Lecture 7
Recurrent Neural Network

Lei Li and Yuxiang Wang
UCSB

Recap

- Building blocks
  - Convolution
  - Stride
  - Padding
  - Channel
  - Pooling
  - Dropout
  - Batch Norm
  - Residual connection
- Data Augmentation
- Deeper is better:
  - LeNet 5 layers of CNN
  - AlexNet
  - ResNet

Outline

- Language Modeling
- Recurrent Neural Network
- Long-short term memory network (LSTM)
- Gated Recurrent Unit (GRU)
- Attention
- Encoder-decoder framework
- LSTM Seq2seq

Why Learning Recurrent Neural Networks?

- a natural and succinct model for sequence data
- explicit modelling memory
- wide applications:
  - text classification
  - text generation
  - dialog response generation
  - time series prediction

Language Generation

- Given a sentence y, estimate the probability

$$P(y) = \prod_t P(y_{t+1} \mid y_1 \ldots y_t)$$

$$P(y_{t+1} \mid y_1 \ldots y_t) = f_\theta(y_1, \ldots, y_t)$$

The	cat	sits	on	a	__
y_1	y_2	y_3	y_4	y_5	y_6

$p(y_6 \mid y_1, \ldots, y_5)$

mat	0.15
rug	0.13
chair	0.08
hat	0.05
dog	0.01

Vocabulary

- To model P(y|x)
- Consider a ten-word sentence, chosen from common English dictionary about 5k words
  - 5000^{10} possible sentences
  - need a table of $5000^{10} \cdot 5000^{10}$ entries, infeasible
- source and target sentences need to break into smaller units.
- Multiple ways to segment
- Language specific considerations

Tokenization

- Break sentences into tokens, basic elements of processing
- Word-level Tokenization
  - Break by space and punctuation.
  - English, French, German, Spanish

The most eager is Oregon which is enlisting 5,000 drivers in the country's biggest experiment.

  - Special treatment: numbers replaced by special token [number]
  - How large is the Vocabulary? Cut-off by frequency, the rest replaced by [UNK]

Pros and Cons of Word-level Tokenization

- Easy to implement
- Cons:
  - Out-of-vocabulary (OOV) or unknown tokens, e.g. Covid
  - Tradeoff between parameters size and unknown chances.
    - Smaller vocab => fewer parameters to learn, easier to generate (deciding one word from smaller dictionary), more OOV
    - Larger vocab => more parameters to learn, harder to generate, less OOV
  - Hard for certain languages with continuous script: Japanese, Chinese, Korean, Khmer, etc. Need separate word segmentation tool (can be neural networks)

最热切的是俄勒冈州，该州正在招募 5,000 名司机参与该国最大的试验。

Character-level Tokenization

- Each letter and punctuation is a token

T	h	e	m	o	s	t	e	a	g	e	r	i	s	O	r	e	g	…

- Pros:
  – Very small vocabulary (except for some languages, e.g. Chinese)
  – No Out-of-Vocabulary token
- Cons:
  – A sentence can be longer sequence
  – Tokens do not representing semantic meaning

Subword-level Tokenization

- Goal:

  - moderate size vocabulary
  - no OOV
- Idea:
  - represent rare words (OOV) by sequence of subwords
- Byte Pair Encoding (BPE)
  - not necessarily semantic meaningful
  - Originally for data compression

Philip Gage. A New Algorithm for Data Compression, 1994

Byte Pair Encoding

- Use smallest sequence of strings to represent original string. Group frequent pair of bytes together.
- Put all characters into symbol table
- For each loop, until table reach size limit
  - count frequencies of symbol pair
  - replace most frequent pair with a new symbol, add to symbol table

Byte Pair Encoding (BPE) for Text Tokenization

1. Initialize vocabulary with all characters as tokens (also add end-of-word symbol) and frequencies
2. Loop until vocabulary size reaches capacity
    1. Count successive pairs of tokens in corpus
    2. Rank and select the top frequent pair
    3. Combine the pair to form a new token, add to vocabulary
3. Output final vocabulary and tokenized corpus

Rico Sennrich et al. Neural Machine Translation of Rare Words with Subword Units. 2016

Example

l, o, w, e, r, n, s, t, i, d, </w>	‘l o w</w>’ : 5 ‘l o w e r</w>’ : 2 ’n e w e s t</w>’ : 6 ‘w i d e s t</w>’ : 3
l, o, w, e, r, n, s, t, i, d, </w>, es	‘l o w</w>’ : 5 ‘l o w e r</w>’ : 2 ’n e w es t</w>’ : 6 ‘w i d es t</w>’ : 3
l, o, w, e, r, n, s, t, i, d, </w>, es, est	‘l o w</w>’ : 5 ‘l o w e r</w>’ : 2 ’n e w est</w>’ : 6 ‘w i d est</w>’ : 3
l, o, w, e, r, n, s, t, i, d, </w>, es, est, est</w>	‘l o w</w>’ : 5 ‘l o w e r</w>’ : 2 ’n e w est</w>’ : 6 ‘w i d est</w>’ : 3
l, o, w, e, r, n, s, t, i, d, </w>, es, est, est</w>, lo,	‘lo w</w>’ : 5 ‘lo w e r</w>’ : 2 ’n e w est</w>’ : 6 ‘w i d est</w>’ : 3
l, o, w, e, r, n, s, t, i, d, </w>, es, est, est</w>, lo, low	‘low</w>’ : 5 ‘low e r</w>’ : 2 ’n e w est</w>’ : 6 ‘w i d est</w>’ : 3

Predict Next Token Probability

There are many methods to predict the next token:

- N-gram: assuming

$$p\left(x_t \middle| x_1, \ldots, x_{t-1}\right) = p\left(x_t \middle| x_{t-k}, \ldots, x_{t-1}\right)$$

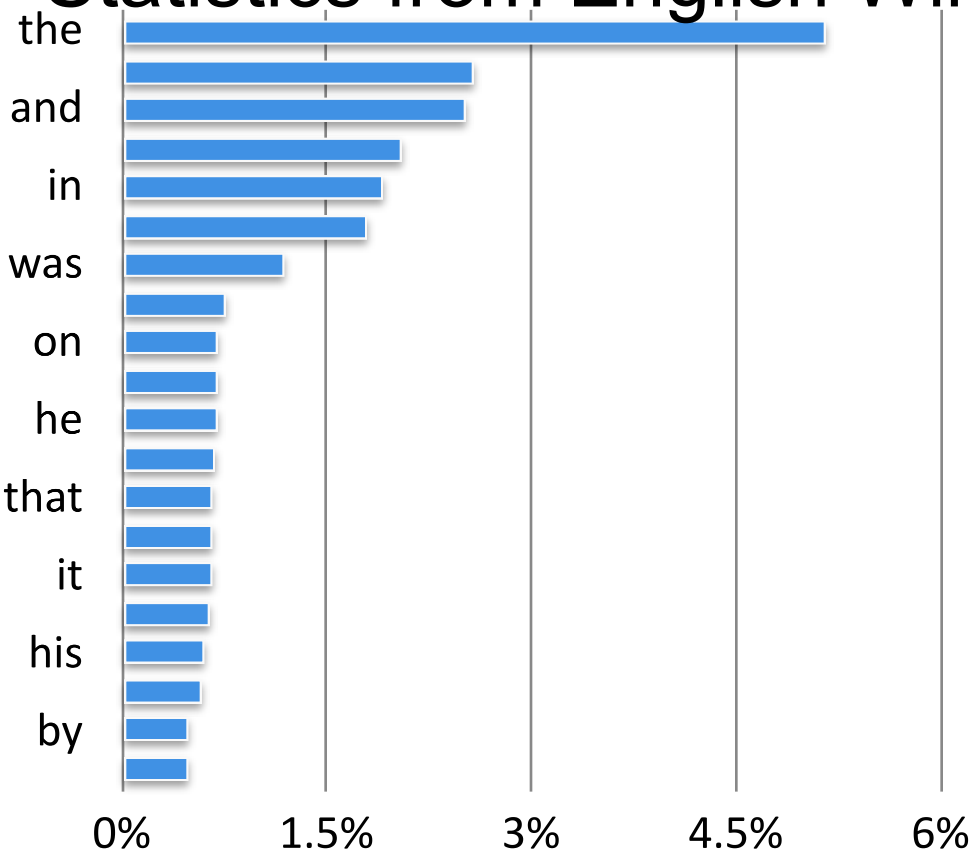
  , and estimate it directly
- Context MLP: use DNN to estimate

$$p\left(x_t \middle| x_{t-k}, \ldots, x_{t-1}\right)$$

- CNN-LM (previous lecture)
- RNN-LM, LSTM, GRU
- GPT

Word and Bigram

Statistics from English Wikipedia and books



cond. prob. $p(x_2|x_1)$

	first	united	the	a	be
the	0.014	0.006			
of			0.283	0.030	
would					0.191
with			0.187	0.122	

Challenge of n-gram LM

- Vocabulary: V
- n-gram needs a probability table of size V^n
- Common V size 30k ~ 100k
- Hard to estimate and hard to generalize
- Solution: Parameterization with generative model
  - $p(y_t \mid y_1, \cdots, y_{t-1}; \theta) = f_\theta(y_1, \cdots, y_{t-1})$
  - f can be a carefully designed and computationally tractable function, e.g. a neural network (later lectures).

