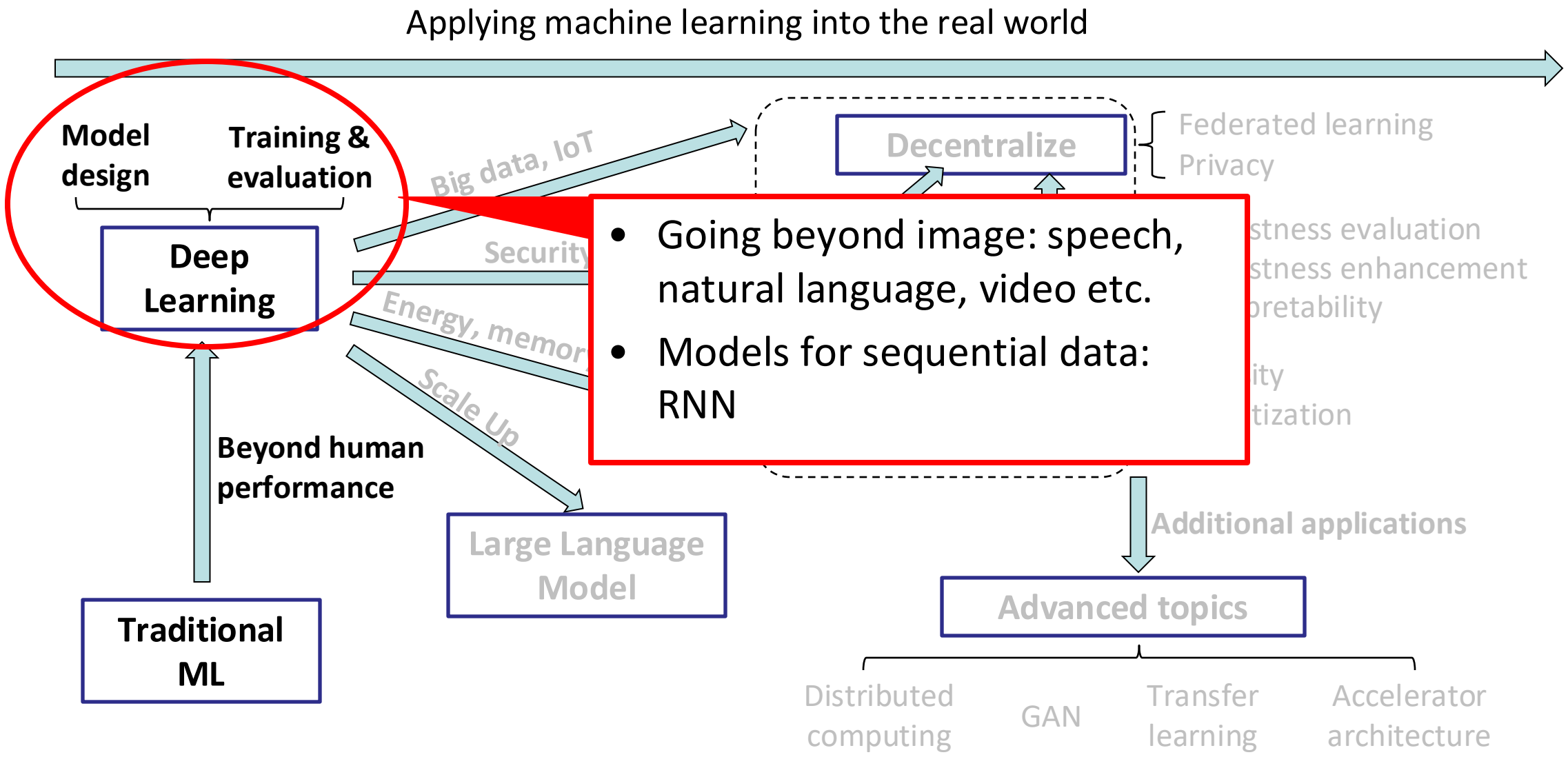
ECE 661 COMP ENG ML & DEEP NEURAL NETS

9. NATURAL LANGUAGE PROCESSING & RECURRENT NEURAL NETWORKS

HAI "HELEN" LI, SPRING 2025

This lecture



Outline

Lecture 9: Natural language processing & Recurrent neural networks

- Natural language processing (NLP)
  - NLP Applications
  - Introduction and Text Normalization
  - Language Models
  - Word Vectors

- Recurrent neural network (RNN)
  - Vanilla RNN
  - LSTM
  - Bidirectional RNN

- RNN language models
  - Encoder-decoder (Seq2seq) model

Outline

Lecture 9: NLP & recurrent neural networks

- Natural language processing (NLP)
  - NLP Applications
  - Introduction and Text Normalization
  - Language Models
  - Word Vectors
- Recurrent neural network (RNN)
  - Vanilla RNN
  - LSTM
  - Bidirectional RNN
- RNN language models
  - Encoder-decoder (Seq2seq) model

