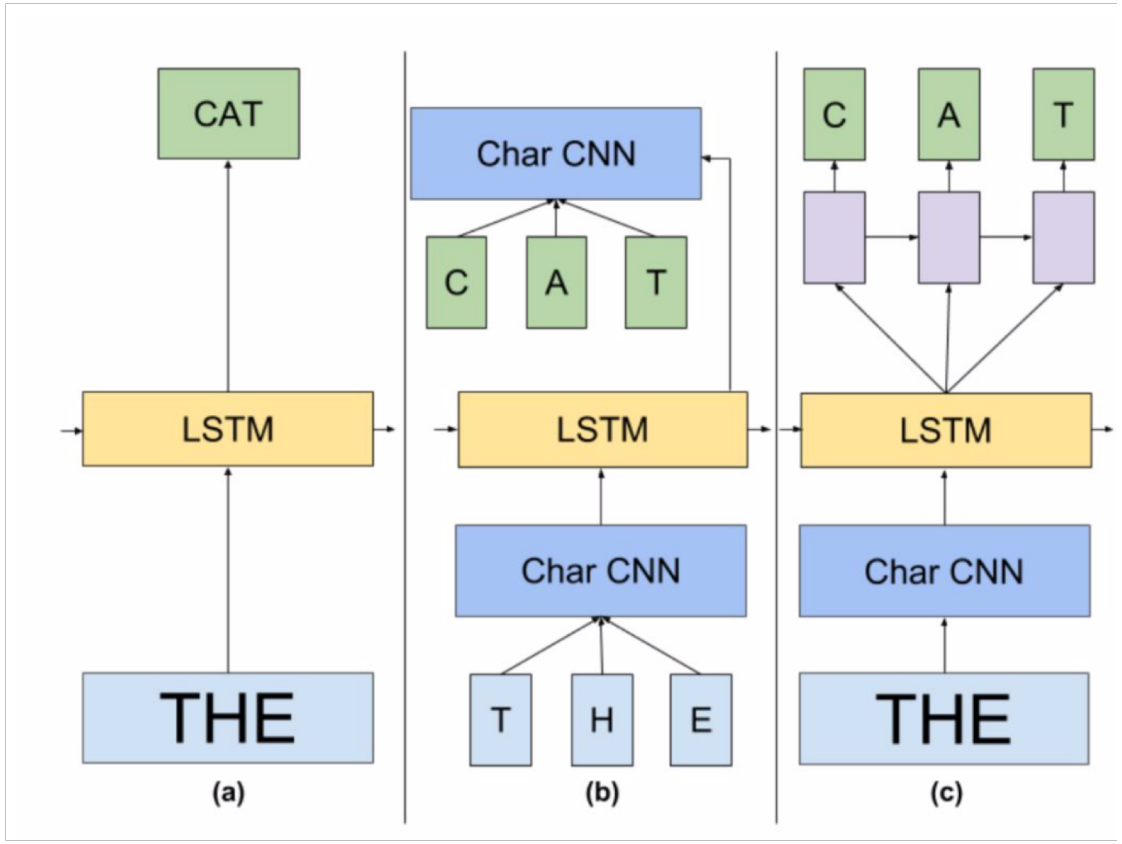

Hey ELMo, what's the embedding of the word "stick"?
There are multiple possible embeddings! Use it in a sentence.
Oh, okay. Here: "Let's stick to improvisation in this skit"
Oh in that case, the embedding is: -0.02, -0.16, 0.12, -0.1 ….etc



- two-layer biLMs with character convolutions
- Different layer represent different information

