香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

DDA4220 Deep Learning and Applications

Lecture 7 Natural Language Processing Basic

Ruimao Zhang
zhangruimao@cuhk.edu.cn

School of Data Science
The Chinese University of Hong Kong (Shenzhen)

Slides partially credited to Prof. Diyi Yang and at Stanford

What is Natural Language Processing (NLP)

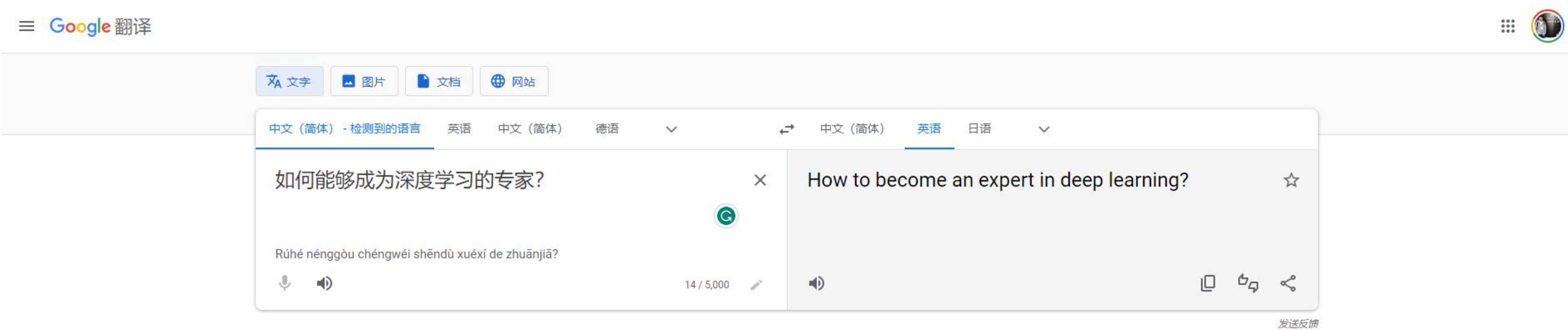
- Natural Language Processing (NLP) is a subfield of artificial intelligence and computational linguistics that focuses on the interaction between computers and human language.
- NLP enables computers to understand, interpret, and generate human language, including speech and text.

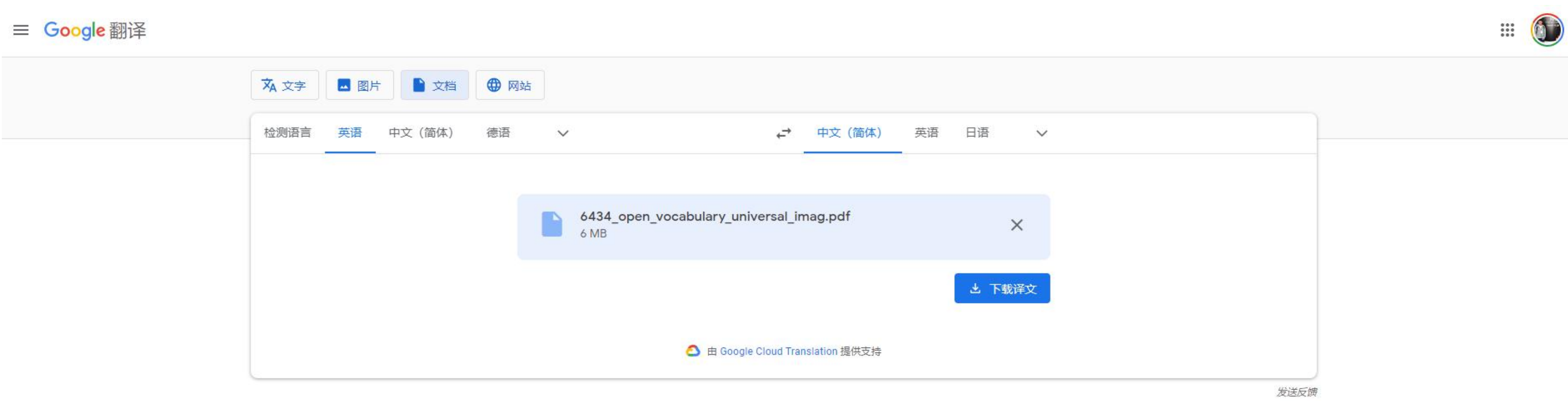
Conversational agents：

- Speech recognition
- Language analysis
- Dialogue processing
- Information retrieval
- Text to speech