NLP Application : Sentiment Classification

HP Officejet 6500A Plus e-All-in-One Color Ink-jet - Fax / copier / printer / scanner

$89 online, **$100** nearby ★★★★☆ 377 reviews

September 2010 - Printer - HP - Inkjet - Office - Copier - Color - Scanner - Fax - 250 sh

Reviews

Summary - Based on 377 reviews

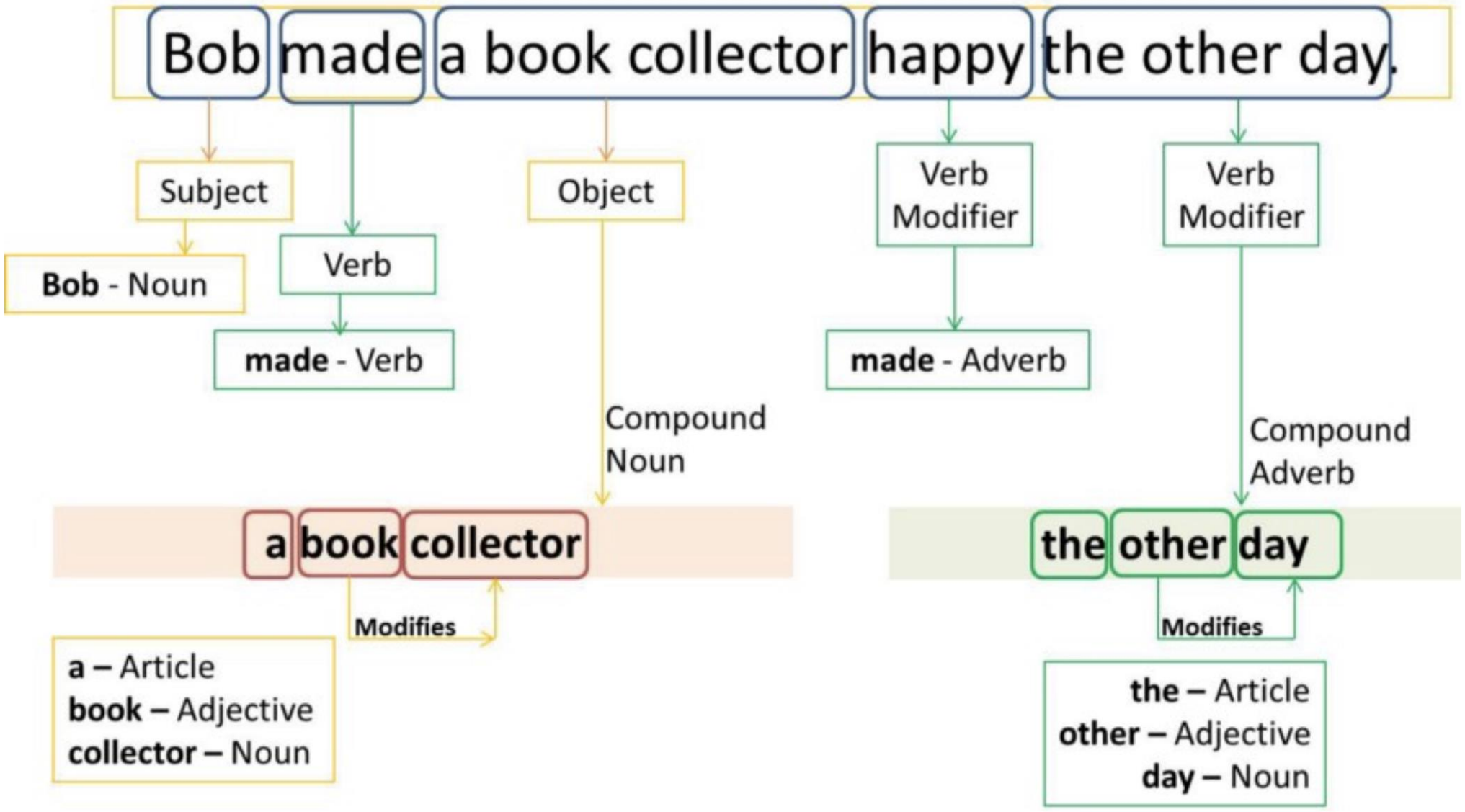
1 star	2	3	4 stars	5 stars

What people are saying

ease of use	"This was very easy to setup to four computers."
value	"Appreciate good quality at a fair price."
setup	"Overall pretty easy setup."
customer service	"I DO like honest tech support people."
size	"Pretty Paper weight."
mode	"Photos were fair on the high quality mode."
colors	"Full color prints came out with great quality."