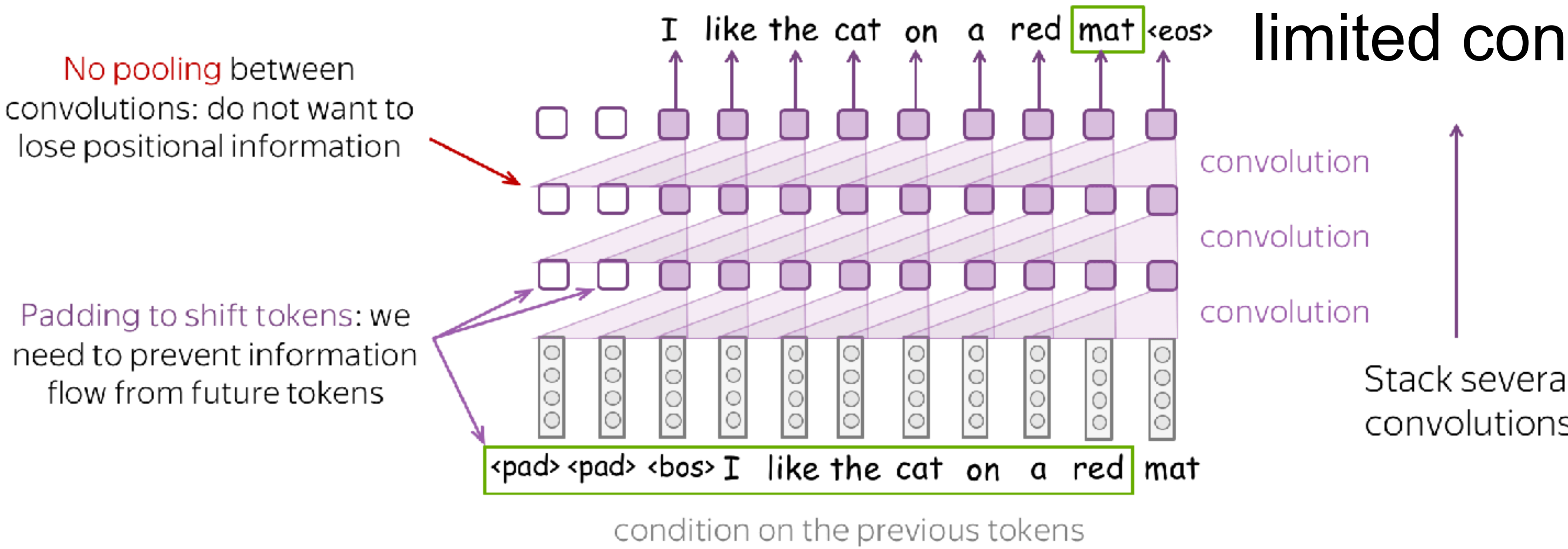
CNN Language Model

$$P(y_{t+1} \,|\, y_1, \ldots, y_t) \approx \text{CNN}_\theta(y_{t-k}, \ldots, y_t)$$

predict the next token

But,

limited cont



https://lena-voita.github.io/nlp_course/models/convolutional.html

Limitation of CNN-LM

- CNN-LM only has a fixed-length receptive field
  - probability of next token only dependent on a fixed-size context
- But sentences are of variable length
- How to handle sentences with variable length?
- Idea:
  - adding memory to network
  - adaptive updating memory

Recurrent Memory

- Introduce memory representation
- RNN-LM: use RNN to estimate

$$p\left(x_t \,\middle|\, x_1, \ldots, x_{t-1}\right) = \text{softmax}(W \cdot h_t)$$

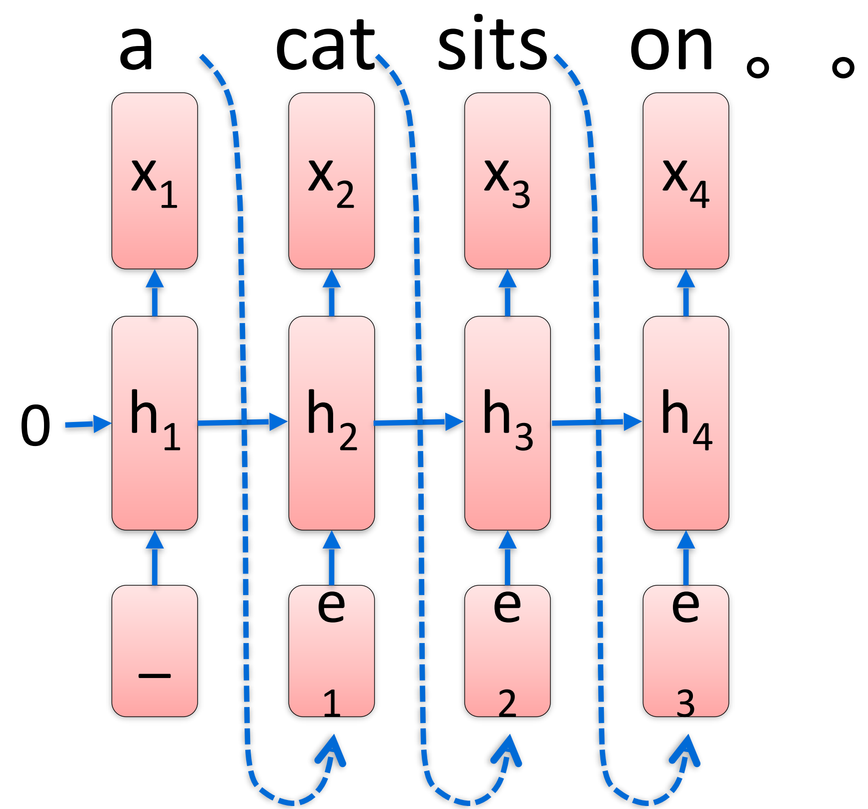
$$h_t = RNN(h_{t-1}, Emb(x_{t-1}))$$

- RNN cell can be
  - Simple feedforward neural network
  - Long-short term memory
  - Gated recurrent units

Recurrent Neural Network

$$p(x_t \mid x_1, \ldots, x_{t-1}) = \text{softmax}(U \cdot h_t)$$

$$h_t = \sigma\left(W \cdot \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} + b \right)$$



Elman, Finding Structure in Time. Cog. Sci. 1990.

Mikolov et al, Recurrent neural network based language model. Interspeech 2010.

Training RNN-LM

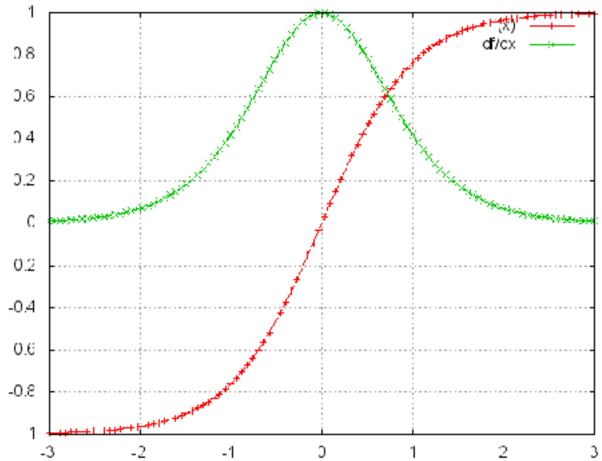
- Empirical Risk:
  - Loss: cross-entropy for every next-token given prefix context
  - CE(x_t+1, f(x_1, …, x_t))
- SGD
  - Calculate gradient: Back-propogation through time (BPTT)
  - ∇E_t

Back-propagation for RNN (python)

```
def bptt(self, x, y):
    T = len(y)
    # Perform forward propagation
    o, s = self.forward_propagation(x)
    # We accumulate the gradients in these variables
    dLdU = np.zeros(self.U.shape)
    dLdV = np.zeros(self.V.shape)
    dLdW = np.zeros(self.W.shape)
    delta_o = o
    delta_o[np.arange(len(y)), y] -= 1.
    # For each output backwards...
    for t in np.arange(T)[::-1]:
        dLdV += np.outer(delta_o[t], s[t].T)
        # Initial delta calculation: dL/dz
        delta_t = self.V.T.dot(delta_o[t]) * (1 - (s[t] ** 2))
        # Backpropagation through time (for at most self.bptt_truncate steps)
        for bptt_step in np.arange(max(0, t-self.bptt_truncate), t+1)[::-1]:
            # Add to gradients at each previous step
            dLdW += np.outer(delta_t, s[bptt_step-1])
            dLdU[:,x[bptt_step]] += delta_t
            # Update delta for next step dL/dz at t-1
            delta_t = self.W.T.dot(delta_t) * (1 - s[bptt_step-1] ** 2)
    return [dLdU, dLdV, dLdW]
```

Computational Issue: Gradient Vanishing

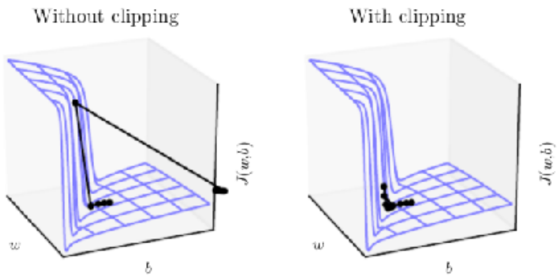
- tanh has derivative close to zero at both ends



Pascanu et al. On the difficulty of training recurrent neural networks. ICML 2013

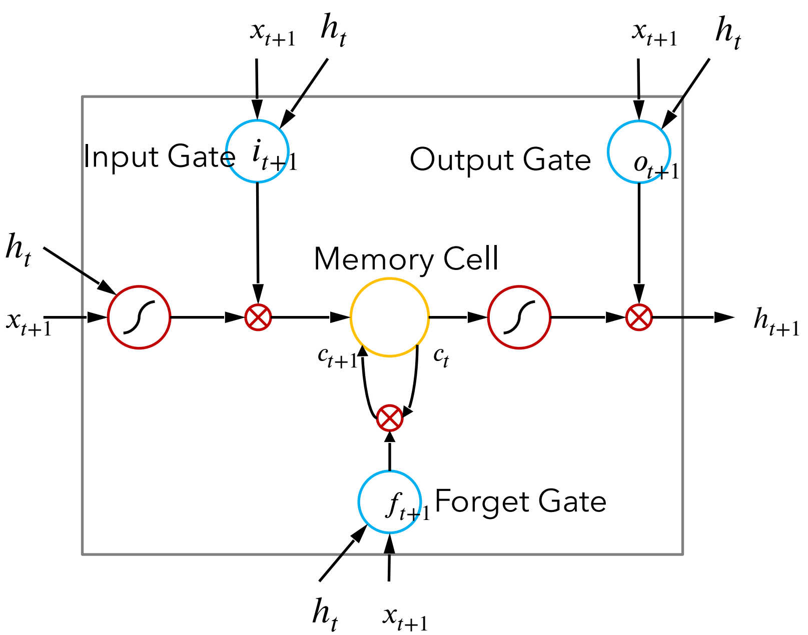
Gradient Exploding

- Use gradient clipping
- Two options: clip by absolute value or rescale norm
- if $|g| > \eta, \hat{g} \leftarrow \eta$
- if $|g| > \eta, \hat{g} \leftarrow \dfrac{\eta}{|g|} g$



Long-Short Term Memory (LSTM)