NLP in our everyday life

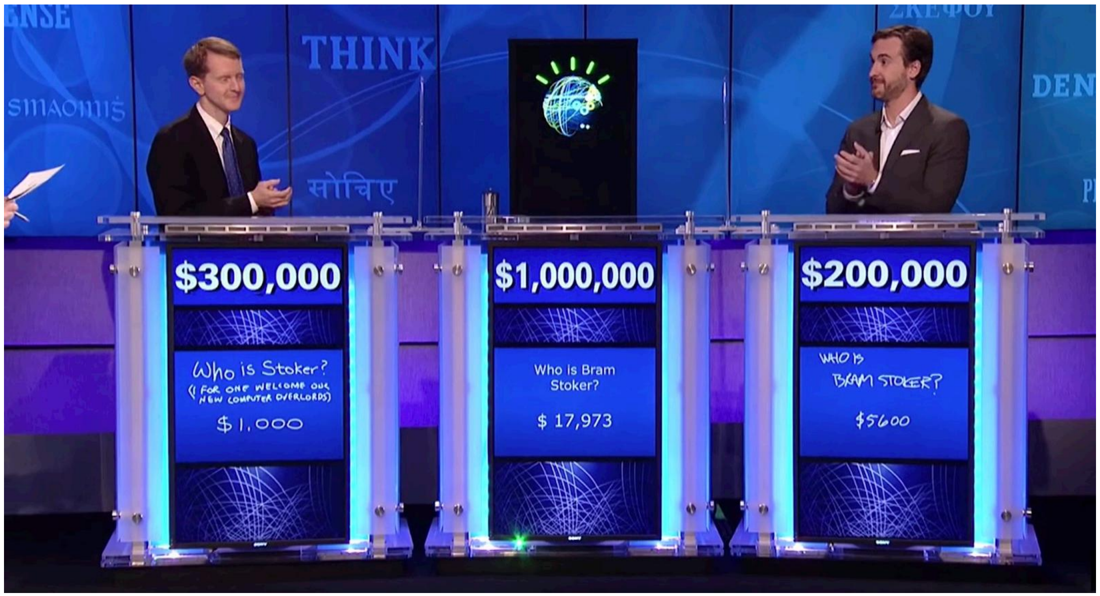
- **Machine Translation**





NLP in our everyday life

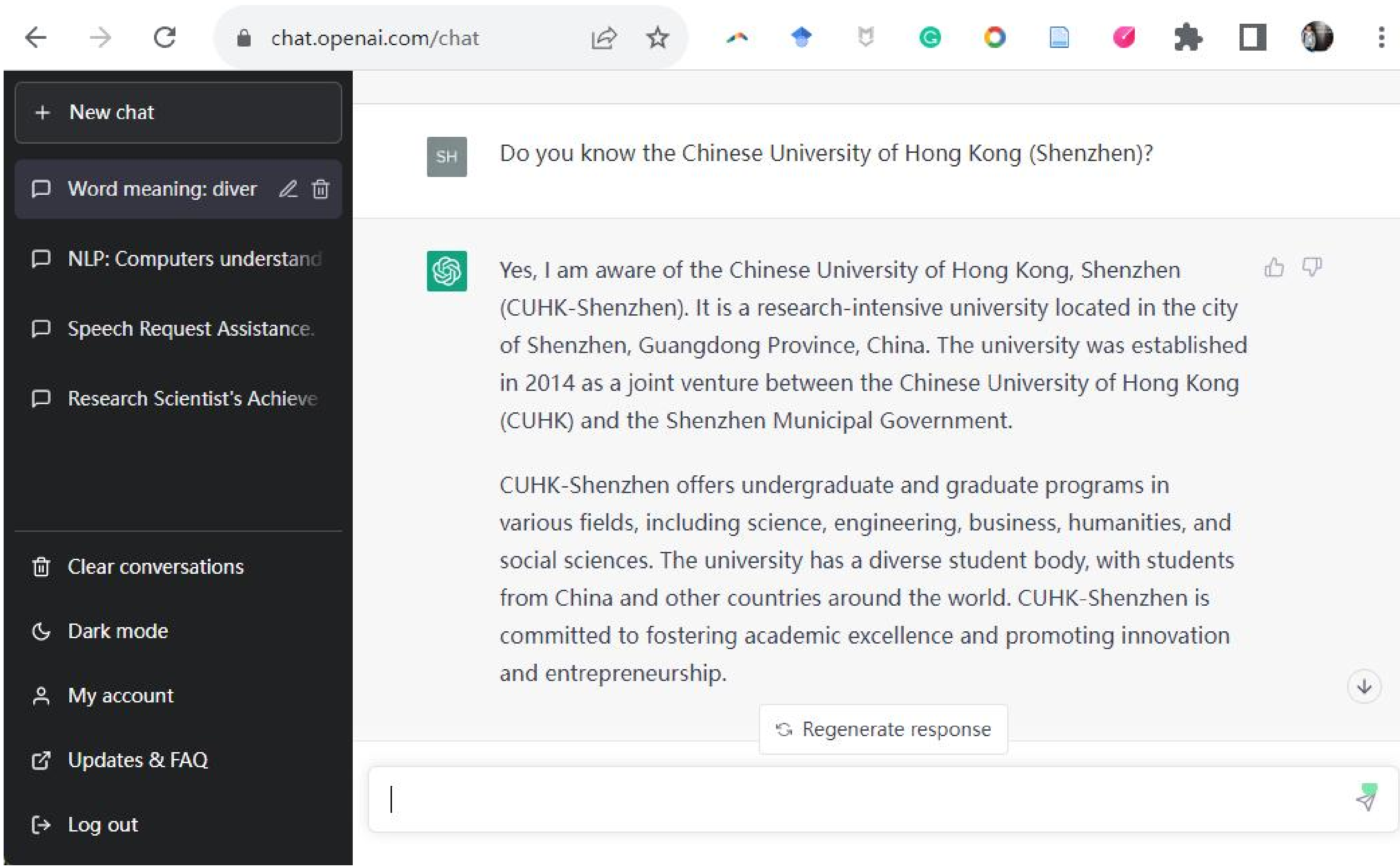
- **Question Answering**



- What does "divergent" mean?
- What year was Abraham Lincoln born?
- How many states were in the United States that year?
- How much Chinese silk was exported to England in the end of the 18th century?
- What do scientists think about the ethics of human cloning?

NLP in our everyday life

- **Dialogue Systems**



Natural Language Processing

Applications

- Text Classification .
- Named Entity Recognition
- Sentiment Analysis
- Machine Translation
- Text Summarization
- Question Answering
- Speech Recognition
- Text Generation
- ....

Core Technologies

- Language modeling
- Part-of-speech tagging
- Syntactic parsing
- Named-entity recognition
- Word sense disambiguation
- Semantic role labeling
- ....

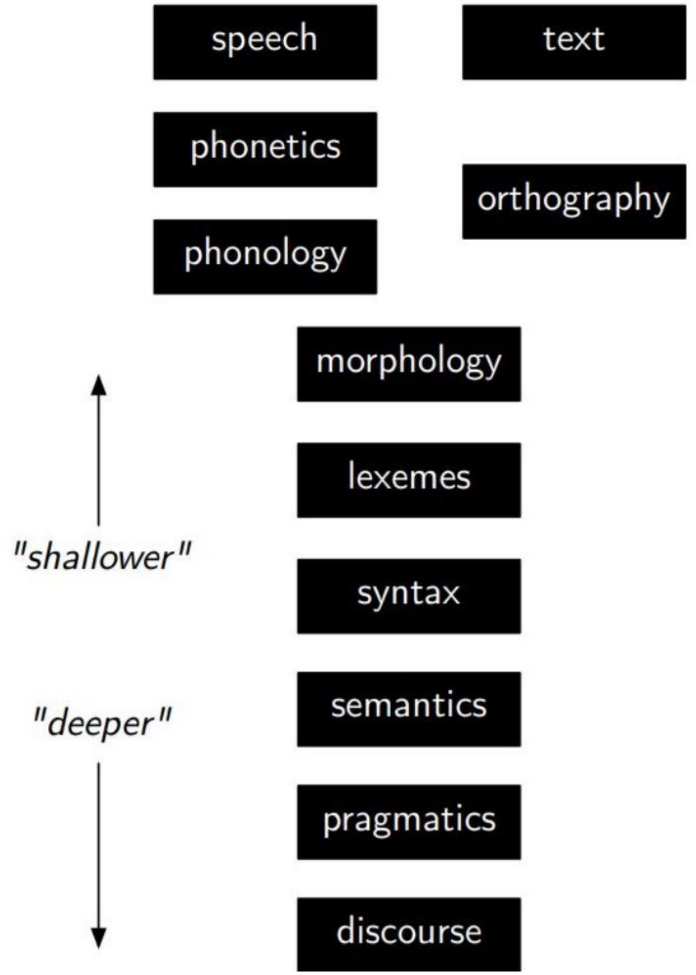
NLP lies at the intersection of computational linguistics and machine learning.