Encoder: **Character-CNN** structure





```python
#将词转化为char_ids
def _convert_word_to_char_ids(self, word):
    code = np.zeros([self.max_word_length], dtype=np.int32)
    code[:] = self.pad_char

    #将word中每一个字符转化为utf-8编码，然后用数组存起来，例如:
    #english中，e:101, n:110, g:103, l:108, h:105, s:115, h:104
    word_encoded = word.encode('utf-8', 'ignore')[:(self.max_word_length-2)]
    code[0] = self.bow_char
    #加上词开始和结尾的编码
    for k, chr_id in enumerate(word_encoded, start=1):
        code[k] = chr_id
    code[k + 1] = self.eow_char

    return code
```

```
def __init__(self, filename, max_word_length, **kwargs):
    #调用父类Vocabulary，生成word和id之间的转换等
    super(UnicodeCharsVocabulary, self).__init__(filename, **kwargs)
    #每个词对应最大字符长
    self._max_word_length = max_word_length

    # char ids 0-255 come from utf-8 encoding bytes
    # assign 256-300 to special chars
    self.bos_char = 256  # <begin sentence>
    self.eos_char = 257  # <end sentence>
    self.bow_char = 258  # <begin word>
    self.eow_char = 259  # <end word>
    self.pad_char = 260 # <padding>

    #单词的个数，父类中的属性
    num_words = len(self._id_to_word)

    #每个词都会对应一个char_ids列表
    self._word_char_ids = np.zeros([num_words, max_word_length],
        dtype=np.int32)

    # the charcter representation of the begin/end of sentence characters
    # 对句首或者句尾的token来一个字符的表示
    def _make_bos_eos(c):
        r = np.zeros([self.max_word_length], dtype=np.int32)
        r[:] = self.pad_char
        #词的开始
        r[0] = self.bow_char
        r[1] = c
        #词的结束
        r[2] = self.eow_char
        return r
    #句子开始对应的char_ids
    self.bos_chars = _make_bos_eos(self.bos_char)
    #句子的结尾对应的char_ids
    self.eos_chars = _make_bos_eos(self.eos_char)
    #遍历id2word数组，得到每一个词的char_ids
    for i, word in enumerate(self._id_to_word):
        self._word_char_ids[i] = self._convert_word_to_char_ids(word)
    #将句子开头和结尾当作一个word处理
    self._word_char_ids[self.bos] = self.bos_chars
    self._word_char_ids[self.eos] = self.eos_chars
```

```
#返回word对应的char_ids数组
def word_to_char_ids(self, word):
    if word in self._word_to_id:
        return self._word_char_ids[self._word_to_id[word]]
    else:
        return self._convert_word_to_char_ids(word)

def encode_chars(self, sentence, reverse=False, split=True):
    '''
    Encode the sentence as a white space delimited string of tokens.
    对一整句话进行编码，编码成chars
    '''
    if split:                                                    #如果切割了句子
        chars_ids = [self.word_to_char_ids(cur_word)
                     for cur_word in sentence.split()]
    else:
        chars_ids = [self.word_to_char_ids(cur_word)
                     for cur_word in sentence]
    if reverse:
        return np.vstack([self.eos_chars] + chars_ids + [self.bos_chars]) #在每一条句子上都加了<eos>和<bos>
    else:
        return np.vstack([self.bos_chars] + chars_ids + [self.eos_chars])
```

```
    def _build_word_char_embeddings(self):
        # 设置初始的batchsize
        batch_size = self.options['batch_size']

        unroll_steps = self.options['unroll_steps']
        #最终维度,投影层维度
        projection_dim = self.options['lstm']['projection_dim']

        cnn_options = self.options['char_cnn']
        filters = cnn_options['filters']
        #一个卷积核抽取了一组特征，总的特征组数
        n_filters = sum(f[1] for f in filters)
        #每个词最大字符数
        max_chars = cnn_options['max_characters_per_token']
        # 映射成词向量的维度
        char_embed_dim = cnn_options['embedding']['dim']
        #字典中字符个数
        n_chars = cnn_options['n_characters']
        if n_chars != 261:
            raise InvalidNumberOfCharacters(
                    "Set n_characters=261 for training see the README.md"
            )
        if cnn_options['activation'] == 'tanh':
            activation = tf.nn.tanh
        elif cnn_options['activation'] == 'relu':
            activation = tf.nn.relu
```

```
        #字符的输入
        self.tokens_characters = tf.placeholder(DTYPE_INT,
                                    shape=(batch_size, unroll_steps, max_chars),
                                    name='tokens_characters')
        # 对字符进行词向量的嵌入
        with tf.device("/cpu:0"):
            self.embedding_weights = tf.get_variable(
                    "char_embed", [n_chars, char_embed_dim],
                    dtype=DTYPE,
                    initializer=tf.random_uniform_initializer(-1.0, 1.0)
            )
            # shape (batch_size, unroll_steps, max_chars, embed_dim)
            self.char_embedding = tf.nn.embedding_lookup(self.embedding_weights,
                                                         self.tokens_characters)
            ## 将字符反向，维下面的双向lstm作准备
            if self.bidirectional:
                self.tokens_characters_reverse = tf.placeholder(DTYPE_INT,
                                    shape=(batch_size, unroll_steps, max_chars),
                                    name='tokens_characters_reverse')
                self.char_embedding_reverse = tf.nn.embedding_lookup(
                    self.embedding_weights, self.tokens_characters_reverse)
```

```python
# the convolutions
def make_convolutions(inp, reuse):
    with tf.variable_scope('CNN', reuse=reuse) as scope:
        convolutions = []
        for i, (width, num) in enumerate(filters):
        # 优化方法
            if cnn_options['activation'] == 'relu':
                w_init = tf.random_uniform_initializer(
                    minval=-0.05, maxval=0.05)
            elif cnn_options['activation'] == 'tanh':
                w_init = tf.random_normal_initializer(
                    mean=0.0,
                    stddev=np.sqrt(1.0 / (width * char_embed_dim))
                )
            w = tf.get_variable(
                "W_cnn_%s" % i,
                [1, width, char_embed_dim, num],
                initializer=w_init,
                dtype=DTYPE)
            b = tf.get_variable(
                "b_cnn_%s" % i, [num], dtype=DTYPE,
                initializer=tf.constant_initializer(0.0))

            conv = tf.nn.conv2d(
                    inp, w,
                    strides=[1, 1, 1, 1],
                    padding="VALID") + b
            # now max pool
            conv = tf.nn.max_pool(
                    conv, [1, 1, max_chars-width+1, 1],
                    [1, 1, 1, 1], 'VALID')

            # activation
            conv = activation(conv)
            #将维数为1的维度去掉，保留数值数据。
            conv = tf.squeeze(conv, squeeze_dims=[2])

            convolutions.append(conv)

    return tf.concat(convolutions, 2)
```

```python
reuse = tf.get_variable_scope().reuse
embedding = make_convolutions(self.char_embedding, reuse)
self.token_embedding_layers = [embedding]
```

```python
#highway设计结构
def high(x, ww_carry, bb_carry, ww_tr, bb_tr):
    carry_gate = tf.nn.sigmoid(tf.matmul(x, ww_carry) + bb_carry)
    transform_gate = tf.nn.relu(tf.matmul(x, ww_tr) + bb_tr)
    return carry_gate * transform_gate + (1.0 - carry_gate) * x

if use_highway:
    highway_dim = n_filters

    for i in range(n_highway):
        with tf.variable_scope('CNN_high_%s' % i) as scope:
            W_carry = tf.get_variable(
                'W_carry', [highway_dim, highway_dim],
                # glorit init
                initializer=tf.random_normal_initializer(
                    mean=0.0, stddev=np.sqrt(1.0 / highway_dim)),
                dtype=DTYPE)
            b_carry = tf.get_variable(
                'b_carry', [highway_dim],
                initializer=tf.constant_initializer(-2.0),
                dtype=DTYPE)
            W_transform = tf.get_variable(
                'W_transform', [highway_dim, highway_dim],
                initializer=tf.random_normal_initializer(
                    mean=0.0, stddev=np.sqrt(1.0 / highway_dim)),
                dtype=DTYPE)
            b_transform = tf.get_variable(
                'b_transform', [highway_dim],
                initializer=tf.constant_initializer(0.0),
                dtype=DTYPE)

        embedding = high(embedding, W_carry, b_carry,
                         W_transform, b_transform)
        if self.bidirectional:
            embedding_reverse = high(embedding_reverse,
                                     W_carry, b_carry,
                                     W_transform, b_transform)
        self.token_embedding_layers.append(
            tf.reshape(embedding,
                [batch_size, unroll_steps, highway_dim])
        )
```

```python
# 进行投影操作
        if use_proj:
            embedding = tf.matmul(embedding, W_proj_cnn) + b_proj_cnn
            if self.bidirectional:
                embedding_reverse = tf.matmul(embedding_reverse, W_proj_cnn)
                    + b_proj_cnn
            self.token_embedding_layers.append(
                tf.reshape(embedding,
                        [batch_size, unroll_steps, projection_dim])
            )

        # 从新rshape成(batch_size, tokens, dim)
        if use_highway or use_proj:
            shp = [batch_size, unroll_steps, projection_dim]
            embedding = tf.reshape(embedding, shp)
            if self.bidirectional:
                embedding_reverse = tf.reshape(embedding_reverse, shp)
        self.embedding = embedding
        if self.bidirectional:
            self.embedding_reverse = embedding_reverse
```