- Replace cell with more advanced one
- Adaptively memorize short and long term information



$$i_{t+1} = \sigma(M_{ix}x_{t+1} + M_{ih}h_t + b_i)$$
$$f_{t+1} = \sigma(M_{fx}x_{t+1} + M_{fh}h_t + b_f)$$
$$o_{t+1} = \sigma(M_{ox}x_{t+1} + M_{oh}h_t + b_o)$$

$$a_{t+1} = \tanh(M_{cx}x_{t+1} + M_{ch}h_t + b_a)$$

$$c_{t+1} = f_{t+1} \otimes c_t + i_{t+1} \otimes a_{t+1}$$
$$h_{t+1} = o_{t+1} \otimes \tanh(c_{t+1})$$

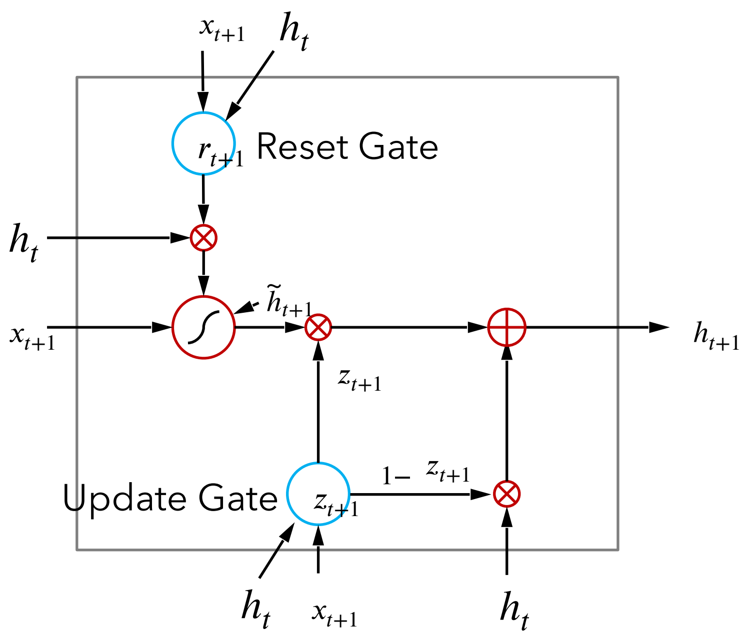
Hochreiter & Schmidhuber. Long Short-Term Memory, 1997
Gers et al. Learning to Forget: Continual Prediction with LSTM. 2000

Jürgen Schmidhuber

- Many fundamental contributions in deep learning, esp. LSTM
- heavily influence deepmind through his students

Gated Recurrent Unit (GRU)

- Adaptively memorize short and long term information
- like LSTM, but fewer parameters



Input: x_t

Memory: h_t

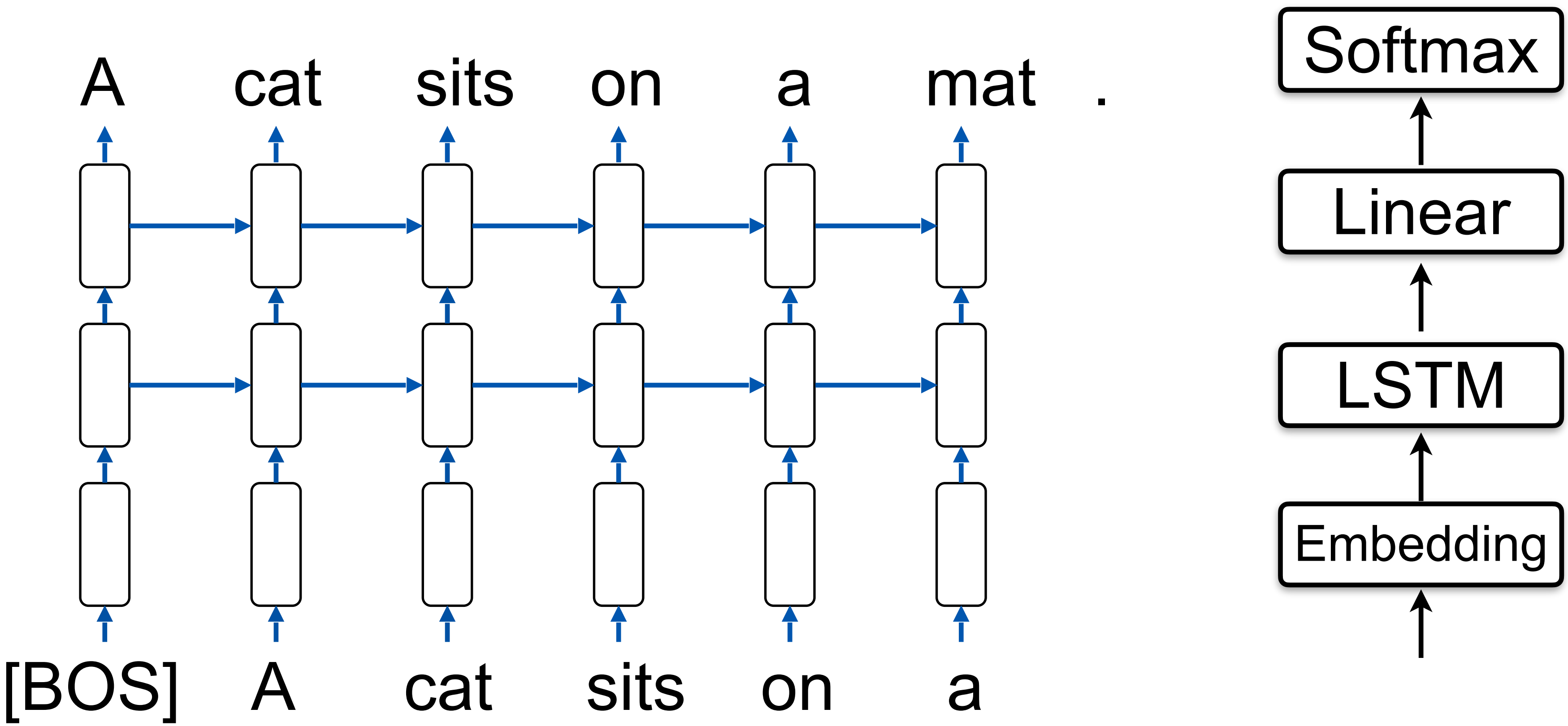
$$r_{t+1} = \sigma(M_{rx}x_{t+1} + M_{rh}h_t + b_r)$$
$$z_{t+1} = \sigma(M_{zx}x_{t+1} + M_{zh}h_t + b_z)$$

$$\tilde{h}_{t+1} = \tanh(M_{hx}x_{t+1} + M_{hh}(r_{t+1} \otimes h_t) + b_h)$$

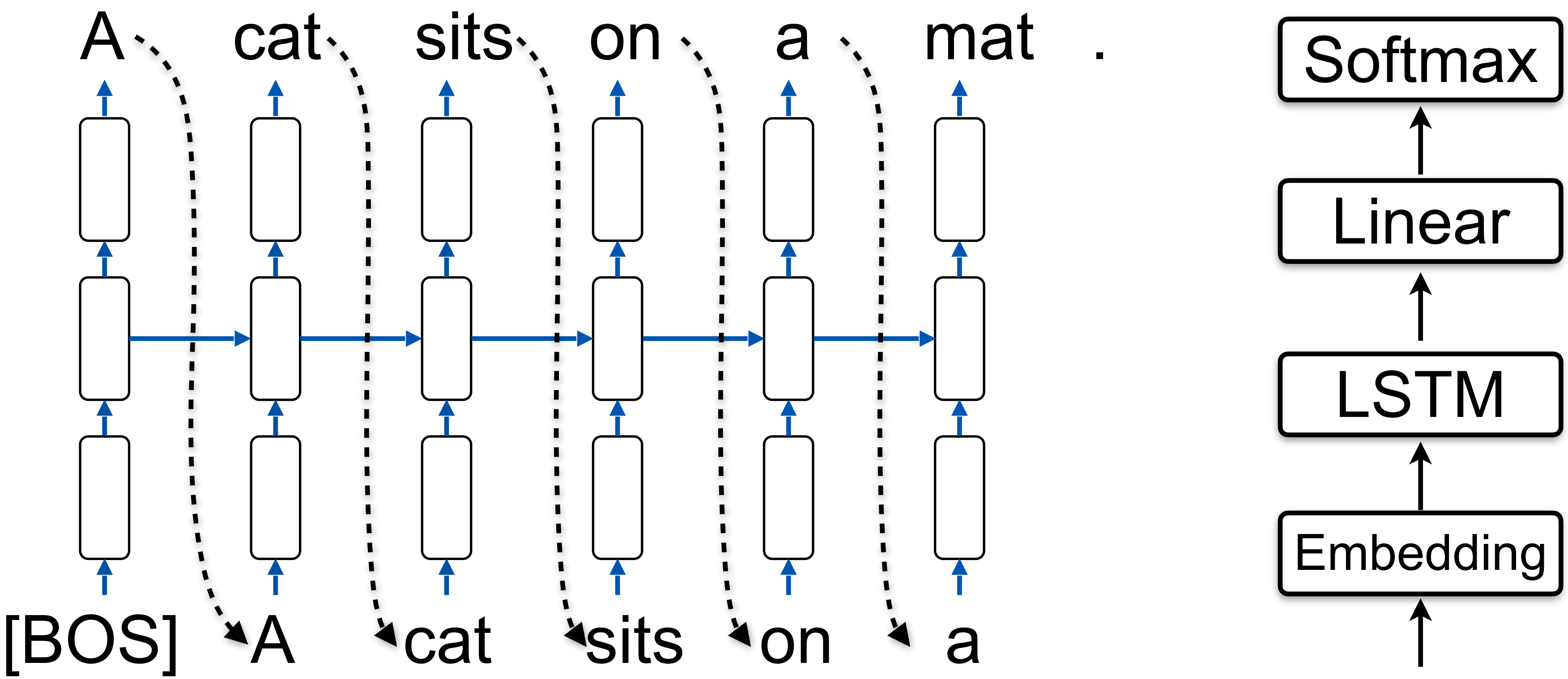
$$h_{t+1} = z_{t+1} \otimes \tilde{h}_{t+1} + (\mathbf{1} - z_{t+1}) \otimes h_t$$