Natural Language Processing

Level Of

Linguistic

Knowledge



Level of Linguistic Knowledge

- Phonetics, Phonology

SOUNDS Th i a si e n

- Words
  - Language Modeling
  - Tokenization
  - Spelling correction

WORDS This is a simple sentence

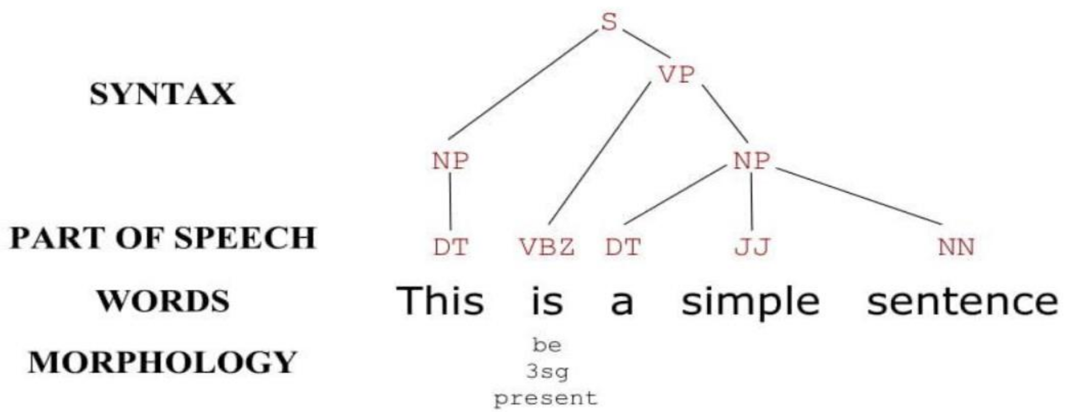
Level of Linguistic Knowledge

- Morphology
  - Morphology analysis
  - Tokenization
  - Lemmatization

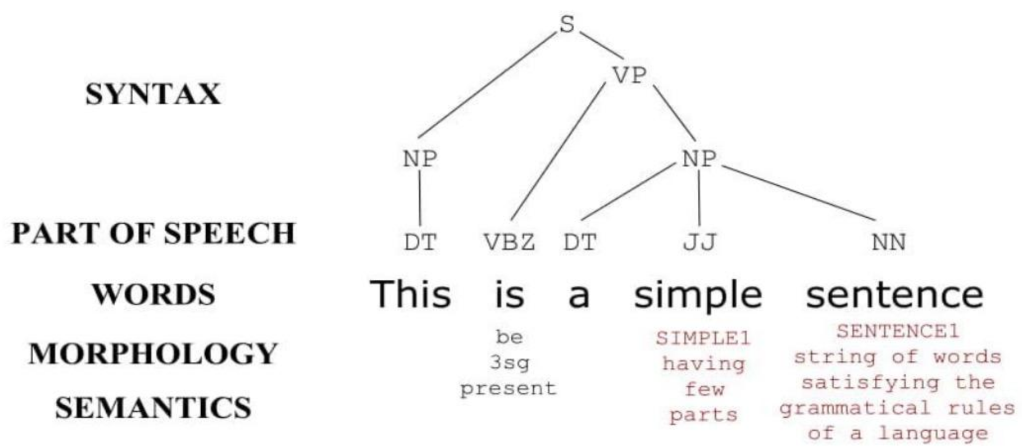
- Part of Speech (POS)
  - POS tagging

Level of Linguistic Knowledge

- Syntax
  - Syntax Parsing

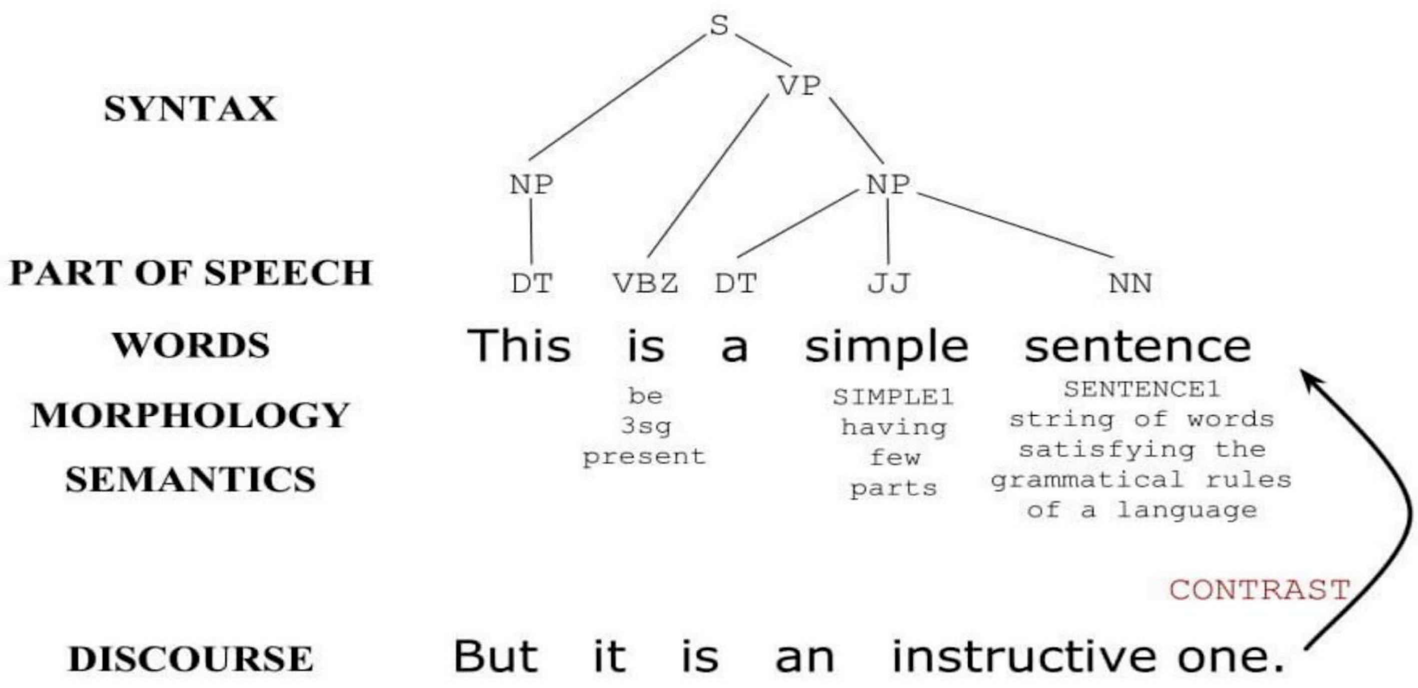


- Semantic
  - Named entity recognition
  - Word sense disambiguation
  - Semantic role labeling



Level of Linguistic Knowledge

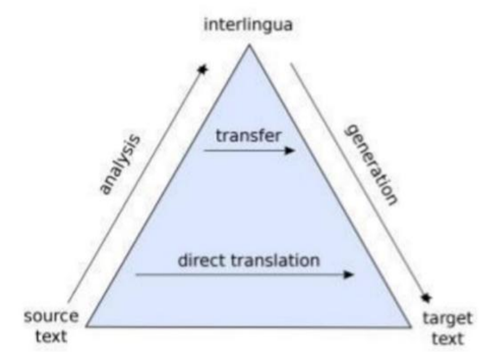
- Discourse



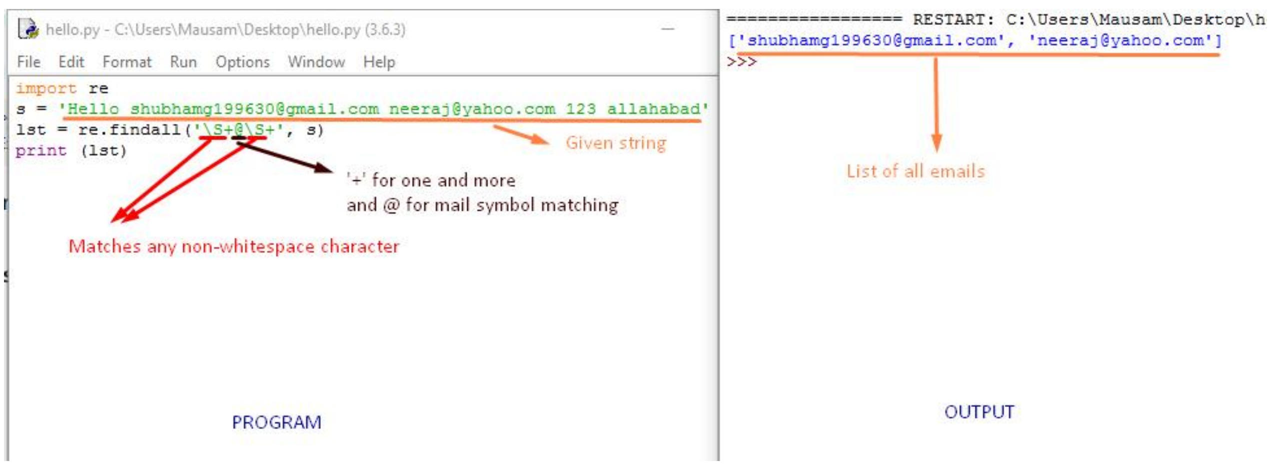
Symbolic NLP

- Symbolic NLP is based on formal rules and logic, where the focus is on the symbolic representation of language.
- Linguistic rules are used to create a formal grammar that can be applied to natural language text.
- Rules are hand-crafted by linguists and experts