Why？？

1.CNN can solve OOV (Out-of-Vocabulary) problem

2.CNN can pre-calculate each word vector and reduce the computational pressure in the inference phase

3.This sharing is reflected in the same set of parameters in the CNN used, which greatly reduces the size of the parameters.

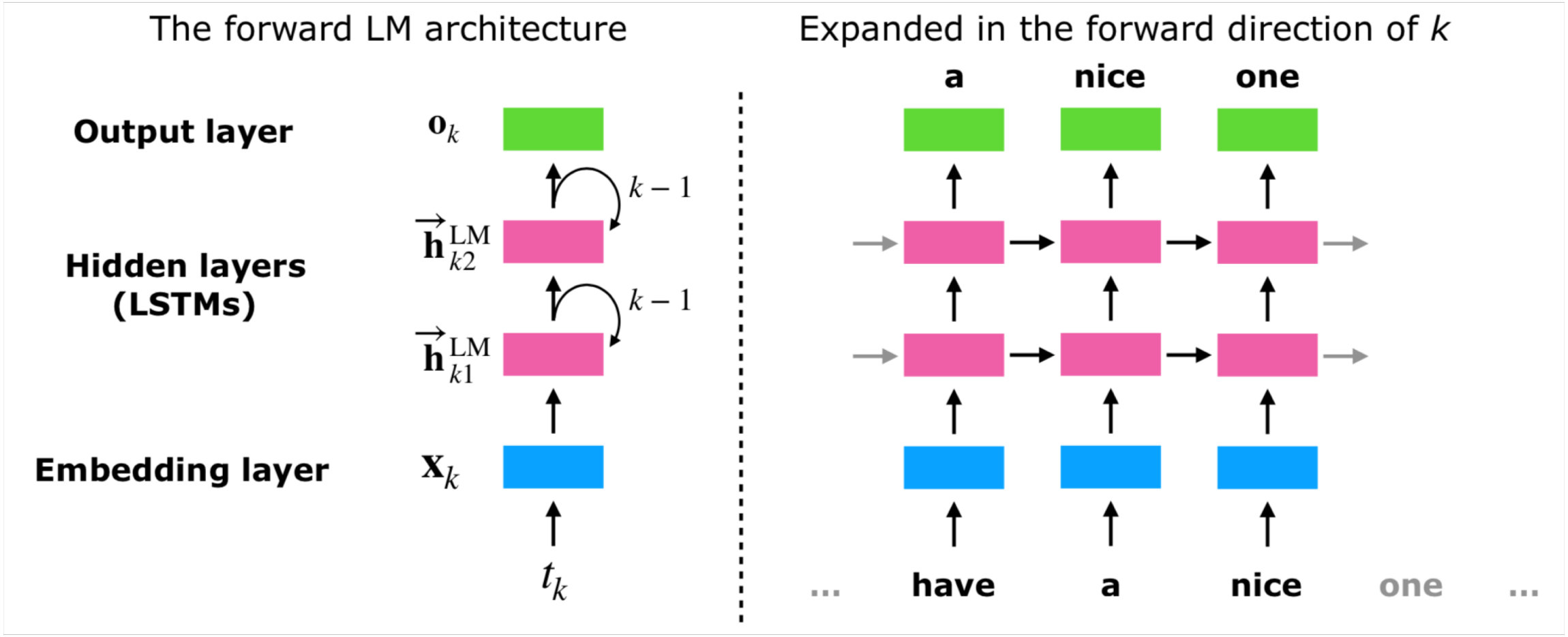
$$P(w_t|c_t) = \frac{exp(e'(w_t)^T x)}{\sum_{i=1}^{|V|} exp(e'(w_i)^T x)}, x = \sum_{i \in c} e(w_i)$$