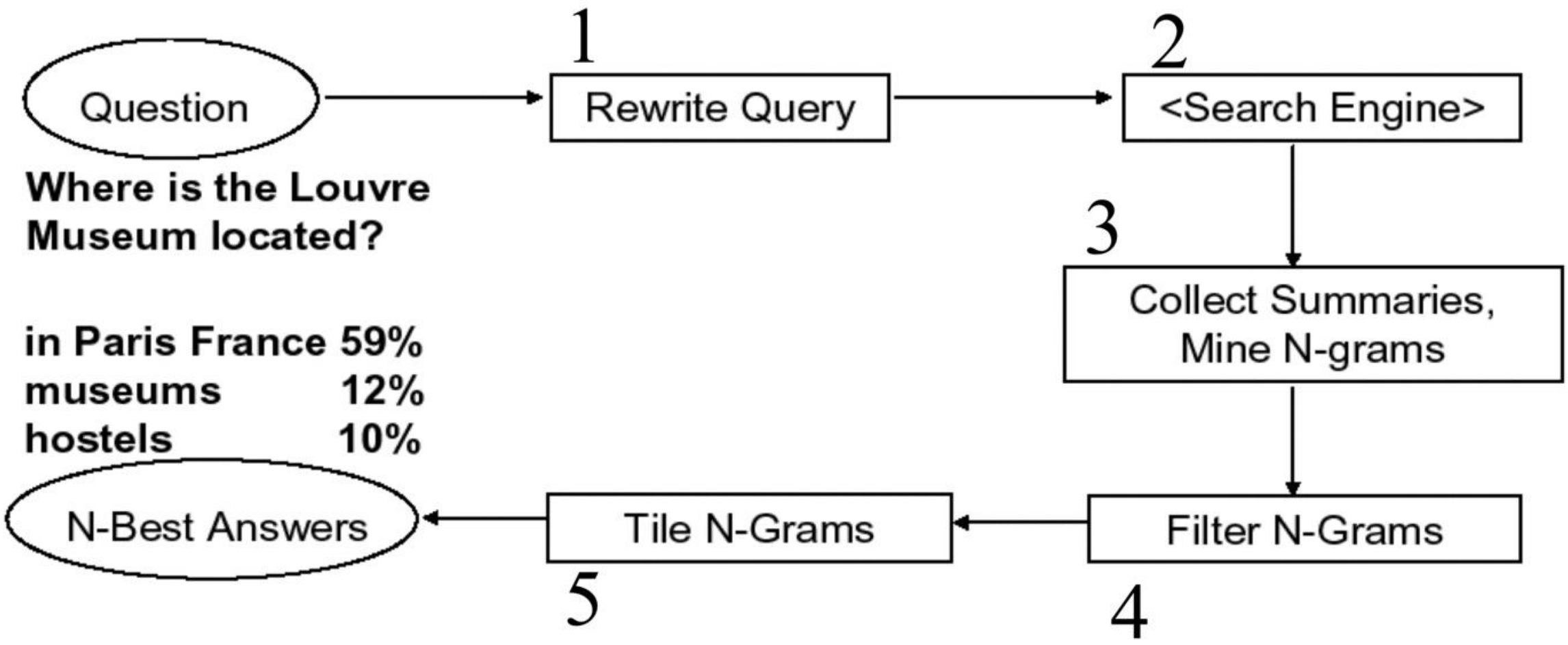
Source: https://web.stanford.edu/~jurafsky/slp3/slides/7_Sent.pdf

NLP Application : Part of Speech Tagging



Source: https://www.freecodecamp.org/news/an-introduction-to-part-of-speech-tagging-and-the-hidden-markov-model-953d45338f24/