Logic-based/Rule-based NLP

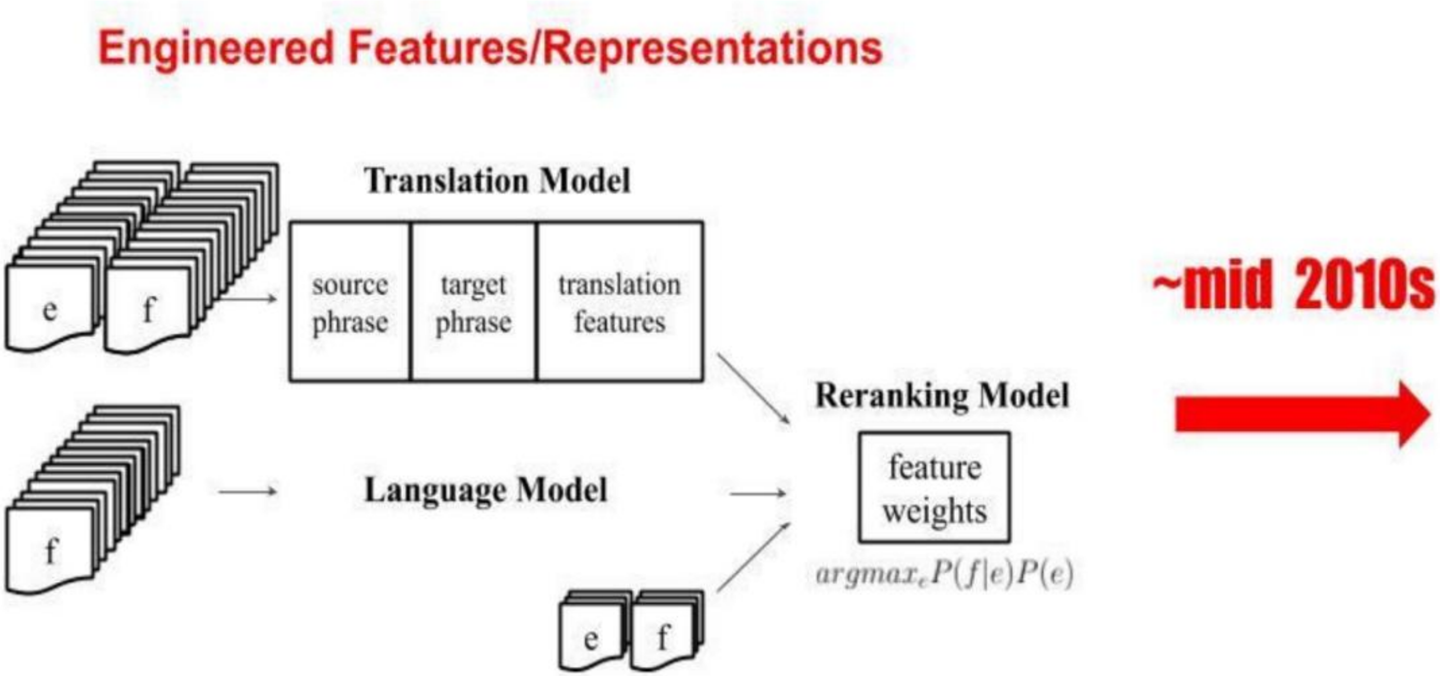


~ 90s



Probabilistic NLP

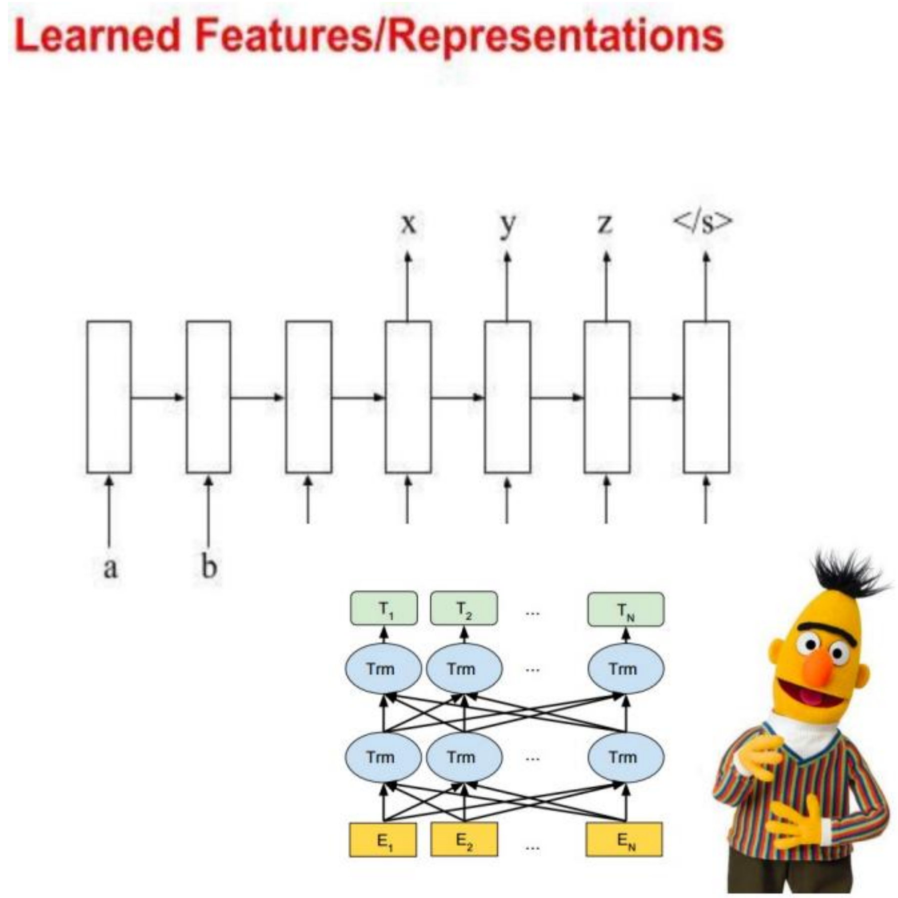
- Probabilistic NLP is based on statistical models and machine learning algorithms
- It learns from data to make predictions about language.
- In probabilistic NLP, the features are automatically learned from the data using statistical methods and machine learning algorithms but not the hand-crafted.



Connectionist NLP

- Connectionist NLP is an approach to natural language processing that uses neural networks to model language and perform various NLP tasks.

- It has the advantage of being able to learn complex features and relationships in data without the need for hand-crafted features or domain-specific knowledge.



Representing words as discrete symbols

- In traditional NLP, we regard words as discrete symbols:

hotel, conference, motel – a localist representation

- Such symbols for words can be represented by **one-hot** vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

- Vector dimension = number of words in vocabulary (e.g., 500,000+)
- These two vectors are **orthogonal**. There is no natural notion of **similarity** for one-hot vectors!

- **Solution:** learn to encode similarity in the vectors themselves

Representing words by their context

- **Distributional semantics:** A word's meaning is given by the words that frequently appear close-by.
  - One of the most successful ideas of modern statistical NLP!
- When a word *w* appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- We use the many contexts of *w* to build up a representation of *w*

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

These **context words** will represent ***banking***

Word vectors

- We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts, measuring similarity as the vector dot (scalar) product

$$\textit{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix} \qquad \textit{monetary} = \begin{pmatrix} 0.413 \\ 0.582 \\ -0.007 \\ 0.247 \\ 0.216 \\ -0.718 \\ 0.147 \\ 0.051 \end{pmatrix}$$

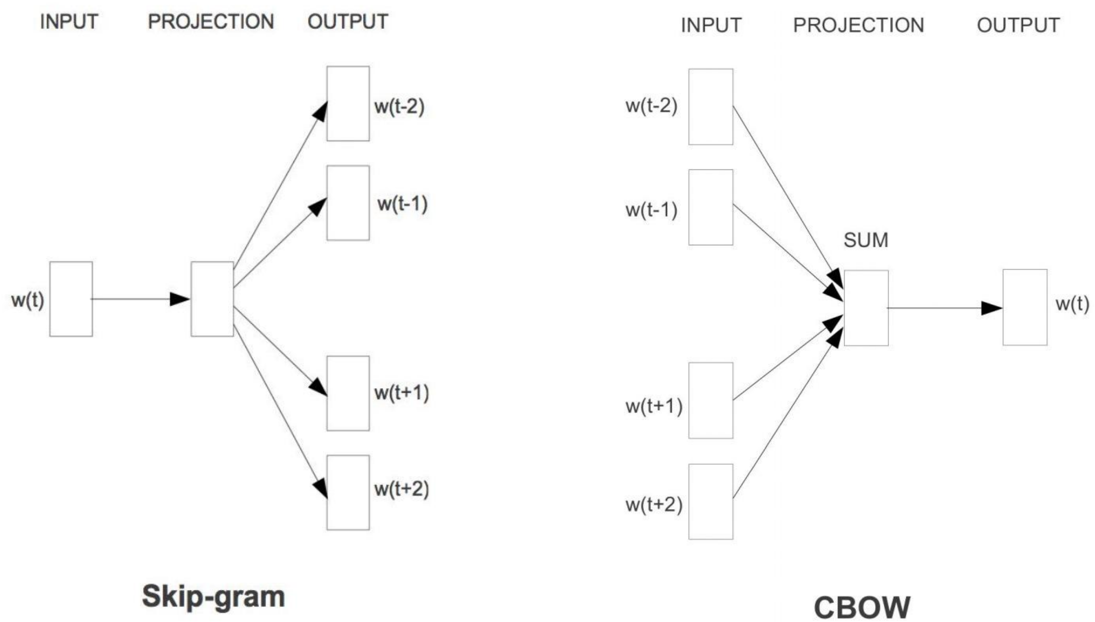
Note: word vectors are also called (word) embeddings or (neural) word representations. They are a distributed representation

Word meaning as a neural word vector – visualization

$$expect = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

Word2vec: Overview

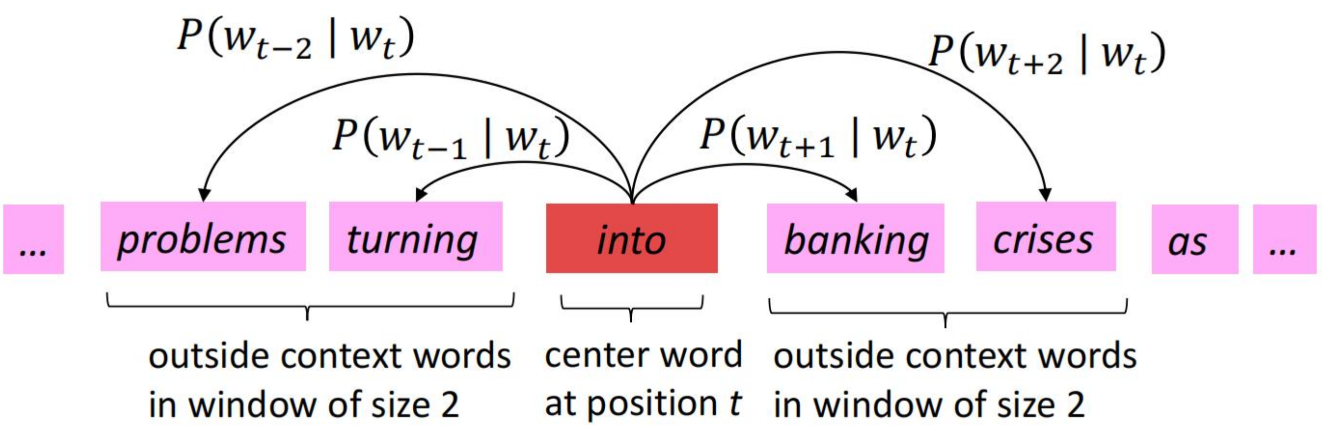
- Similar to language model, but predicting next word is not the goal.
- Idea: **words that are semantically similar often occur in similar context**
- Embeddings that are good at predicting neighboring words are also good at representing similarity