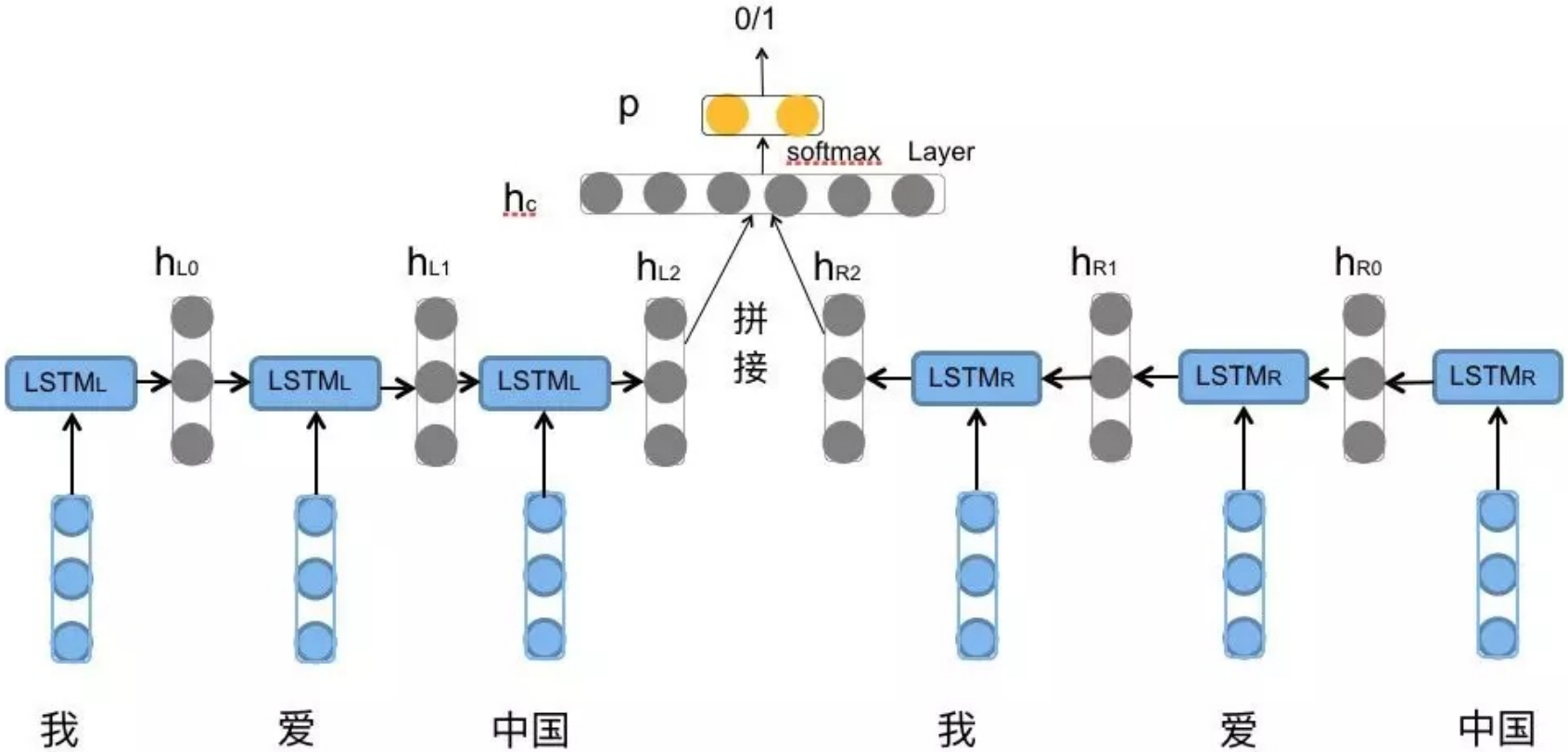
Word2vec Softmax

$$p(t_k|t_1, \ldots, t_{k-1}) = \frac{exp(CNN(t_k)^T h)}{\sum_{i=1}^{|V|} exp(CNN(t_i)^T h)}, h = LSTM(t_k|t_1, \ldots, t_{k-1})$$

CNN Softmax

Jozefowicz, Rafal, et al. "Exploring the limits of language modeling." *arXiv preprint arXiv:1602.02410* (2016).


The forward LM architecture
Output layer
Hidden layers (LSTMs)
Embedding layer
$\mathbf{o}_k$
$k-1$
$\overrightarrow{\mathbf{h}}^{LM}_{k2}$
$k-1$
$\overrightarrow{\mathbf{h}}^{LM}_{k1}$
$\mathbf{x}_k$
t_k
Expanded in the forward direction of k
a
nice
one
...
have
a
nice
one
...