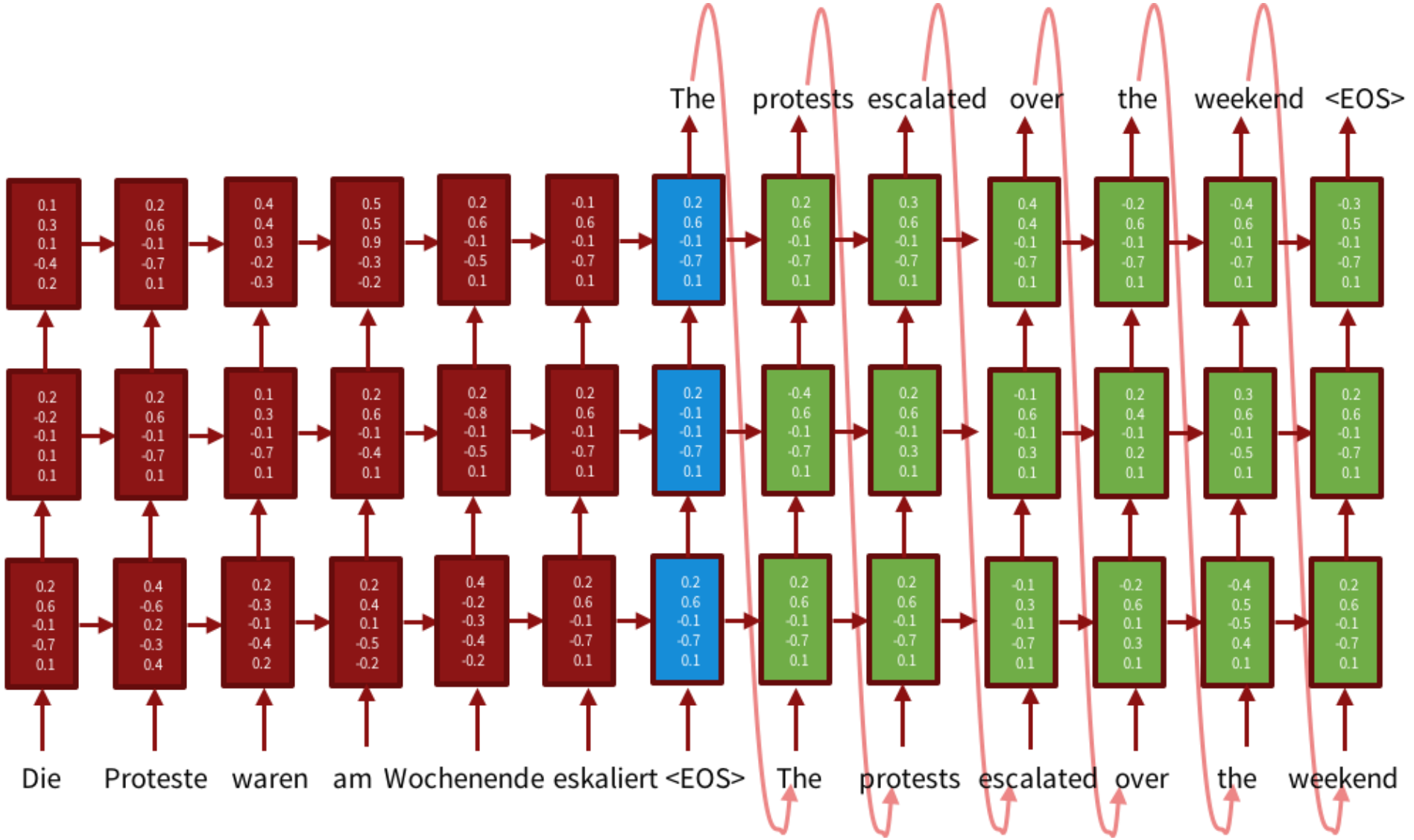
NLP Application : Question Answering



N-grams: naive approach aiming at quantifying the probability that an expression appears in a corpus by counting its number of appearance in the training data.