Skip-gram v.s. Continuous bag-of-words

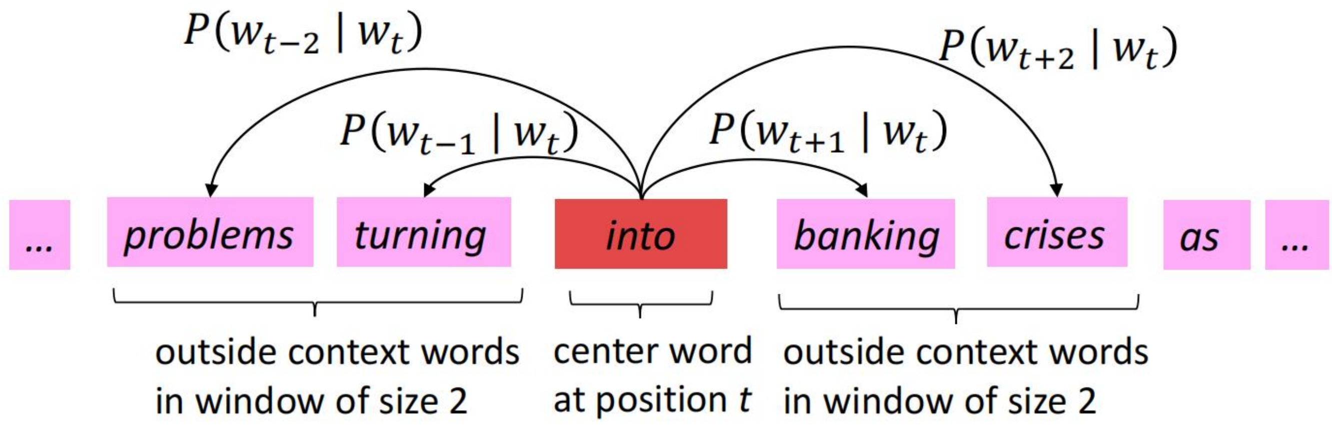
Word2vec: Overview

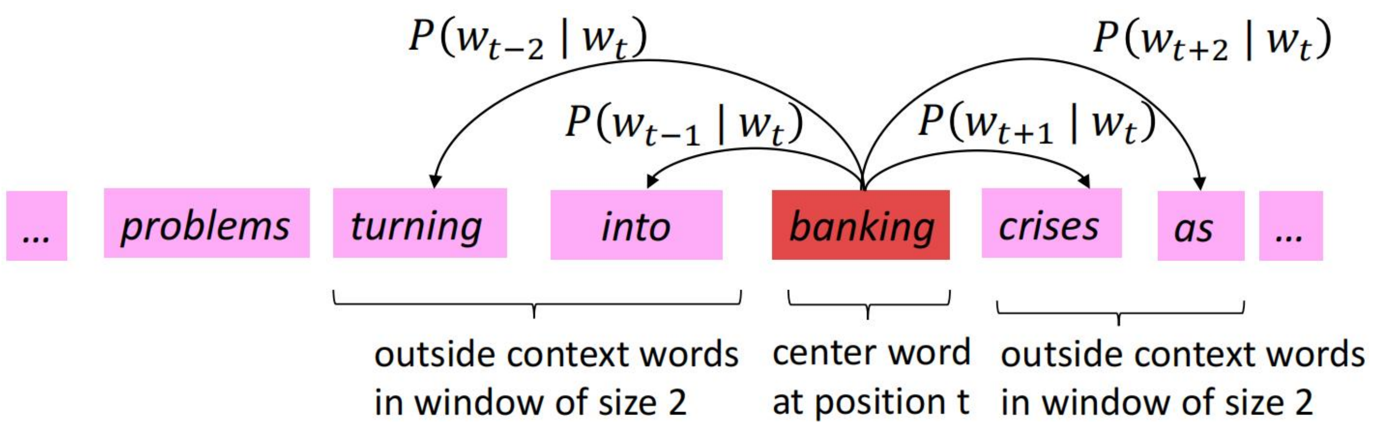
- Basic Solution
  - We have a large corpus (“body”) of text: a long list of words
  - Every word in a fixed vocabulary is represented by a vector
  - Go through each position *t* in the text, which has a center word c and context (“outside”) words *o*
  - Use the similarity of the word vectors for *c* and *o* to calculate the probability of *o* given *c* (or vice versa)
  - Keep adjusting the word vectors to maximize this probability



Word2Vec Overview

- Example windows and process for computing $P(w_{t+j} \mid w_t)$





Skip-gram: objective function

- For each position $t = 1,\ldots,T$, predict context words within a window of fixed size m, given center word w_t. Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

- The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T}\log L(\theta) = -\frac{1}{T}\sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Skip-gram: objective function

- We want to minimize the objective function:

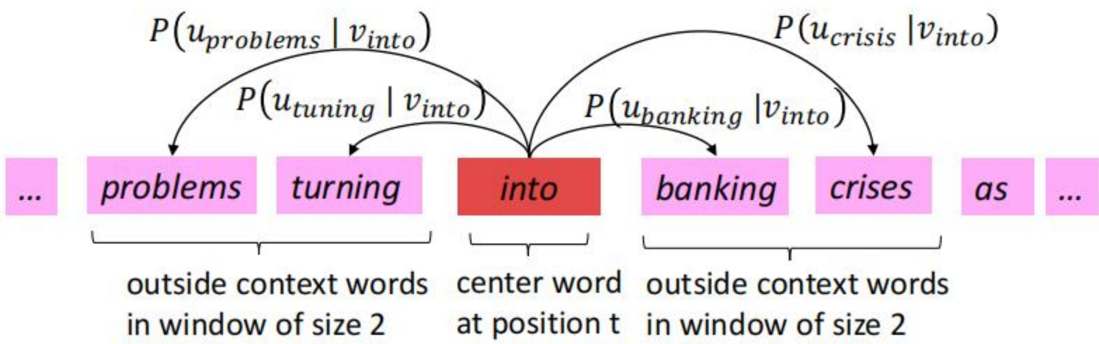
$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m \\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

- We will use two vectors per word *w* to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$
  - v_w when w is a center word
  - u_w when w is a context word
- Then for a center word *c* and a context word *o*:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

Skip-gram with Vectors

- Example windows and process for computing $P(w_{t+j} \mid w_t)$
- $P(u_{problems} \mid v_{into})$ short for $\mathrm{P}(problems \mid into ; u_{problems}, v_{into}, \theta)$



- Treat the target word and a neighboring context word as positive examples
- Randomly sample other words in the lexicon to get negative samples
- Use logistic regression to train a classifier to distinguish those two cases
- Use the weights as the embeddings

Skip-gram with Vectors

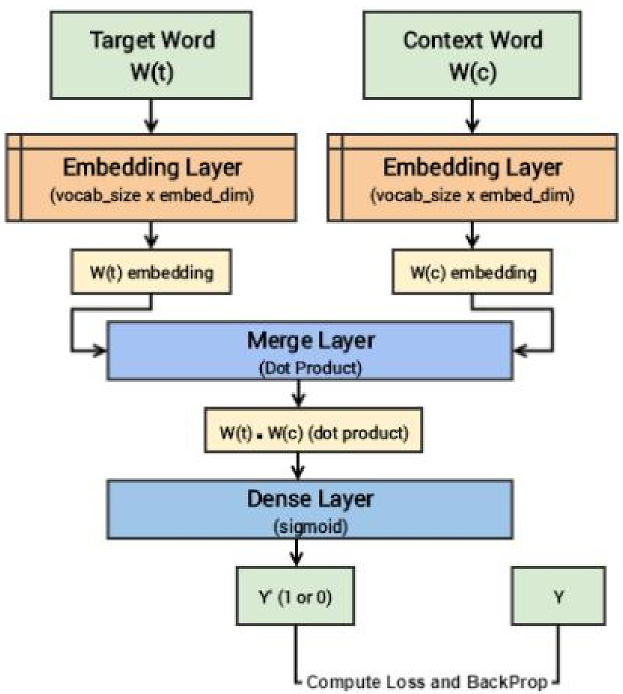
- Prediction Function: the softmax function to map values to a probability distribution

② Exponentiation makes anything positive

$$P(o|c) = \frac{\exp(\boxed{u_o^T v_c})}{\boxed{\sum_{w \in V} \exp(u_w^T v_c)}}$$

① Dot product compares similarity of *o* and *c*.
$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$
Larger dot product = larger probability

③ Normalize over entire vocabulary
to give probability distribution



- Training the Classifier
  - Start with random weights
  - Adjust the word weights to
    - Make the positive pairs more likely
    - Make the negative pairs less likely
  - Over the entire training set

Intrinsic word vector evaluation

- Word Vector Analogies

a:b :: c:?

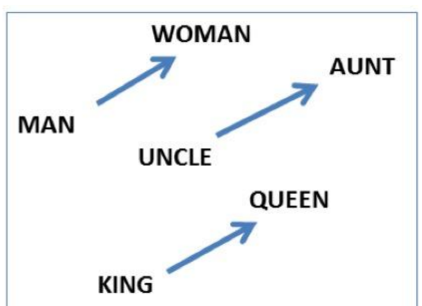
man:woman :: king:?