0/1
p
softmax Layer
hc
hL0
hL1
hL2
hR2
hR1
hR0
拼接
LSTML
LSTML
LSTML
LSTMR
LSTMR
LSTMR
我
爱
中国
我
爱
中国

Why use this structure

- Kim, Yoon, et al. "Character-Aware Neural Language Models." *AAAI*. 2016.

- Jozefowicz R, Vinyals O, Schuster M, et al. “Exploring the limits of language modeling[J].” ICML2016

Different

- Allowing the model to train two-way parameters

- Increase the residual connection between LSTMs units

Bidirectional language models

Given a sequence of N tokens (t_1, t_2, ..., t_N):

- Forward language model: Predict token t_k from its history

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_1, t_2, ..., t_{k-1})$$

Many recent SotA neural language models compute a context-independent representation X_{kj}^{LM}, then pass it through L layers of LSTM, producing $\mathrm{h}_{\mathrm{kj}}^{LM}$, where j = 1,...,L.

Backward language model: Similar as above, but in reverse

$$p(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} p(t_k | t_{k+1}, t_{k+2}, ..., t_N)$$

A LSTM analogous to the forward model produces representations $\mathrm{h}_{\mathrm{kj}}^{LM}$

NOTE: direction

Bidirectional language model (biLM): Combines the forward and backward model.
Learn by maximizing:

$$\sum_{k=1}^{N} (\log p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s))$$

where:

- Θ_x: token representation parameters (shared between forward and backwards)
- Θ_s: softmax parameters (shared between forward and backwards)
- $\overrightarrow{\Theta}_{LSTM}$, $\overleftarrow{\Theta}_{LSTM}$: forward and backward LSTM parameters, respectively

The biLM produces 2L + 1 intermediate representations:

$$
\begin{aligned}
R_k =& \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\
=& \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\}
\end{aligned}
$$

where $\mathbf{h}_{k,0}^{LM} = \mathbf{x}_k^{LM}$ is the token layer and $\mathbf{h}_{k,j}^{LM} = [\overrightarrow{\mathbf{h}}_{k,j}^{LM}; \overleftarrow{\mathbf{h}}_{k,j}^{LM}]$, for each biLSTM layer.

ELMo: A task specific combination of these features:

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM} .$$

$$s_i^{task} = e^{s_i} / \sum_{j}^{N} e^{s_j}$$

where s^{task} are softmax-normalized weights and γ^{task} is a scaling parameter.

ELMo represents a word t_k as a linear combination of corresponding hidden layers

ELMo is a task specific representation. A down-stream task learns weighting parameters

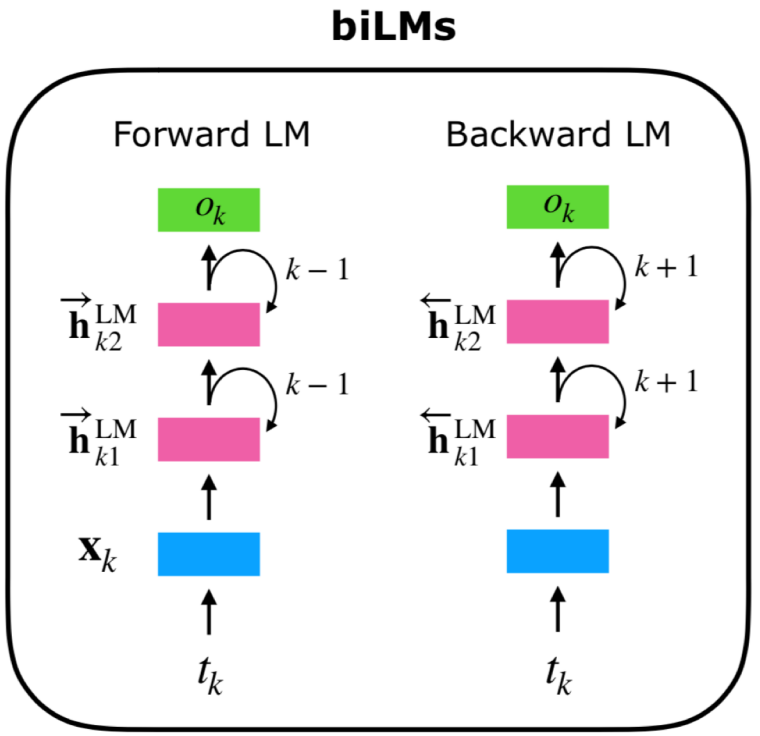
$$\mathbf{ELMo}_k^{\text{task}} = \gamma^{\text{task}} \times \sum \left\{ s_2^{\text{task}} \times \mathbf{h}_{k2}^{\text{LM}},\ s_1^{\text{task}} \times \mathbf{h}_{k1}^{\text{LM}},\ s_0^{\text{task}} \times \mathbf{h}_{k0}^{\text{LM}} \right.$$

$$\left. ([\mathbf{x}_k ; \mathbf{x}_k]) \right.$$

Concatenate hidden layers

$$[\overrightarrow{\mathbf{h}}_{kj}^{\text{LM}} ; \overleftarrow{\mathbf{h}}_{kj}^{\text{LM}}]$$

Unlike usual word embeddings, ELMo is assigned to every *token* instead of a *type*



Intuition

Different layers of deep Bi-LSTM encode different types of information.