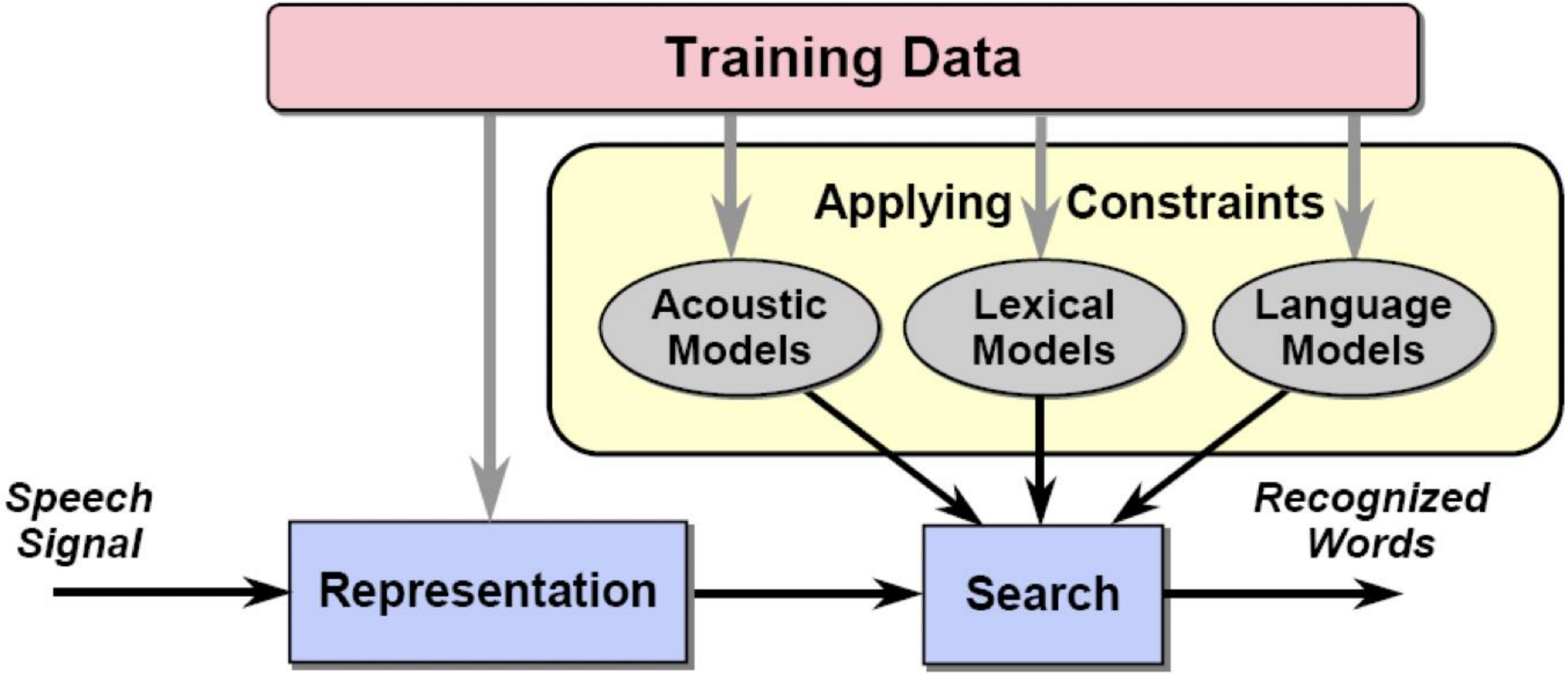
Source:https://sites.cs.ucsb.edu/~tyang/class/293S17/slides/TopicQA.pdf

NLP Application : Neural Machine Translation



Source: https://f417b5b8-a-62cb3a1a-s-sites.googlegroups.com/site/acl16nmt/home/nmt.png

NLP Application : Automatic Speech Recognition (ASR)



Source: http://www.iitk.ac.in/eclub/summercamp/Courses/Special%20Topics/Speech%20Processing.ppt

Languages can be tricky!

Source: https://www.lingq.com/blog/2019/02/08/english-language-memes/

Outline

Lecture 9: NLP & recurrent neural networks

- Natural language processing (NLP)
  - NLP Applications
  - Introduction and Text Normalization
  - Language Models
  - Word Vectors
- Recurrent neural network (RNN)
  - Vanilla RNN
  - LSTM
  - Bidirectional RNN
- RNN language models
  - Encoder-decoder (Seq2seq) model

Natural Language Processing

What is natural language processing (NLP)?

Natural language processing is at the intersection of:

- Computer science
- Artificial intelligence
- Linguistics

With interaction between:

- Computer
- Human languages

NLP Text Normalization

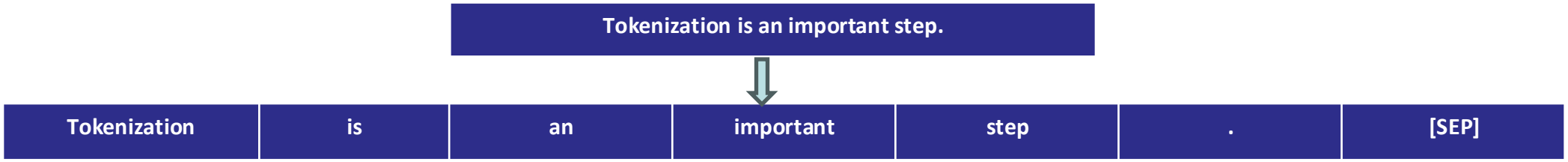
Text normalization is a crucial preprocessing step in natural language processing.

Some common tasks involved in text normalization are:

- **Tokenizing words**: Segmenting text into individual words.
- **Normalizing word formats**: Converting words to a standardized format (e.g., lowercasing, removing punctuation).
- **Lemmatization**: Determining that words like sang, sung, and sings share the same root, with "sing" as their common lemma, is the task of lemmatization.
- **Segmenting sentences**: Breaking the text into individual sentences.

Tokenization

- Tokenization is the process of dividing text into smaller units called tokens, which are typically words or sub-words.
- Tokens are mapped to vectors for use in neural networks.