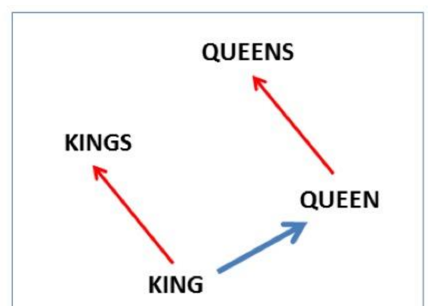
$$\longrightarrow \quad \boxed{d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{||x_b - x_a + x_c||}}$$

- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions
- Discarding the input words from the search (!)

vector('*king*') - vector('*man*') + vector('*woman*') ≈ vector('queen')

vector('*Paris*') - vector('*France*') + vector('*Italy*') ≈ vector('Rome')





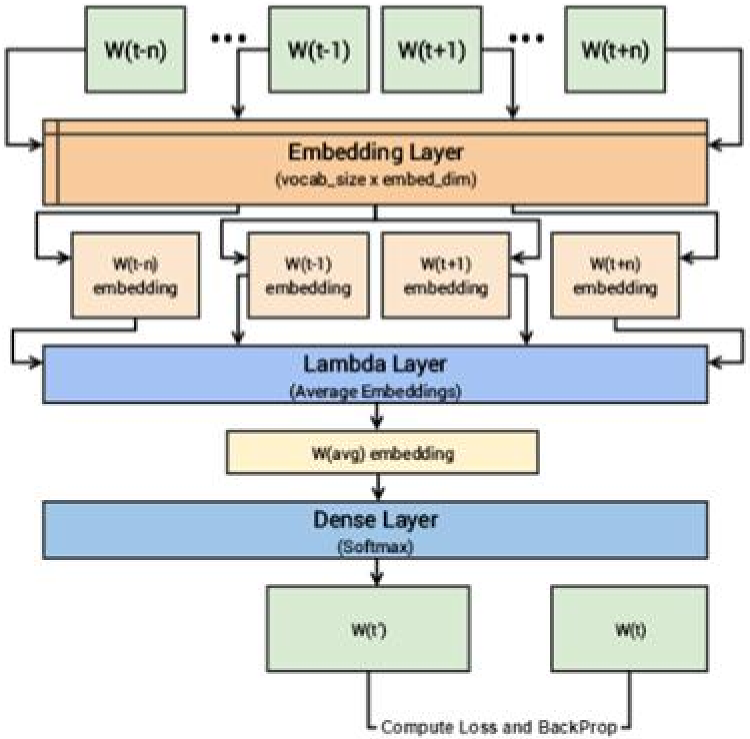
Summary of Skip-gram

- Start with random word vectors as initial embeddings
- Try to predict surrounding words using word vectors: $P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$
  - Take a corpus and take pairs of words that co-occur as positive examples
  - Take pairs of words that don't co-occur as negative examples
  - Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
  - Throw away the classifier code and keep the embeddings

- Doing no more than this, this algorithm learns word vectors that capture well word similarity and meaningful directions in a word space!

Continuous Bag of Words (CBOW)

- The context words are first passed as an input to an embedding layer (initialized with some random weights)
- The word embeddings are then passed to a lambda layer where we average out the word embeddings.
- We then pass these embeddings to a dense SoftMax layer that predicts our target word.
- We match this with our target word and compute the loss and then we perform backpropagation with each epoch to update the embedding layer in the process.

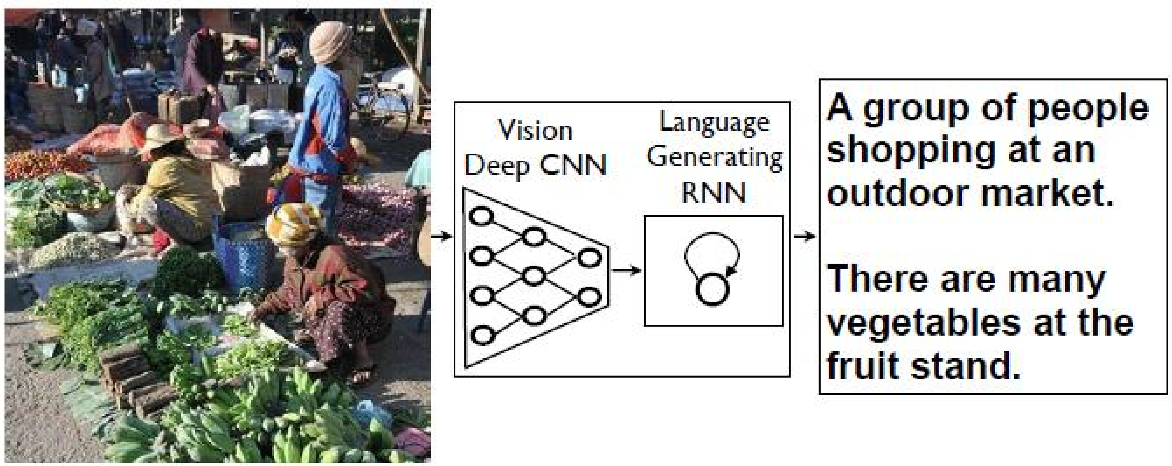


Visual depiction of the CBOW deep learning model

- Extracting out the embeddings of the needed words from our embedding layer, once the training is completed.

Generate image captions (Vinyals et al. arXiv 2014)

- Use a CNN as an image encoder and transform it to a fixed-length vector
- It is used as the initial hidden state of a “decoder” RNN that generates the target sequence



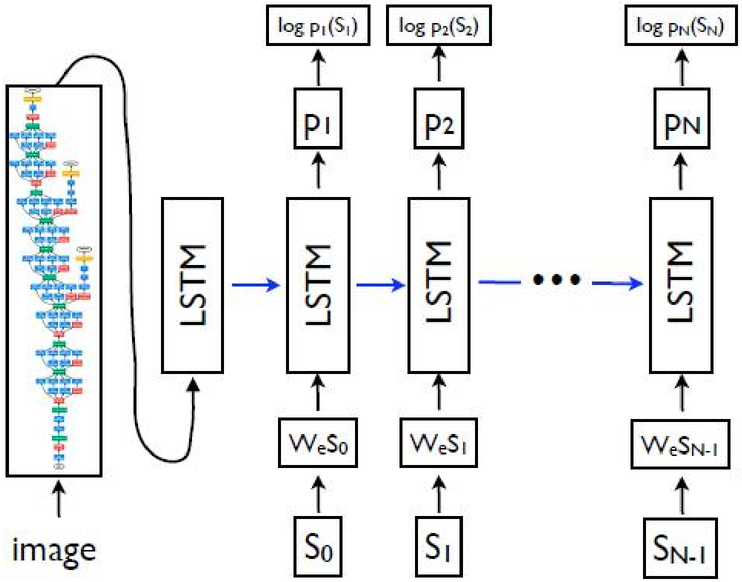
Generate image captions

- The learning process is to maximize the probability of the correct description given the image

$$\theta^* = \arg\max \sum_{(I,S)} \log P(S|I;\theta)$$

$$\log P(S|I) = \sum_{t=0}^{N} \log P(S_t|I, S_0, \ldots, S_{t-1})$$

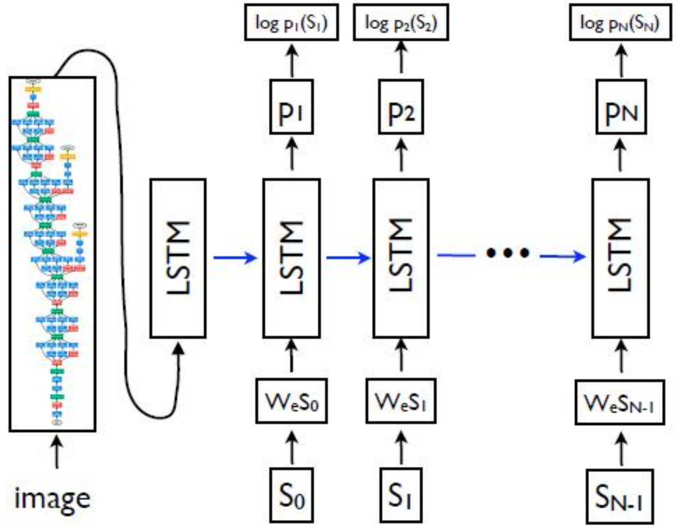
I is an image and **S** is its correct description