- Higher-level LSTM states capture context-dependent aspects of word meaning (e.g., they can be used without modification to perform well on supervised word sense disambiguation tasks)
- lower level states model aspects of syntax(e.g., they can be used to do part-of-speech tagging).

The higher layer seemed to learn semantics while the lower layer probably captured syntactic features

Word sense disambiguation

Model	F_1
WordNet 1st Sense Baseline	65.9
Raganato et al. (2017a)	69.9
Iacobacci et al. (2016)	**70.1**
CoVe, First Layer	59.4
CoVe, Second Layer	*64.7*
biLM, First layer	67.4
biLM, Second layer	*69.0*

Table 5: All-words fine grained WSD F_1. For CoVe and the biLM, we report scores for both the first and second layer biLSTMs.

PoS tagging

Model	Acc.
Collobert et al. (2011)	97.3
Ma and Hovy (2016)	97.6
Ling et al. (2015)	**97.8**
CoVe, First Layer	*93.3*
CoVe, Second Layer	92.8
biLM, First Layer	*97.3*
biLM, Second Layer	96.8

Table 6: Test set POS tagging accuracies for PTB. For CoVe and the biLM, we report scores for both the first and second layer biLSTMs.

The higher layer seemed to learn semantics while the lower layer probably captured syntactic features**???**