Two Approaches :

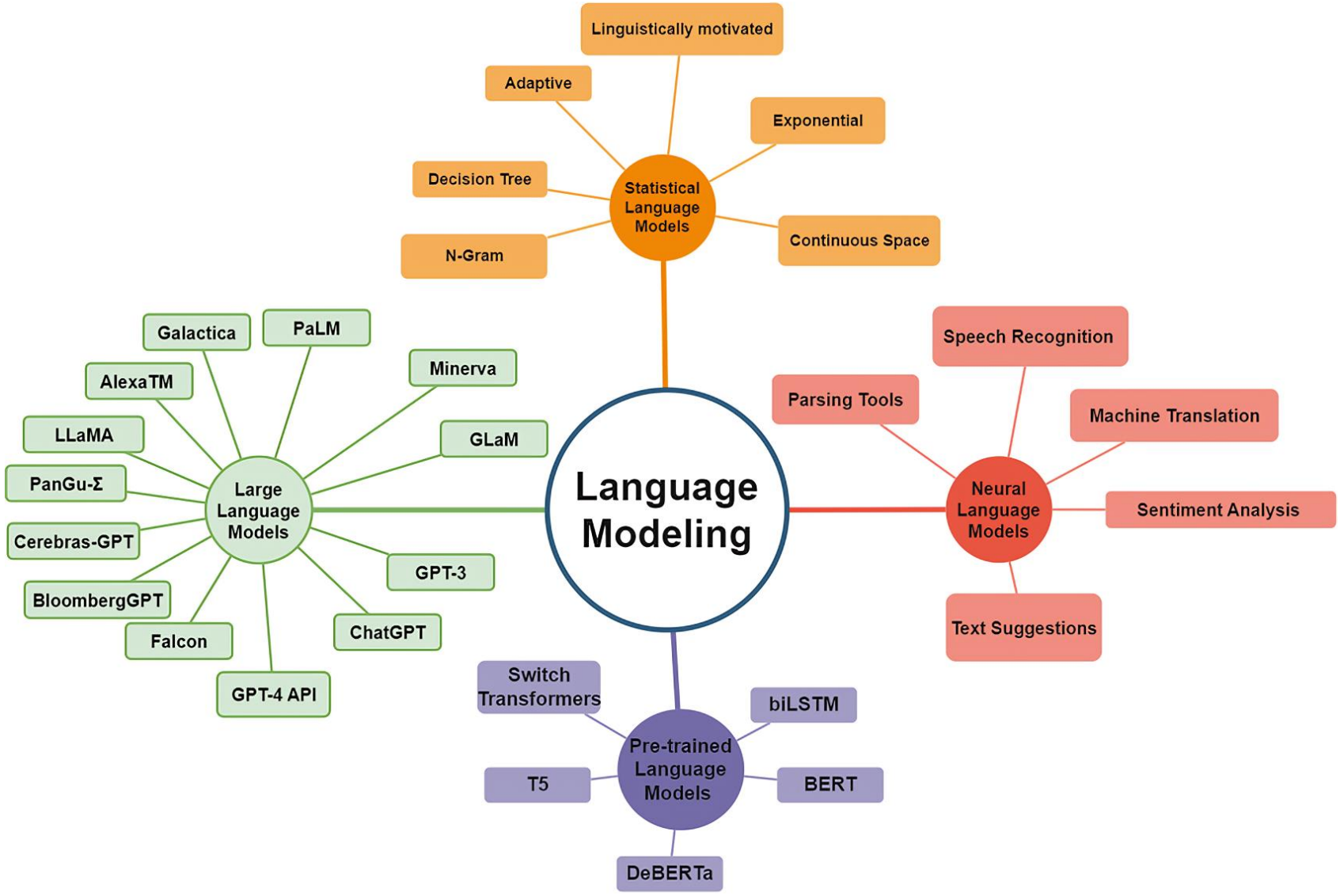
- **Top-Down (Rule-based tokenization)** uses predefined rules to segment text into tokens, typically based on grammar and syntax, e.g., splitting sentences at punctuation marks or spaces.
- **Bottom-up (Subword tokenization)** breaks down words into smaller units, such as subwords or characters, allowing for the handling of unknown words and variations, e.g., Byte Pair Encoding used in BERT and GPT.