Generate image captions

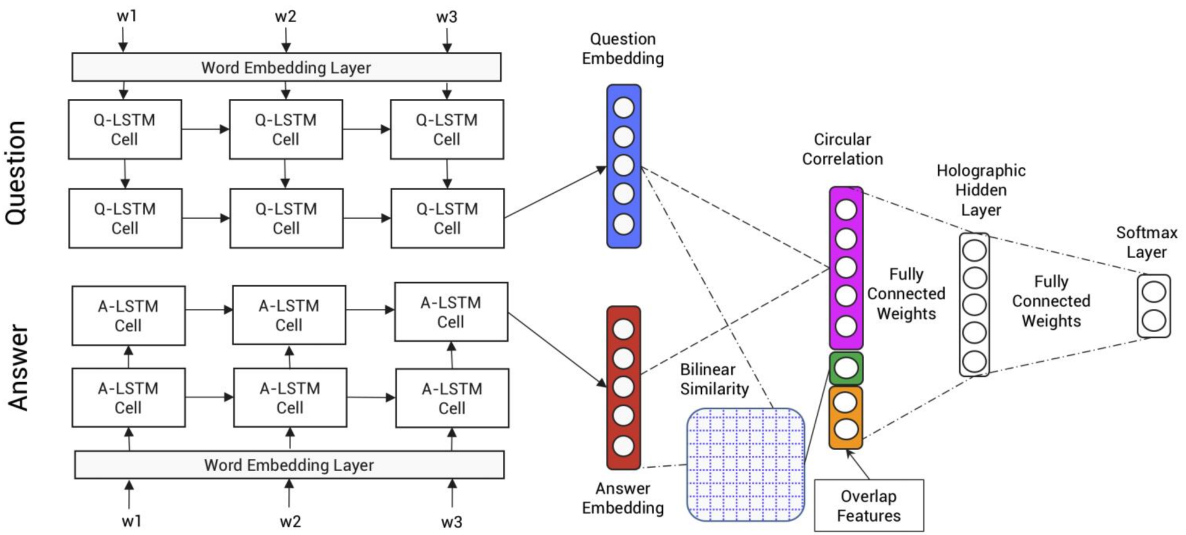
- Denote by S_0 a special start work and by S_N a special stop word
- Both the image and the words are mapped to the same space, the image by using CNN, the words by using word embedding W_e
- The image ***I*** is only input once to inform the LSTM about the image contents
- Sampling: sample the first word according to P_1, then provide the corresponding embedding as input and sample P_2, continuing like this until it samples the special end-of-sentence token

$$x_{-1} = \mathrm{CNN}(\mathbf{I})$$
$$x_t = W_e S_t, t \in \{0, \ldots, N-1\}$$
$$P_{t+1} = \mathrm{LSTM}(x_t), t \in \{0, \ldots, N-1\}$$
$$L(I, S) = -\sum_{t=1}^{N} \log P_t(S_t)$$

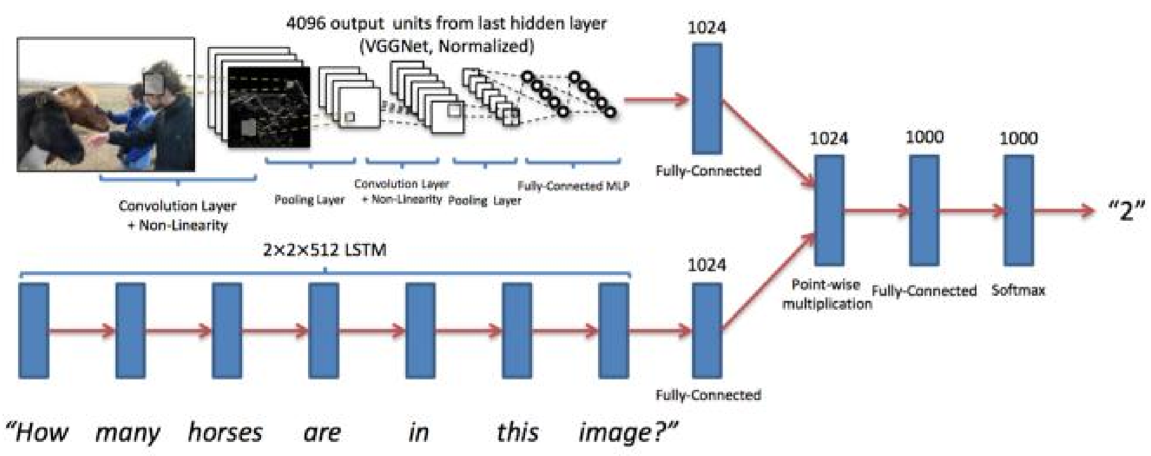


Question Answering and Visual Question Answering

- Question Answering

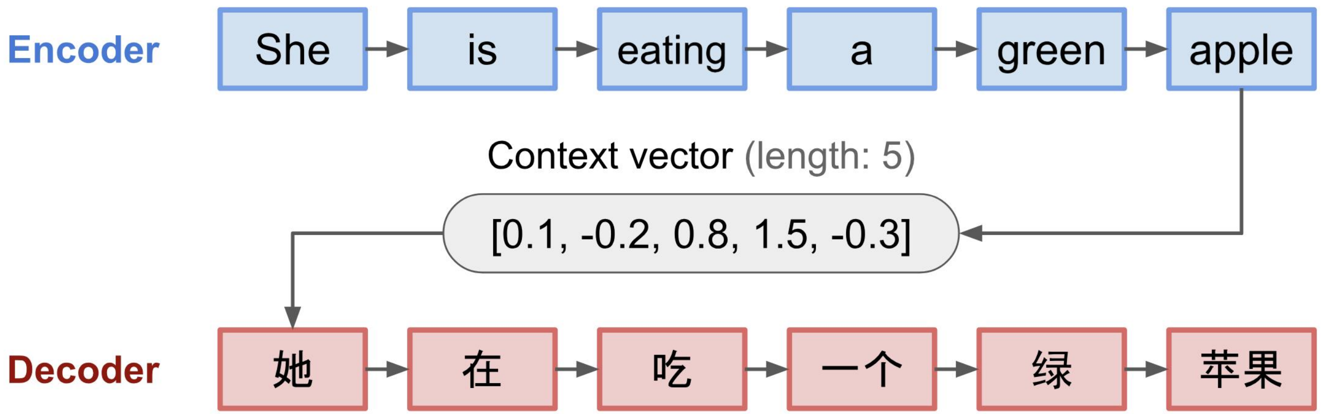


- Visual Question Answering



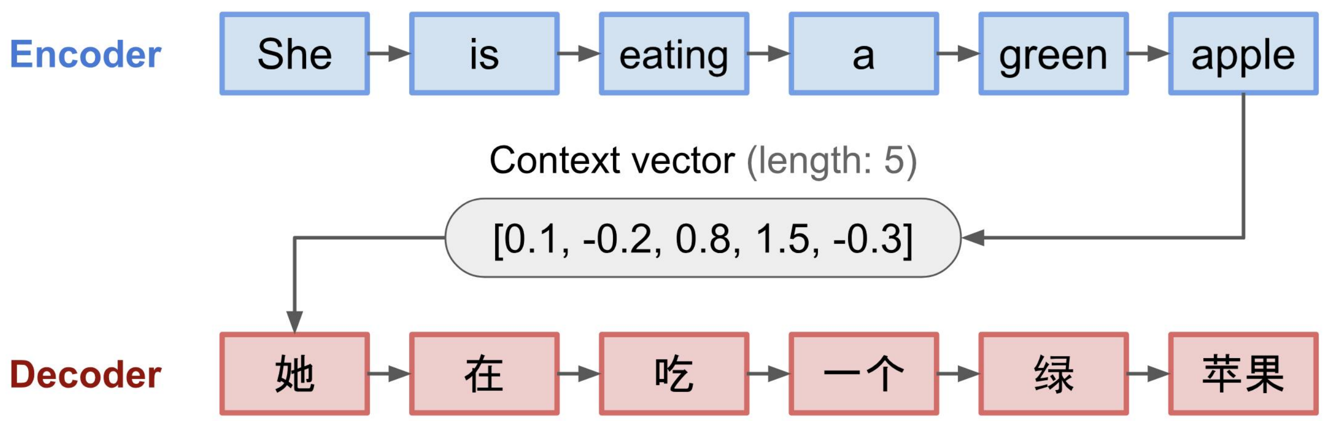
Revisit of seq2seq model

- The conventional seq2seq model is proposed for neural machine translation and generally follows an encoder-decoder architecture
- The encoder converts the input sequence into a sentence embedding (or context vector, or "thought" vector) of a fixed length (dimension)
- The embedding vector is expected to be a comprehensive summary of the whole input sequence
- The decoder takes only the sentence embedding from the encoder as input to emit the output sequence



Attention for Neural Machine Translation (NMT)