Cho et al. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. 2014

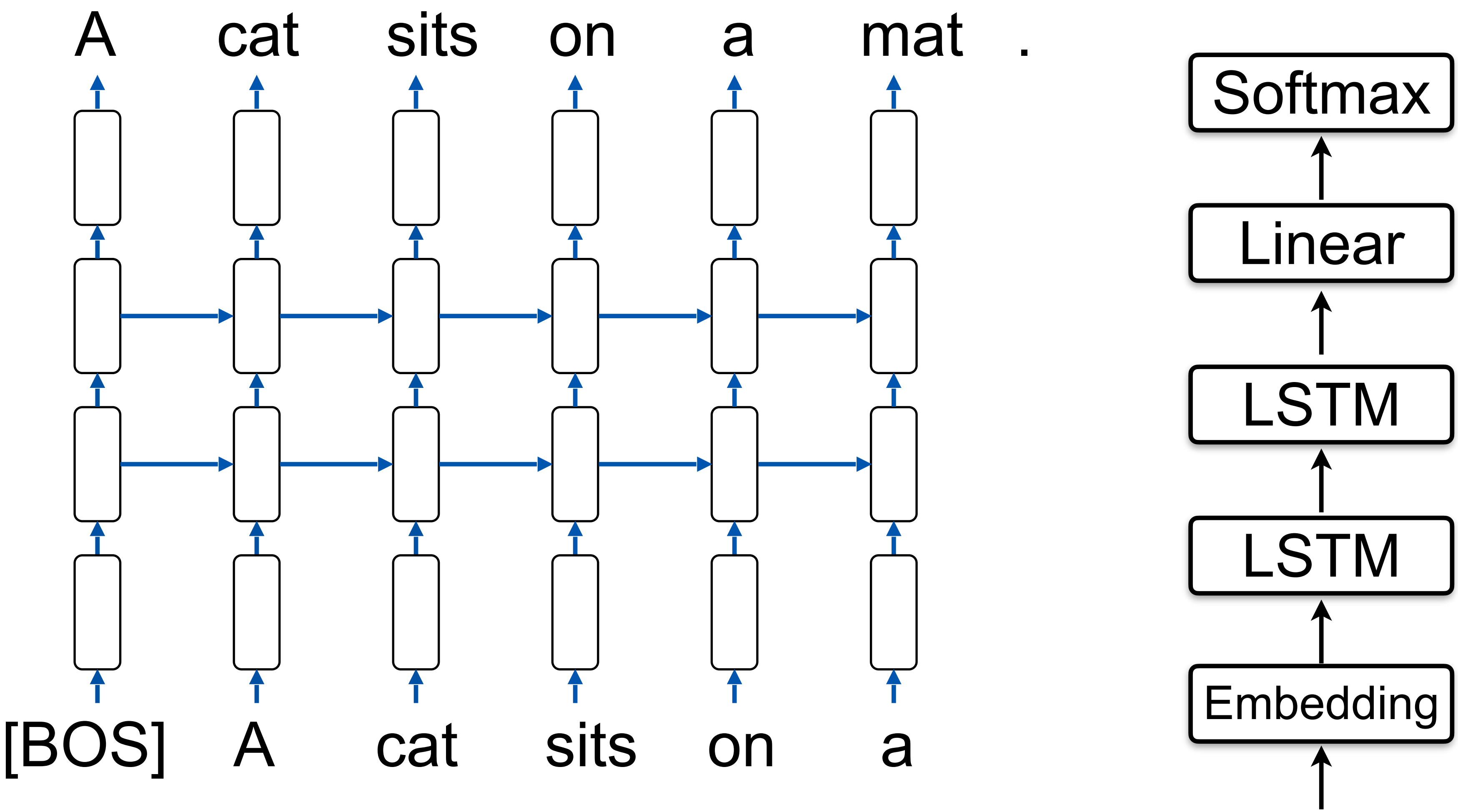
LSTM Language Modelling



LSTM Generation



LSTM: More layers



Expressive Power of RNN-LM

Perplexity:

$$PPL = P(x_1, \ldots, x_N)^{-\frac{1}{N}} = \exp(-\frac{1}{N}\sum_{n=1}^{N} \log P(x_n \,|\, x_1 \ldots x_{n-1}))$$

Model	Test Perplexity	Number of Params [billions]
Sigmoid-RNN-2048 (Ji et al., 2015a)	68.3	4.1
Interpolated KN 5-gram, 1.1B n-grams (Chelba et al., 2013)	67.6	1.76
Sparse Non-Negative Matrix LM (Shazeer et al., 2015)	52.9	33
RNN-1024 + MaxEnt 9-gram features (Chelba et al., 2013)	51.3	20
LSTM-512-512	54.1	0.82
LSTM-1024-512	48.2	0.82
LSTM-2048-512	43.7	0.83
LSTM-8192-2048 (No Dropout)	37.9	3.3
LSTM-8192-2048 (50% Dropout)	32.2	3.3
2-Layer LSTM-8192-1024 (BIG LSTM)	30.6	1.8
BIG LSTM+CNN Inputs	**30.0**	**1.04**
BIG LSTM+CNN Inputs + CNN Softmax	39.8	**0.29**
BIG LSTM+CNN Inputs + CNN Softmax + 128-dim correction	35.8	**0.39**
BIG LSTM+CNN Inputs + Char LSTM predictions	47.9	**0.23**

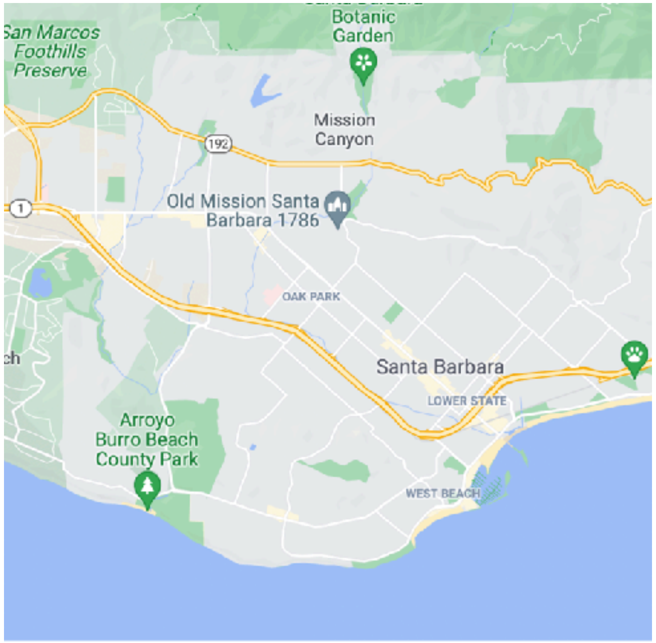
Jozefowicz et al. Exploring the limits of language modelling, 2016

Sequence Labelling

Understanding Query Intention

Noodle house near Santa Barbara
[Keyword] [Location]

How to go from Santa Barbara to Log Angeles ?
[Origin] [Destination]



Sequence Labelling

Named entity recognition

date   Location

In **April 1775** fighting broke out between **Massachusetts** militia units and **British** regulars at **Lexington** and **Concord** .

Geo-Political

Sequence Labelling

- Named entity recognition

  In **April 1775** fighting broke out between **Massachusetts** militia units and **British** regulars at **Lexington** and **Concord** .

- Semantic role labeling

The excess supply pushed gasoline prices down in that period .

subject verb object

- Question Answering: subject parsing

  Who created Harry Potter ?