Outline

Lecture 9: NLP & recurrent neural networks

- Natural language processing (NLP)
  - NLP Applications
  - Introduction and Text Normalization
  - Language Models
  - Word Vectors
- Recurrent neural network (RNN)
  - Vanilla RNN
  - LSTM
  - Bidirectional RNN
- RNN language models
  - Encoder-decoder (Seq2seq) model

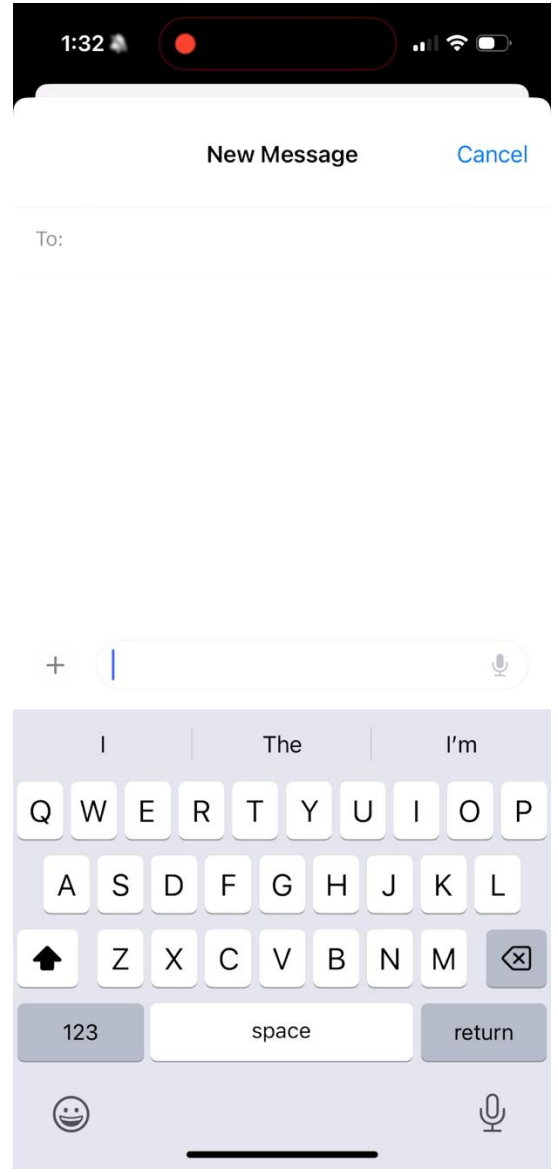
Language Models



https://www.linkedin.com/pulse/simplified-overview-language-modelslms-beginners-hannah-igboke-r4izf

Statistical Model : N-Gram

Task : Compute P(w|h), the probability of a word *w* given a history *h*



n-Gram Model Intuition : Instead of calculating the probability of a word given its entire history, we approximate it using only the last few words.

$$P(w_n \mid w_{1:n-1}) \approx P(w_n \mid w_{n-1})$$

- Language models can be used for prediction
  e.g., text completion on your phone : while typing "ineff...", the model predicts "inefficient" or "inefficiency."

Predicting Next Words

N-gram models: Sequence of 'n' words used for prediction.
Prediction: Next word based on previous 'n-1' words.

Types of N-grams:
Bigram (n=2) : Predicts next word based on the previous word.
Trigram (n=3): Predicts next word based on the previous two words.

Example
Phrase: "The dog barks"
Prediction (Trigram): "loudly"
Reason: "dog barks loudly" is a common phrase in training data.

Modern approach include RNN, LSTM and Transformers to predict next words.

Evaluating Language Models

- **Perplexity (PPL)**
  - Perplexity measures how well a language model predicts human words.
  - Given a sentence $S = (W_1, W_2, \dots, W_k)$, its probability is written as

$$P(S) = P(W_1, W_2, \dots, W_k) = p(W_1)P(W_2|W_1) \dots P(W_k|W_1, W_2, \dots, W_{k-1})$$

  - A lower perplexity score signifies that the language model assigns higher probabilities to the words in the test set, indicating better predictions.

$$PP(W) = P(W_1, W_2, \dots, W_k)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(W_1, W_2, \dots, W_k)}}$$

- **BLEU score**
  - Bilingual evaluation understudy score is used to evaluate the quality of text which has been machine translated.
  - It takes the length of sentence into consideration.
  - See https://en.wikipedia.org/wiki/BLEU for details.
- **CIDEr score**
  - Consensus-based image description evaluation
  - See https://arxiv.org/pdf/1411.5726.pdf for details.

Going Beyond N-grams

Limitations on N-gram models:

- Limited context awareness.
- Inability to capture complex relationships between words.

Modern Approaches: Neural Networks

Deep learning models use vector representations called embeddings, which capture word context and relationships for better language understanding.

Outline

Lecture 9: NLP & recurrent neural networks

- Natural language processing (NLP)
  - NLP Applications
  - Introduction and Text Normalization
  - Language Models
  - Word Vectors
- Recurrent neural network (RNN)
  - Vanilla RNN
  - LSTM
  - Bidirectional RNN
- RNN language models
  - Encoder-decoder (Seq2seq) model

Word vectors

Represent words (“motel” and “hotel”) with sparse 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]

BUT …

- These two vectors are orthogonal. We cannot explore the similarity between ‘motel’ and ‘hotel’.
- Vocabulary usually contains more than 50k words. Storing the one-hot word vectors will be memory hungry.