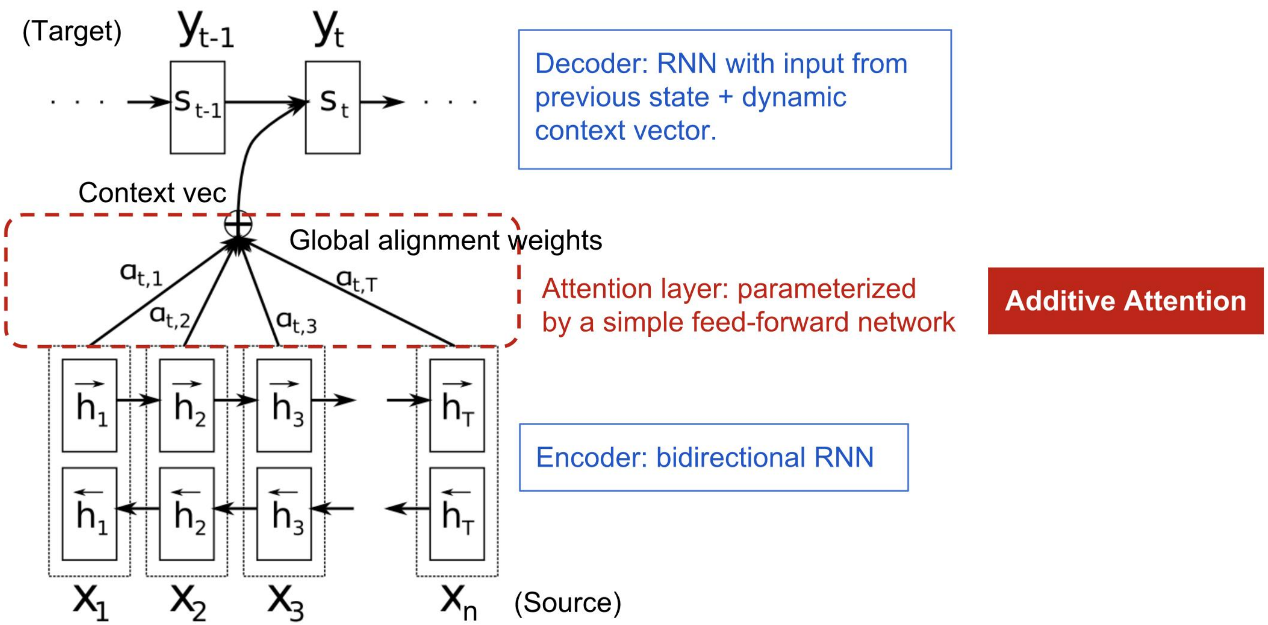
- **The major drawback of the seq2seq model:** this fixed-length embedding vector is incapable of remembering the whole long sentences. Often it is more likely to forget the early parts of the input sequence.
- As the context vector might not be able to capture the information of the whole sentence, the attention mechanism [Bahdanau et al., 2015] explicitly builds word-level alignment between the input and output sequences



Attention for Neural Machine Translation (NMT)

- $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ – input (source) sequence of length n
- $\mathbf{y} = [y_1, y_2, \ldots, y_m]$ – output (target) sequence of length m
- The encoder is a **bidirectional RNN** having forward hidden states $\overrightarrow{h}_i$ and backward hidden states $\overleftarrow{h}_i$, and generating the hidden state at position i

$$h_i = [\overrightarrow{h}_i^T; \overleftarrow{h}_i^T]^T$$



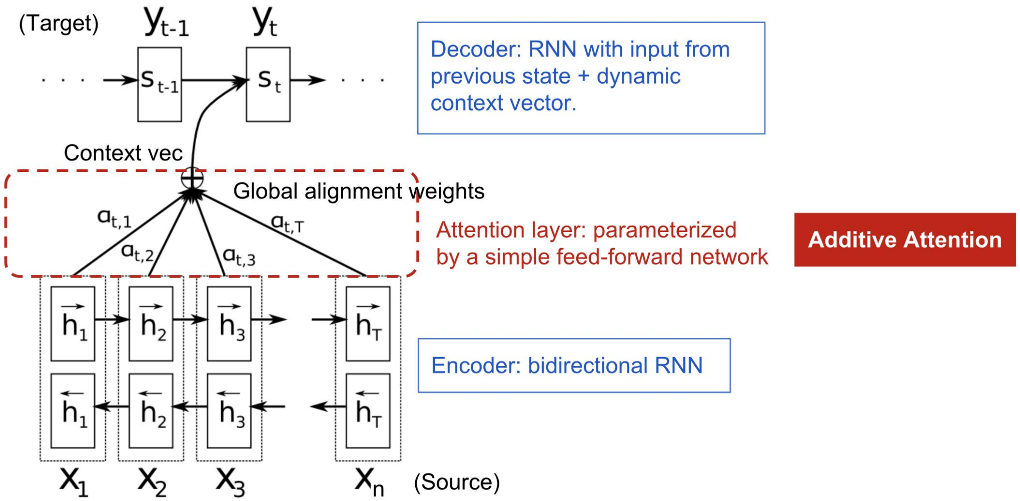
Attention for Neural Machine Translation (NMT)

- The decoder has hidden state $s_t = f(s_{t-1}, y_{t-1}, c_t)$ for the output word at position *t* for *t = 1, . . . , m*

$$c_t = \sum_{i=1}^{n} \alpha_{t,i} h_i$$ Context vector for output y_t

$$\alpha_{t,i} = \text{align}(y_t, x_i)$$ How well two words y_t and x_i are aligned

$$= \frac{\exp(\text{score}(s_{t-1}, h_i))}{\sum_{i'=1}^{n} \exp(\text{score}(s_{t-1}, h_{i'}))}$$ Softmax of some predefined alignment score



Attention for Neural Machine Translation (NMT)

- c_t – the sum of hidden states of the input sequence weighted by the alignment scores, based on which, the class or regression prediction of each output position can be made
- The alignment model assigns a score $\alpha_{t,i}$ to the pair (y_t, x_i) at input position *t* and output position *i*
- score – a 2-layer feed-forward network (or MLP) estimating the affinity between s_{t-1} (just before emitting y_t) and h_i

$$\text{score}(s_t, h_i) = W_1(\tanh(W_2[s_t; h_i] + b_2) + b_1)$$

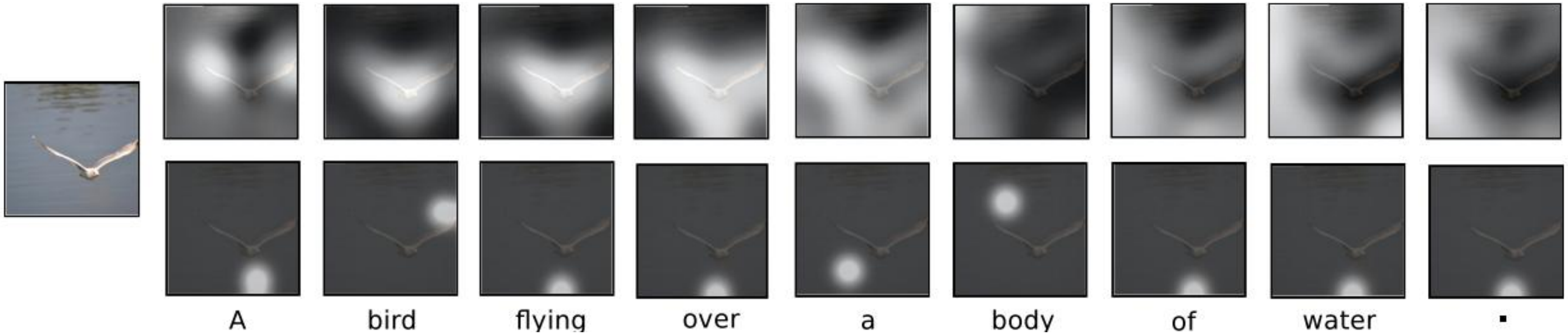
W_1, W_2, b_1, b_2 are weight matrices and biases to be learned

Image Captioning

- By changing the encoder to a CNN model, we can output an image caption according to the contents of an input image
- The above mentioned attention mechanism can also be used to align different image regions with the output words as in the **Show, attend and tell** paper
- The alignment weights $\alpha_{t,i}$ for each output position *t* are normalized across the whole 2D spatial image plane. Each input index *i* indices a pixel (*x, y*) in the 2D feature maps from the visual CNN

Soft and Hard Attention

- The **Show, attend and tell** paper also discusses "soft" vs. "hard" attention, based on whether the attention weights are discrete or continuous over the image regions
- **Soft attention**: the alignment weights are learned and placed "softly" over all patches in the source image, the same as the NMT alignment paper
  - The model is smooth and differentiable
  - Expensive when the source input is large
- **Hard attention:** only selects one part of fixed scale of the image to attend to at a time
  - Less calculation at the inference time
  - The model is non-differentiable and requires more complicated techniques such as variance reduction or reinforcement learning to train



(Top) Soft attention.

(Bottom) Hard attention.

Different Alignment Score Functions

- Different alignment (or similarity measurement) functions

Type	Alignment Score Function
Cosine similarity	$\mathrm{score}(s_t, h_i) = \cos(s_t, h_i)$
Concatenation-based*	$\mathrm{score}(s_t, h_i) = W_1 \tanh(W_2 [s_t; h_i])$
General	$\mathrm{score}(s_t, h_i) = s_t^T W h_i$
Dot-product	$\mathrm{score}(s_t, h_i) = s_t^T h_i$
Scaled dot-product†	$\mathrm{score}(s_t, h_i) = \frac{s_t^T h_i}{\sqrt{\mathrm{dim}}}$

*Bias vectors are not shown. †dim is the dimension of the vectors s_t and h_i.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Thanks for Listening !

Ruimao Zhang

Room 517, Daoyuan Building, The Chinese Univeristy of Hong Kong, Shenzhen
zhangruimao@cuhk.edu.cn
ruimao.zhang@ieee.org