Represent the Output Labels

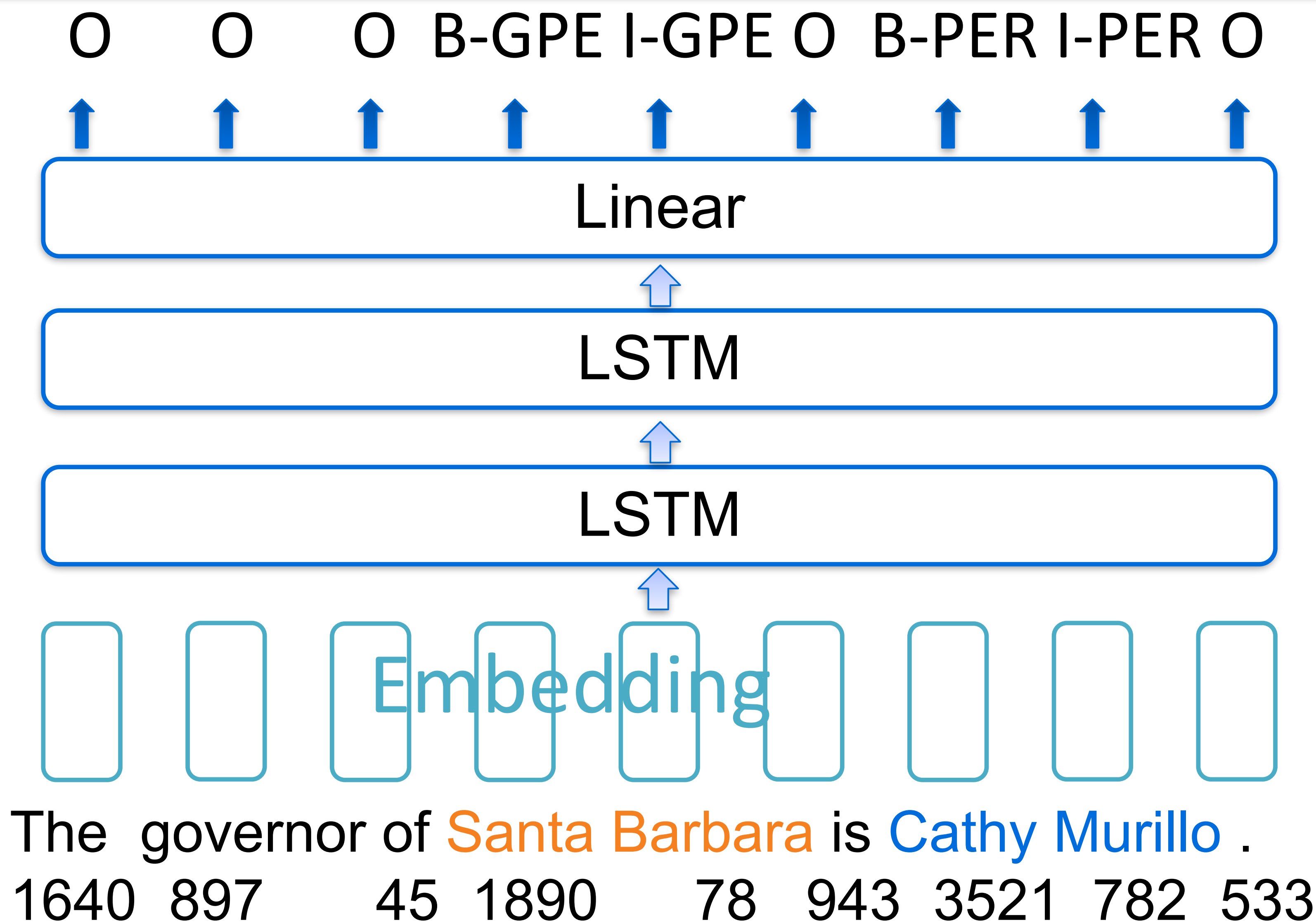
- BIO scheme

O O O B-GPE I-GPE O B-PER I-PER O

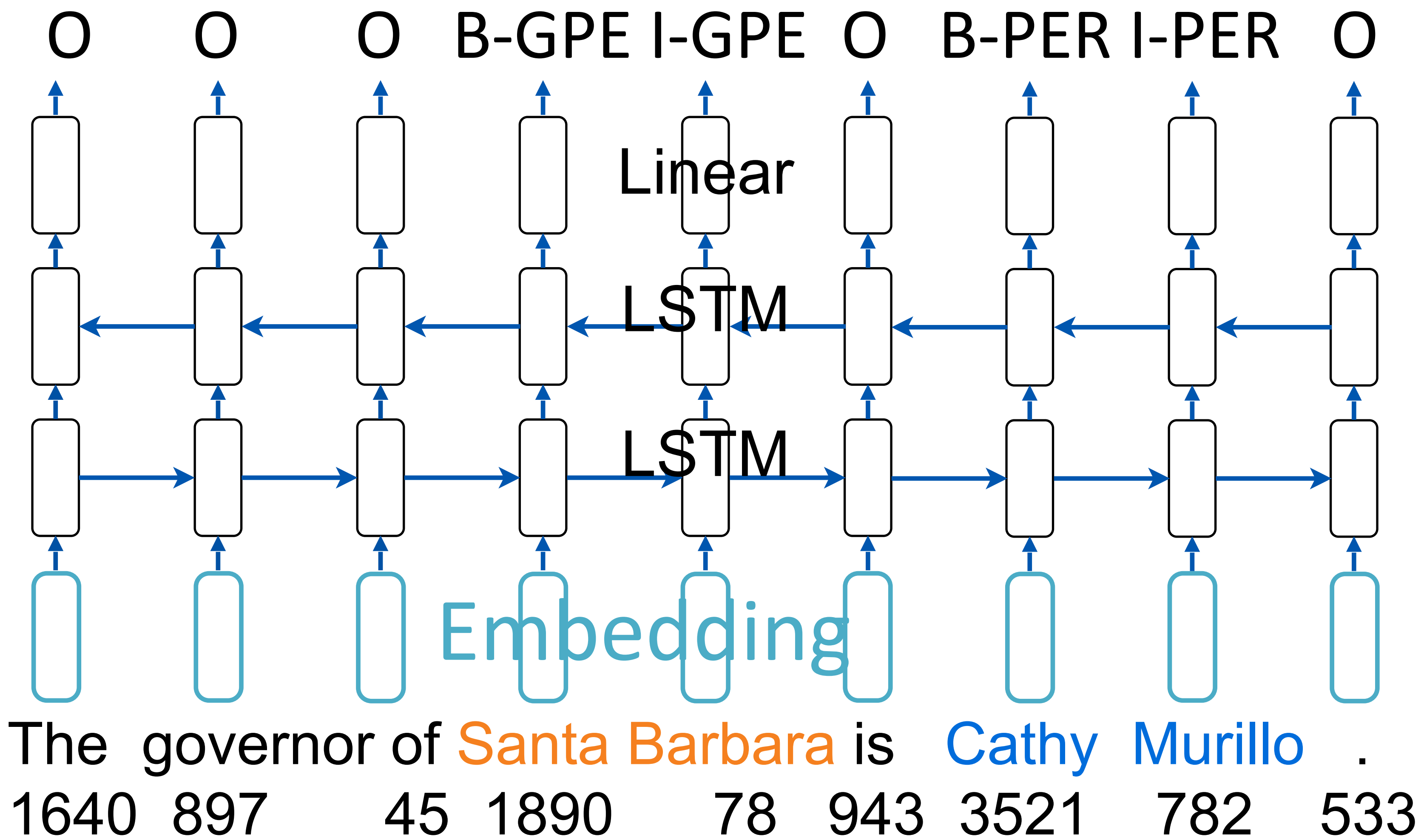
The governor of Santa Barbara is Cathy Murillo .

1640 897 45 1890 78 943 3521 782 533

RNN/LSTM for Sequence Labelling



Bi-LSTM


O O O B-GPE I-GPE O B-PER I-PER O
Linear
LSTM
LSTM
Embedding
The governor of Santa Barbara is Cathy Murillo .
1640 897 45 1890 78 943 3521 782 533

Sequence Labelling using LSTM (Pytorch)

```
class LSTMTagger(nn.Module):

    def __init__(self, embedding_dim, hidden_dim, vocab_size, tagset_size):
        super(LSTMTagger, self).__init__()
        self.hidden_dim = hidden_dim

        self.word_embeddings = nn.Embedding(vocab_size, embedding_dim)

        # The LSTM takes word embeddings as inputs, and outputs hidden states
        # with dimensionality hidden_dim.
        self.lstm = nn.LSTM(embedding_dim, hidden_dim)

        # The linear layer that maps from hidden state space to tag space
        self.hidden2tag = nn.Linear(hidden_dim, tagset_size)

    def forward(self, sentence):
        embeds = self.word_embeddings(sentence)
        lstm_out, _ = self.lstm(embeds.view(len(sentence), 1, -1))
        tag_space = self.hidden2tag(lstm_out.view(len(sentence), -1))
        tag_scores = F.log_softmax(tag_space, dim=1)
        return tag_scores
```

Training in Pytorch

```
model = LSTMTagger(EMBEDDING_DIM, HIDDEN_DIM, len(word_to_ix), len(tag_to_ix))
loss_function = nn.NLLLoss()
optimizer = optim.SGD(model.parameters(), lr=0.1)

# See what the scores are before training
# Note that element i,j of the output is the score for tag j for word i.
# Here we don't need to train, so the code is wrapped in torch.no_grad()
with torch.no_grad():
    inputs = prepare_sequence(training_data[0][0], word_to_ix)
    tag_scores = model(inputs)
    print(tag_scores)

for epoch in range(300):  # again, normally you would NOT do 300 epochs, it is toy data
    for sentence, tags in training_data:
        # Step 1. Remember that Pytorch accumulates gradients.
        # We need to clear them out before each instance
        model.zero_grad()

        # Step 2. Get our inputs ready for the network, that is, turn them into
        # Tensors of word indices.
        sentence_in = prepare_sequence(sentence, word_to_ix)
        targets = prepare_sequence(tags, tag_to_ix)

        # Step 3. Run our forward pass.
        tag_scores = model(sentence_in)

        # Step 4. Compute the loss, gradients, and update the parameters by
        #  calling optimizer.step()
        loss = loss_function(tag_scores, targets)
        loss.backward()
        optimizer.step()
```

Testing in Pytorch

```
# See what the scores are after training
with torch.no_grad():
    inputs = prepare_sequence(training_data[0][0], word_to_ix)
    tag_scores = model(inputs)
```

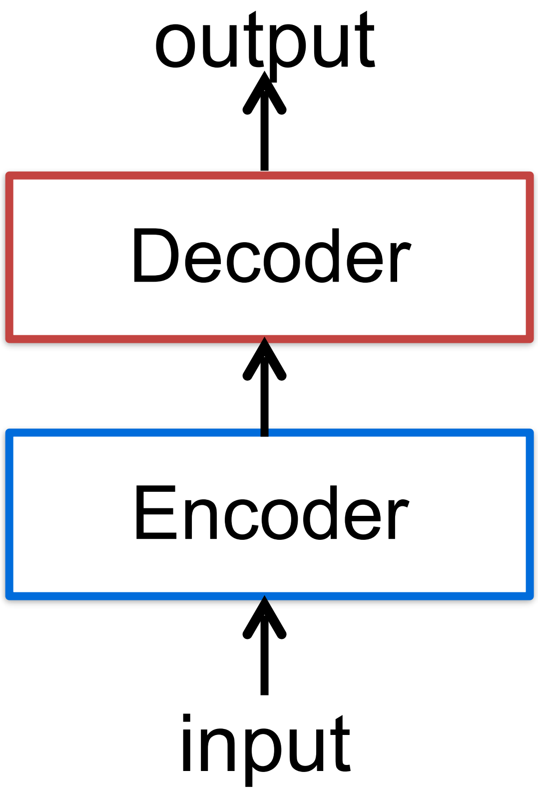
Better Loss Function (advanced)

- Loss using Conditional Random Fields

$$
\begin{aligned}
-\log(P(\mathbf{y} \mid \mathbf{X})) &= -\log\left(\frac{\exp\left(\sum_{k=1}^{\ell} U(\mathbf{x}_k, y_k) + \sum_{k=1}^{\ell-1} T(y_k, y_{k+1})\right)}{Z(\mathbf{X})}\right) \\
&= \log\left(Z(\mathbf{X})\right) - \log\left(\exp\left(\sum_{k=1}^{\ell} U(\mathbf{x}_k, y_k) + \sum_{k=1}^{\ell-1} T(y_k, y_{k+1})\right)\right) \\
&= \log\left(Z(\mathbf{X})\right) - \left(\sum_{k=1}^{\ell} U(\mathbf{x}_k, y_k) + \sum_{k=1}^{\ell-1} T(y_k, y_{k+1})\right) \\
&= Z_{\log}(\mathbf{X}) - \left(\sum_{k=1}^{\ell} U(\mathbf{x}_k, y_k) + \sum_{k=1}^{\ell-1} T(y_k, y_{k+1})\right)
\end{aligned}
$$

will revisit in graphical models lecture

Encoder-decoder framework



A generic formulation

ImageCaption

Text-to-Image Generation

ASR (speech-to-text)