Most models preferred "syntactic (probably)" features

Even in sentiment analysis

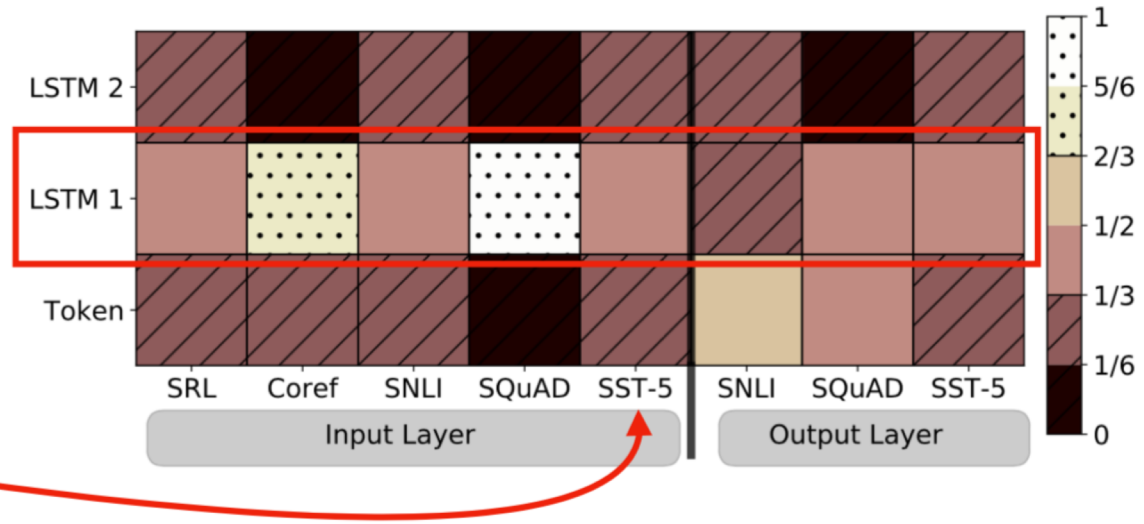


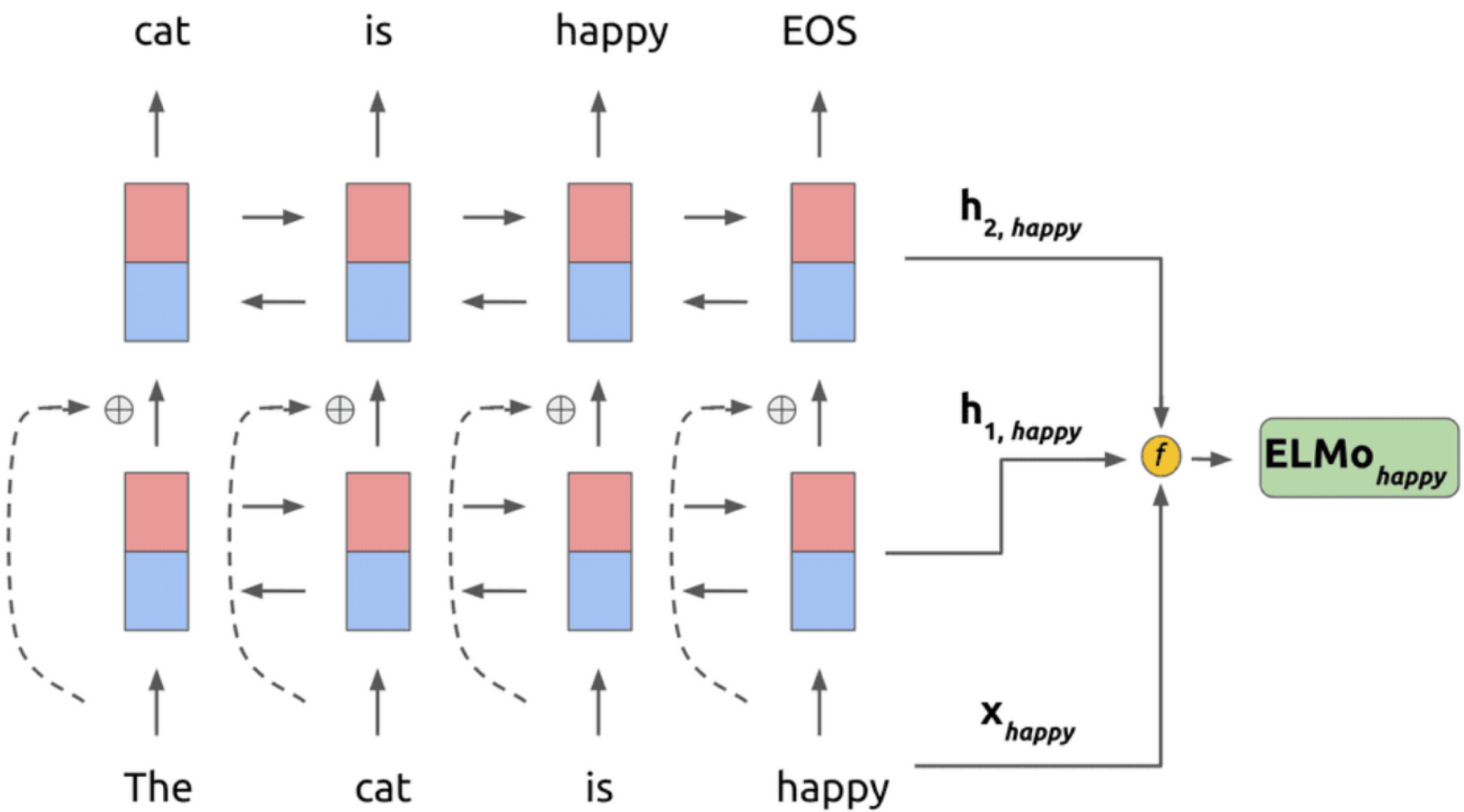
Figure 2: Visualization of softmax normalized biLM layer weights across tasks and ELMo locations. Normalized weights less then $1/3$ are hatched with horizontal lines and those greater then $2/3$ are speckled.

Using biLMs for supervised NLP tasks

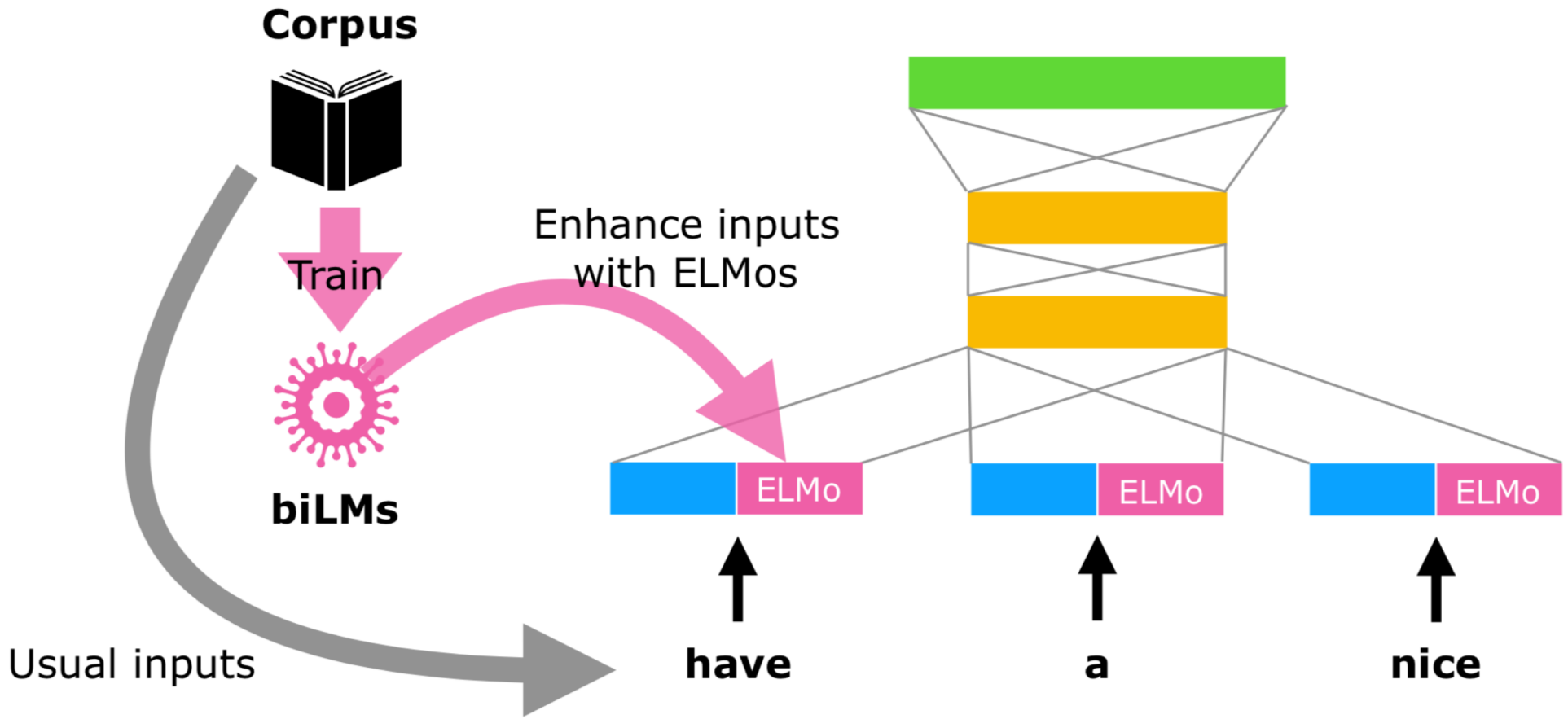
Given a pre-trained biLM, ELMo can easily be dropped into many supervised NLP architectures. Typical NLP architecture:

$$t_k \rightarrow \boldsymbol{x}_k \rightarrow \boldsymbol{h}_k$$

- To use ELMo, concatenate ELMo vector to token representation: [x ;ELMo^{task}]
- Occasionally also helpful to add to the task representation: [h ; ELMo^{task}]
- Moderate dropout to ELMo beneficial
- Occasionally also helpful to add regularization to the loss: $\lambda ||w||_2$


cat
is
happy
EOS
$h_{2,\ happy}$
$h_{1,\ happy}$
f
$ELMo_{happy}$
x_{happy}
The
cat
is
happy

ELMo can be integrated to almost all neural NLP tasks with simple concatenation to the embedding layer



Many linguistic tasks are improved by using ELMo

	Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
Q&A	SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
Textual entailment	SNLI	Chen et al. (2017)	88.6	88.0	88.7 $\pm$ 0.17	0.7 / 5.8%
Semantic role labelling	SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coreference resolution	Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
Named entity recognition	NER	Peters et al. (2017)	91.93 $\pm$ 0.19	90.15	92.22 $\pm$ 0.10	2.06 / 21%
Sentiment analysis	SST-5	McCann et al. (2017)	53.7	51.4	54.7 $\pm$ 0.5	3.3 / 6.8%

Table 1: Test set comparison of ELMo enhanced neural models with state-of-the-art single model baselines across six benchmark NLP tasks. The performance metric varies across tasks – accuracy for SNLI and SST-5; F_1 for SQuAD, SRL and NER; average F_1 for Coref. Due to the small test sizes for NER and SST-5, we report the mean and standard deviation across five runs with different random seeds. The "increase" column lists both the absolute and relative improvements over our baseline.

ELMo-enhanced models can make use of small datasets more efficiently

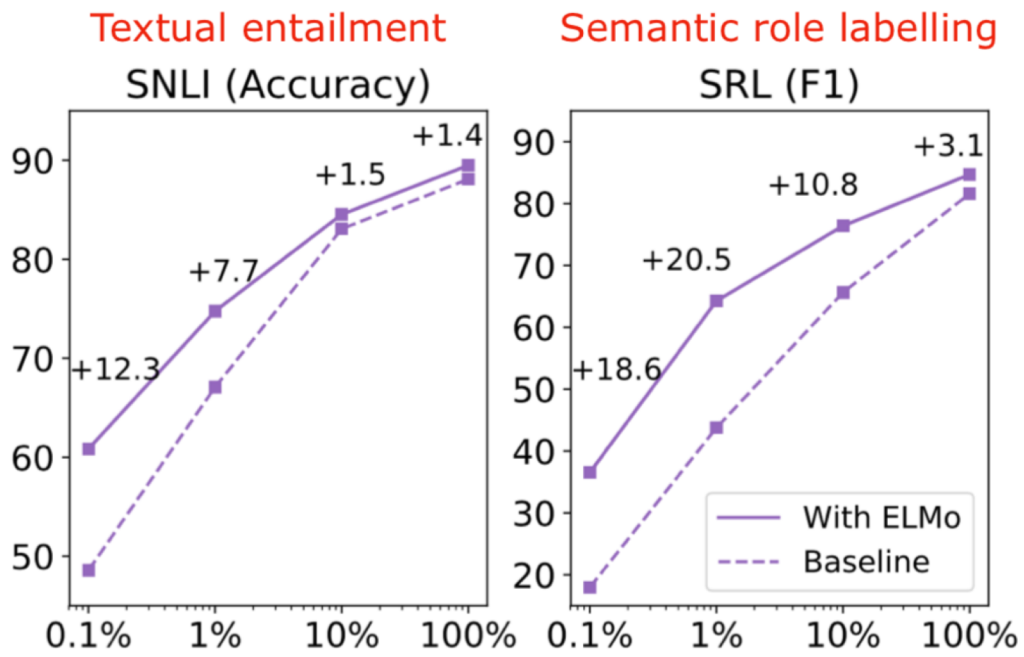


Figure 1: Comparison of baseline vs. ELMo performance for SNLI and SRL as the training set size is varied from 0.1% to 100%.

Continue…..

ELMO/ULMFit ->GPT -> BERT

1.Peters, Matthew E., et al. "Deep contextualized word representations." *arXiv preprint arXiv:1802.05365* (2018).

2. Howard, Jeremy, and Sebastian Ruder. "Universal language model fine-tuning for text classification." *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vol. 1. 2018.

3. Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/language-unsupervised/language_ understanding_paper. pdf*.

4. Devlin J, Chang M W, Lee K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv preprint arXiv:1810.04805, 2018.

Thank you !
Q&A