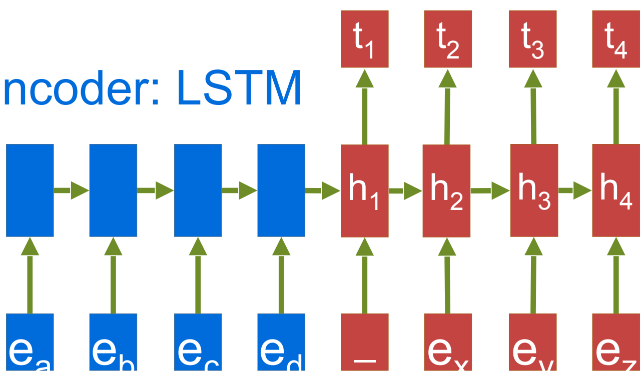
MT (text-to-text)

Sequence To Sequence (Seq2seq)

- Machine translation as directly learning a function mapping from source sequence to target sequence

target:
The weather is nice

Encoder: LSTM



Source: 天 气 很 好

Decoder: LSTM

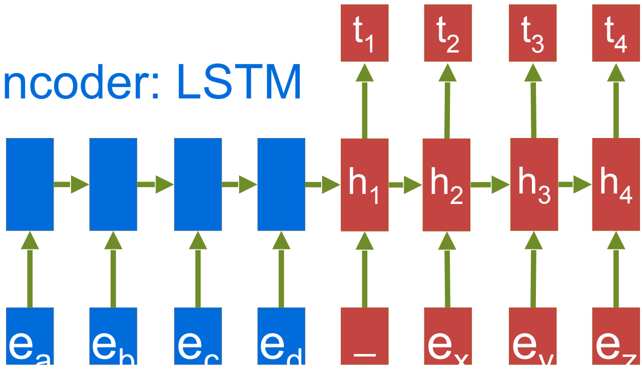
Sutskever et al. Sequence to Sequence Learning with Neural Networks. 2014

Sequence To Sequence (Seq2seq)

- Machine translation as directly learning a function mapping from source sequence to target sequence

target:
The weather is nice

Encoder: LSTM



Source: 天 气 很 好

Decoder: LSTM

$$P(Y|X) = \prod P(y_t | y_{<t}, x)$$

Training loss: Cross-Entropy

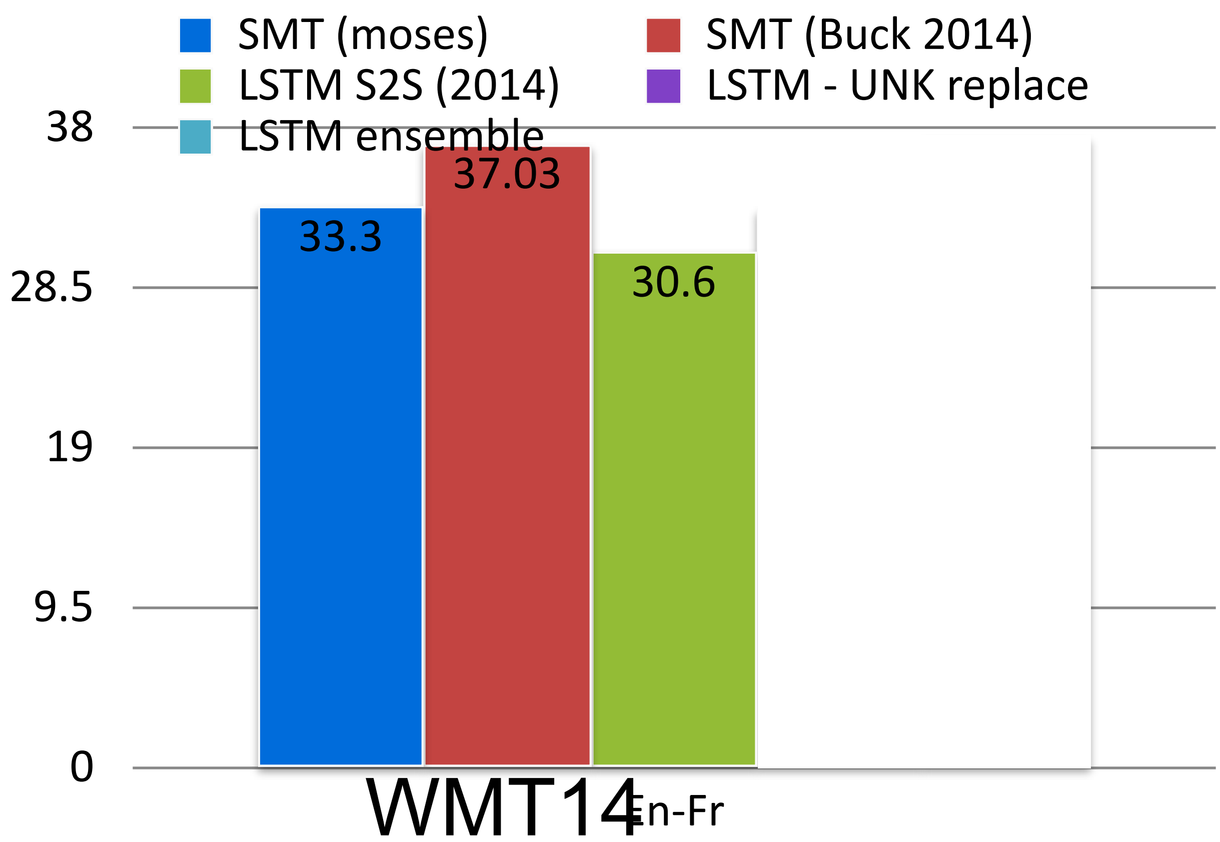
$$l = -\sum_n \sum_t \log f_\theta(x_n, y_{n,1}, \ldots, y_{n,t-1})$$

Teacher-forcing during training.

(pretend to know groundtruth for prefix)

Sutskever et al. Sequence to Sequence Learning with Neural Networks. 2014

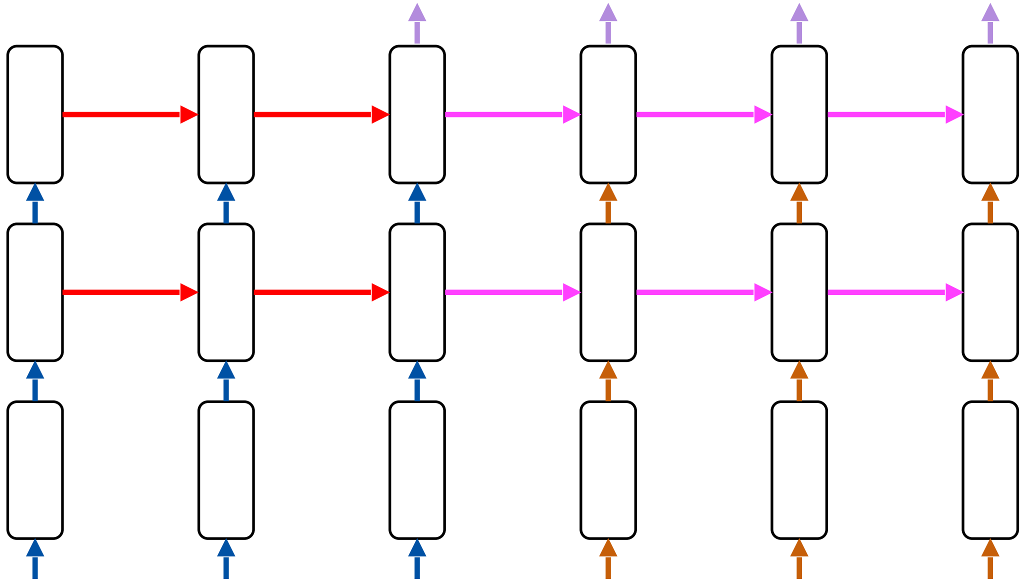
Performance (2014)



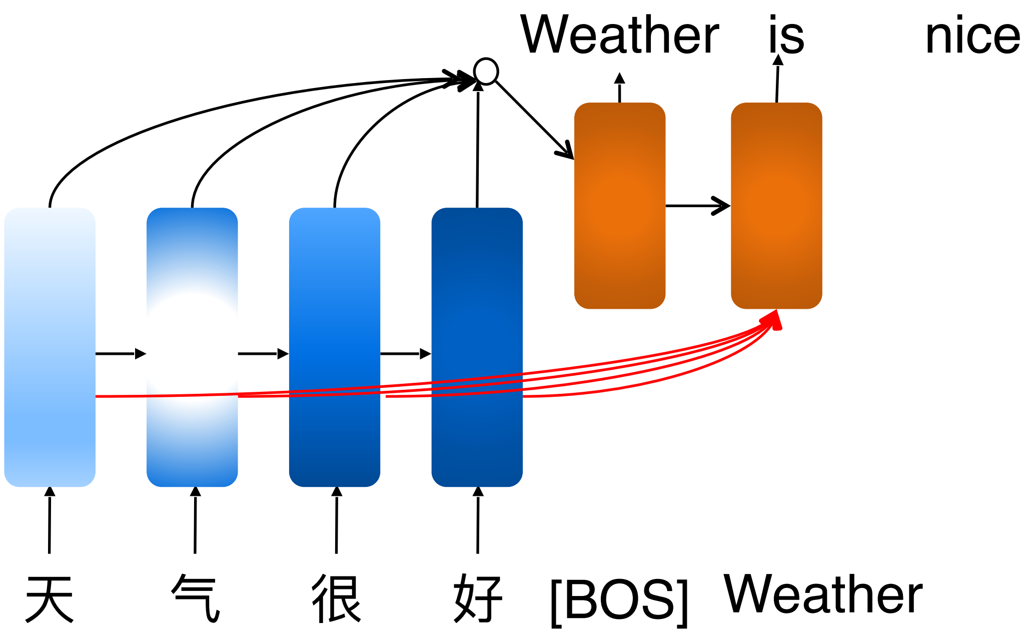
Sutskever et al. Sequence to Sequence Learning with Neural Networks. 2014
Durrani et al. Edinburgh's Phrase-based Machine Translation Systems for WMT-14. 201

Stacked LSTM for seq-2-seq

- More layers of LSTM

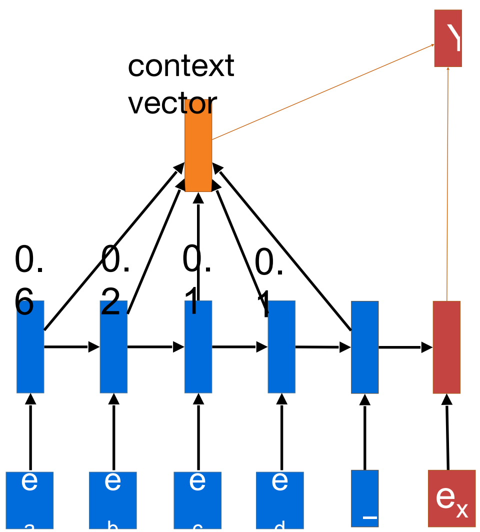


LSTM Seq2seq with Attention



Bahdanau et al., Neural Machine Translation by Jointly Learning to Align and Translate 2015

Generation by Attention



A context vector c will be predicted before, which represents the related source context for current predicted word.

$$\alpha_{nj} = \mathrm{Softmax}(D(s_n, h_{1\ldots n-1})) = \frac{\exp(D(s_n, h_j))}{\sum_k \exp(D(s_n, h_k))}$$

$$c_n = \sum_j \alpha_{nj} h_j$$

The probability of word y_i is computed as:

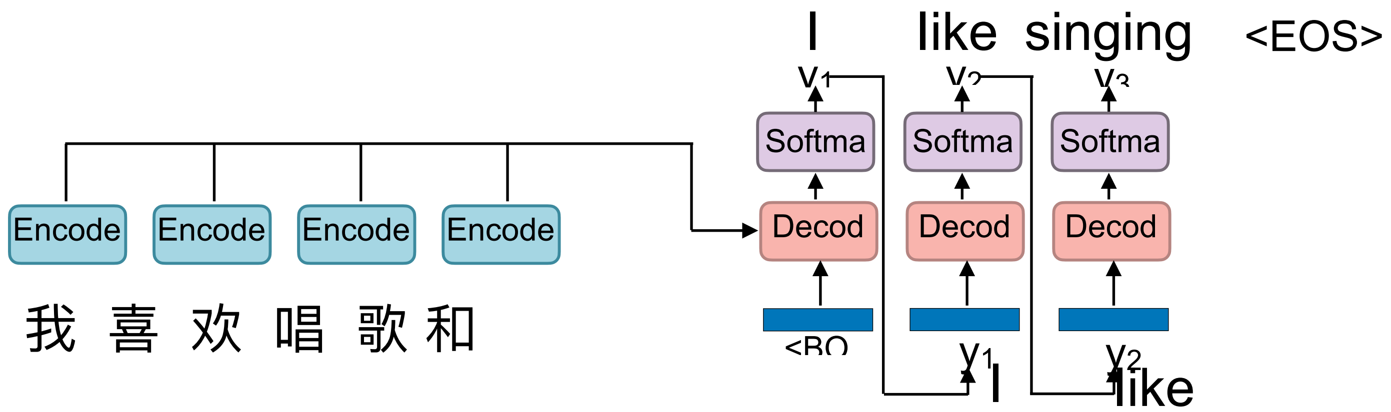
$$p(y_i) \propto \exp(W h_i) \Rightarrow p(y_i) \propto \exp(W h_i + V c_i)$$

Mnih et al. Recurrent Models of Visual Attention. 2014.

Decoding

Autoregressive Generation

greedy decoding: output the token with max next token prob



But, this is not necessary the best

Inference

- Now already trained a model θ
- Decoding/Generation: Given an input sentence x, to generate the target sentence y that maximize the probability $P(y \mid x; \theta)$
- $\underset{y}{\operatorname{argmax}}\, P(y \mid x) = f_\theta(x, y)$
- Two types of error
  - the most probable translation is bad → fix the model
  - search does not find the most probably translation → fix the search
- Most probable translation is not necessary the highest BLEU one!

Decoding

- $\underset{y}{\operatorname{argmax}} P(y \mid x) = f_\theta(x, y)$
- naive solution: exhaustive search
  - too expensive
- Beam search
  - (approximate) dynamic programming

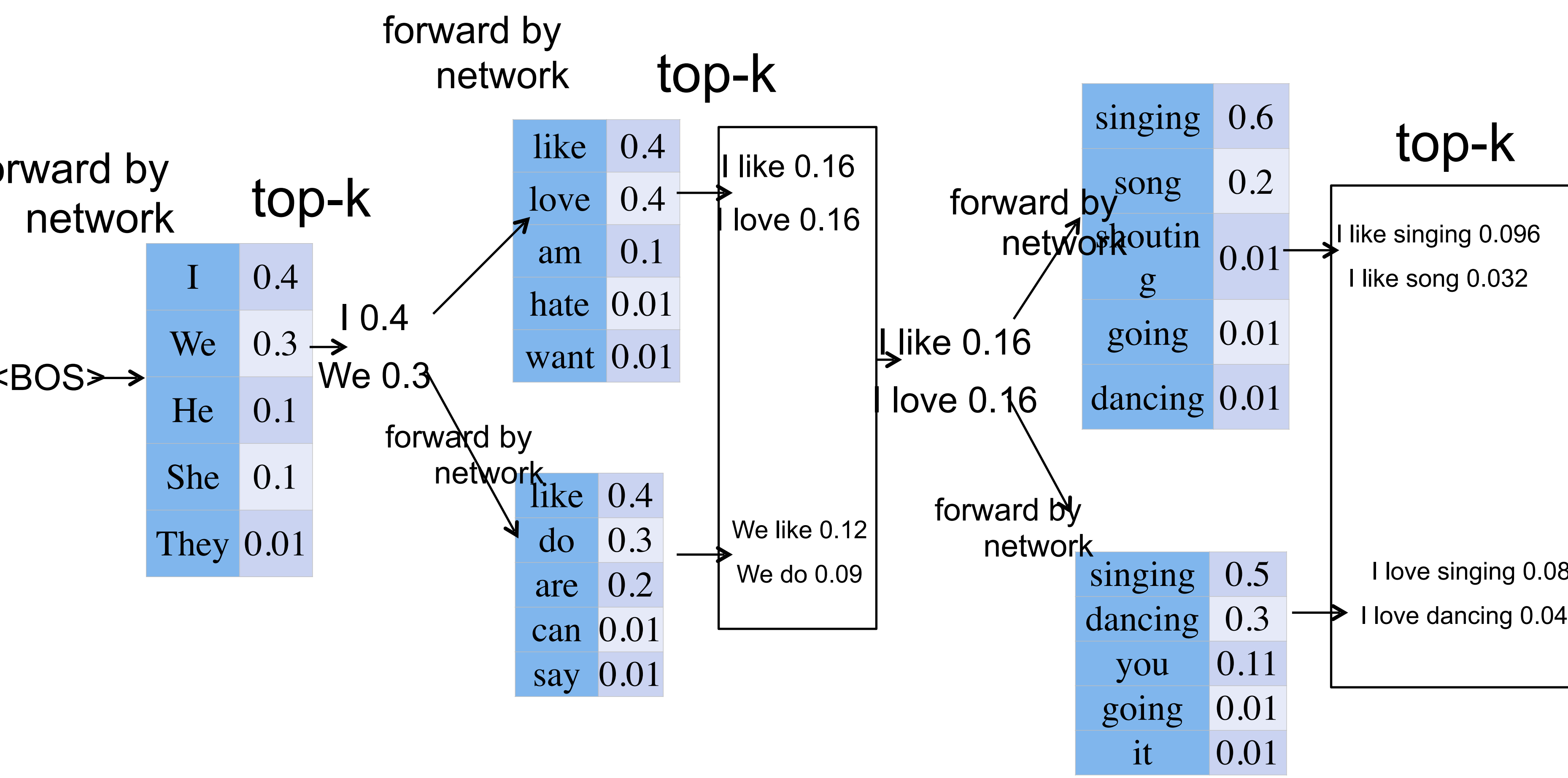
Beam Search

- start with empty S
- at each step, keep k best partial sequences
- expand them with one more forward generation
- collect new partial results and keep top-k

Beam Search (pseudocode)

```
best_scores = []
add {[0], 0.0} to best_scores # 0 is for beginning of sentence token
for i in 1 to max_length:
  new_seqs = PriorityQueue()
  for (candidate, s) in best_scores:
    if candidate[-1] is EOS:
        prob = all -inf
        prob[EOS] = 0
      else:
      prob = using model to take candidate and compute next token
probabilities (logp)
    pick top k scores from prob, and their index
    for each score, index in the top-k of prob:
      new_candidate = candidate.append(index)
      new_score = s + score
      if not new_seqs.full():
```

Beam Search



forward by network
top-k
<BOS>
I 0.4
We 0.3
He 0.1
She 0.1
They 0.01
I 0.4
We 0.3
forward by network
top-k
like 0.4
love 0.4
am 0.1
hate 0.01
want 0.01
forward by network
like 0.4
do 0.3
are 0.2
can 0.01
say 0.01
I like 0.16
I love 0.16
We like 0.12
We do 0.09
I like 0.16
I love 0.16
forward by network
singing 0.6
song 0.2
shouting 0.01
going 0.01
dancing 0.01
forward by network
singing 0.5
dancing 0.3
you 0.11
going 0.01
it 0.01
top-k
I like singing 0.096
I like song 0.032
I love singing 0.08
I love dancing 0.048

Seq2seq for Machine Translation

Many possible translation, which is better?

SpaceX周三晚间进行了一次发射任务，将四名毫无航天经验的业余人士送入太空轨道。

SpaceX launched a mission Wednesday night to put four amateurs with no space experience into orbit.

SpaceX conducted a launch mission on Wednesday night, sending four amateurs with no aerospace experience into space orbit.

SpaceX conducted a launch mission Wednesday night that sent four amateurs with no spaceflight experience into orbit.

SpaceX carried out a launch mission on Wednesday night to put four amateurs without Aerospace experience into orbit.

BLEU

- Measuring the precision of n-grams
  - Precision of n-gram: percentage of tokens in output sentences

$$\_\,p_n = \frac{num.of.correct.token.ngram}{total.output.ngram}$$

- Penalize for brevity
  - if output is too short
  - $bp = min(1, e^{1-r/c})$
- BLEU=$bp \cdot (\prod p_i)^{\frac{1}{4}}$
- Notice BLEU is computed over the whole corpus, not on one sentence

Example

Ref: A SpaceX rocket was launched into a space orbit Wednesday evening.

System A: SpaceX launched a mission Wednesday evening into a space orbit.

System B: A rocket sent SpaceX into orbit Wednesday.

Example

Ref: A SpaceX rocket was launched into a space orbit Wednesday evening.

System A: SpaceX launched a mission Wednesday evening into a space orbit.

	Precision
Unigram	9/11
Bigram	4/10
Trigram	2/9
Four-gram	1/8

$$bp = e^{1-12/11} = 0.91$$

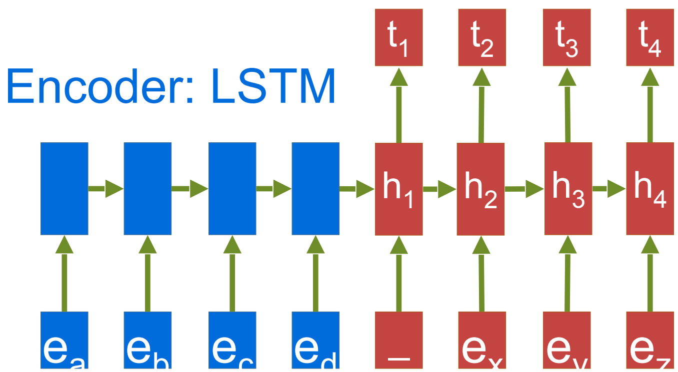
$$BLEU = 0.91 * (9/11 * 4/10 * 2/9 * 1/8)^{1/4}$$

$$= 28.1\%$$

LSTM Seq2Seq for NMT

- Machine translation as directly learning a function mapping from source sequence to target sequence

target:
The weather is nice



Source: 天 气 很 好 Decoder: LSTM

$$P(Y|X) = \prod P(y_t | y_{<t}, x)$$

Training loss: Cross-Entropy

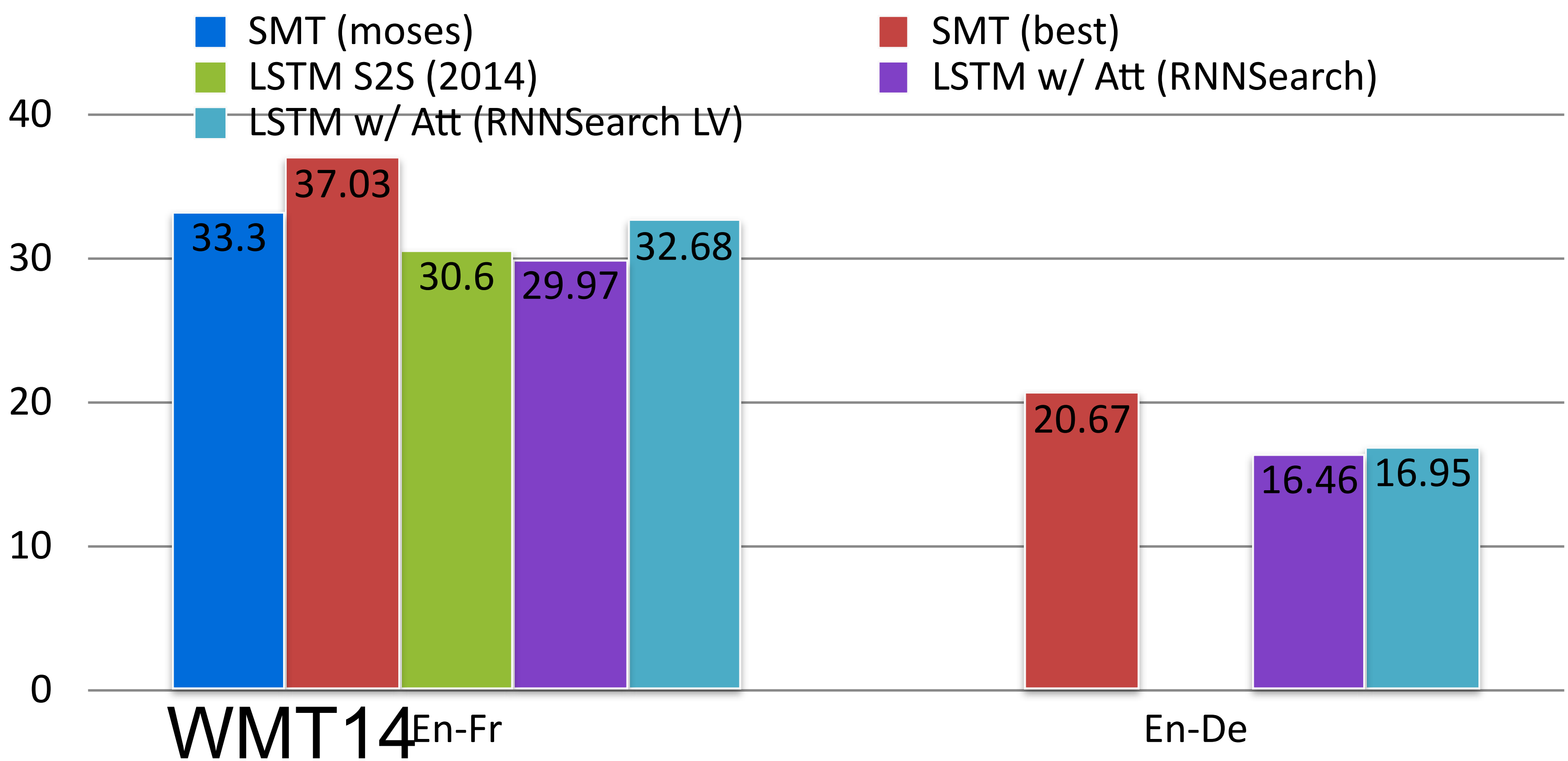
$$l = -\sum_n \sum_t \log f_\theta(x_n, y_{n,1}, \ldots, y_{n,t-1})$$

Teacher-forcing during training.

(pretend to know groundtruth for prefix)

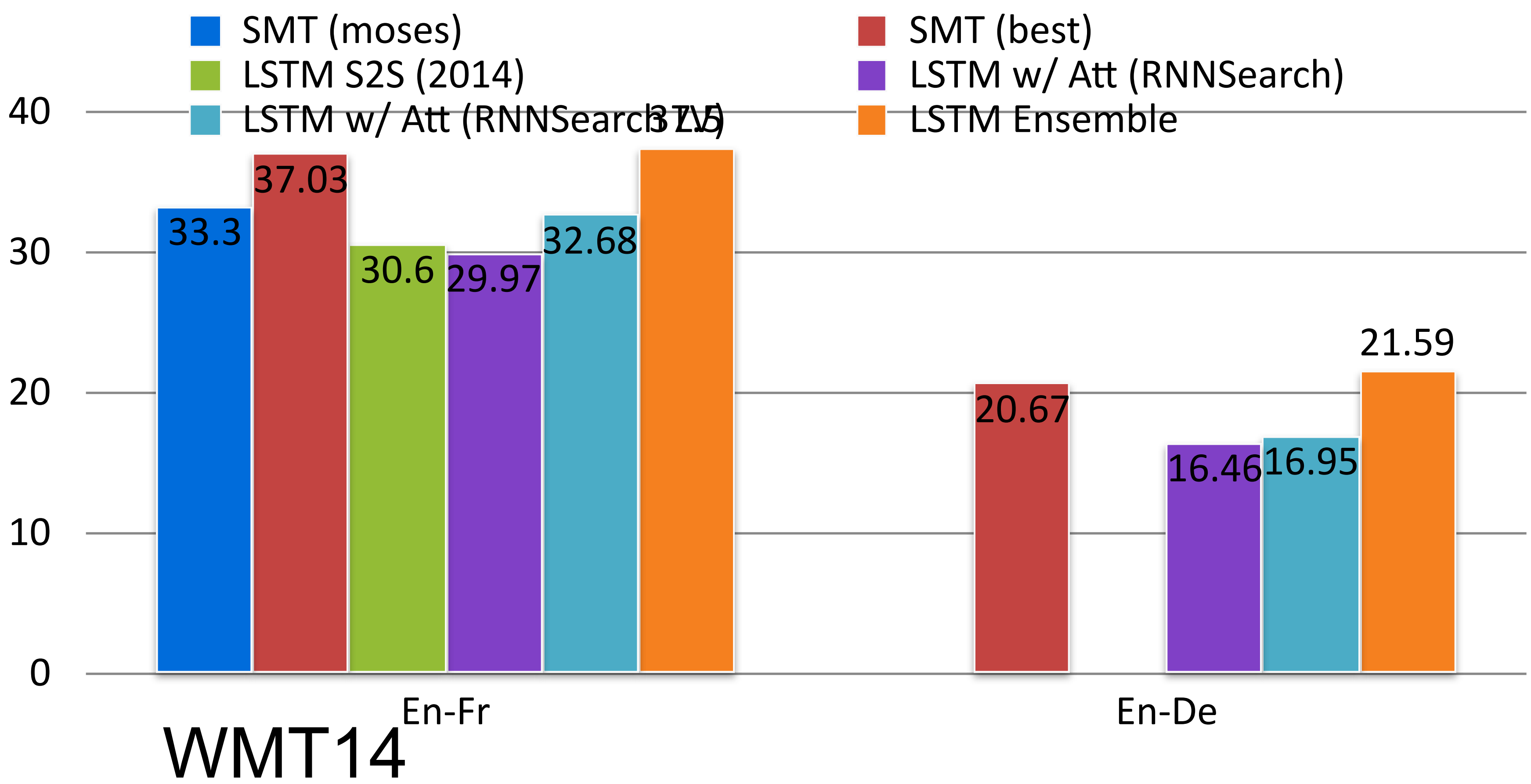
Sutskever et al. Sequence to Sequence Learning with Neural Networks. 2014

LSTM Seq2Seq w/ Attention



Jean et al. On Using Very Large Target Vocabulary for Neural Machine Translation. 2015

Performance with Model Ensemble



Luong et al. Effective Approaches to Attention-based Neural Machine Translation. 2015

Summary

- Recurrent Neural Network
- Long-short term memory
- Gated recurrent units
- Attention between decoder and encoder
- Sequence Labelling with LSTM
- LSTM seq2seq for Machine Translation

Video Cover Selection

- Your manager assigns a task for you: build a system to automatically select the cover photo for a short video on Tiktok
- Please discuss in groups how you plan to build the system

Next up

- Transformer
- What story you'd like to hear about?
  – A robot writer that can write Olympic sport news, or
  – Lessons learned in building real MT product, or
  – an 8-week journey to develop AI component for map product