So, we should

- Learn to encode similarity into word vectors
- Reduce the dimension of word vectors

Word Vectors

Core ideas: A word's meaning can be represented using other words surrounding it (context). We use *context* to build up the meaning of each *word* and encode *it* in the word vector.

Example:

The quick brown **fox** jumps over the lazy dog.

Using nearby contexts, we can build a dense vector for the word **fox**:

$$\text{fox} = \begin{bmatrix} 0.142 \\ -0.920 \\ -0.011 \\ 0.024 \\ 0.815 \\ -0.912 \end{bmatrix} \quad \text{jump} = \begin{bmatrix} -0.997 \\ 0.239 \\ -0.004 \\ 0.958 \\ -0.235 \\ -0.811 \end{bmatrix}$$

This is also called word embedding!

$< \text{fox}, \text{jump} >=?$

Word Vectors

- Another Example for Word Vectors or Embedding

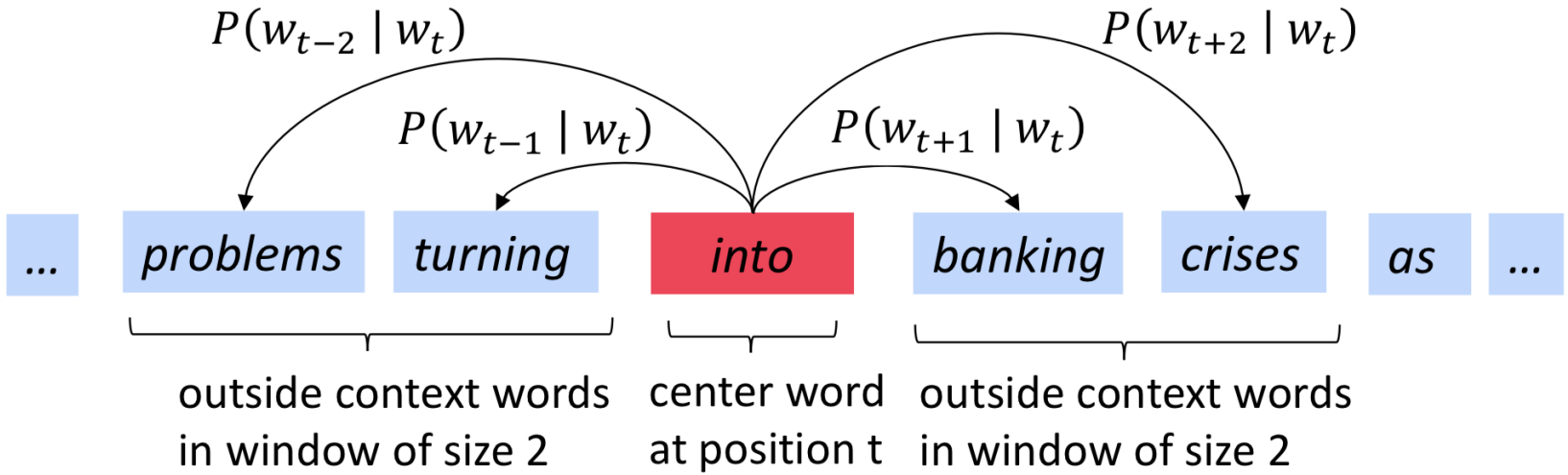
		King	Queen	Prince	Man
Royalness		0.99	0.97	0.91	0.01
Sex		0.95	-0.5	0.93	0.94
Age		0.32	0.41	0.81	0.33
	...	...	...	...	...

Pre-trained Embeddings are trained on large corpus of text. Some Examples :

- Word2Vec (Developed at Google)
- GloVe (Developed at Stanford University)
- FastText (Developed at Meta)

Word2Vec

Define the probability of a word



$P(w_{t+1}|w_t)$: probability of word w_{t+1} given word w_t.

Mikolov et al., Distributed Representations of Words and Phrases and their Compositionality (2013)
Credit: Stanford CS224N

Word2Vec

Objective function

- Compute the likelihood by multiplying all the probabilities in the sequence

$$L(\theta) = \prod_{t=1}^{T} \prod_{j=-m, j\neq 0}^{m} P(w_{t+j}|w_t;\theta)$$

- Use negative log likelihood to generate the cost function.

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T} \sum_{j=-m, j\neq 0}^{m} LogP(w_{t+j}|w_t;\theta)$$

θ is all the variables to be optimized.

- But how to calculate $P(w_{t+j}|w_t;\theta)$?

Use two vectors per word: use v_w when *w* is a center word; use u_w when *w* is a context word. Then, for a center word w=c and a context word w=o, we have:

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

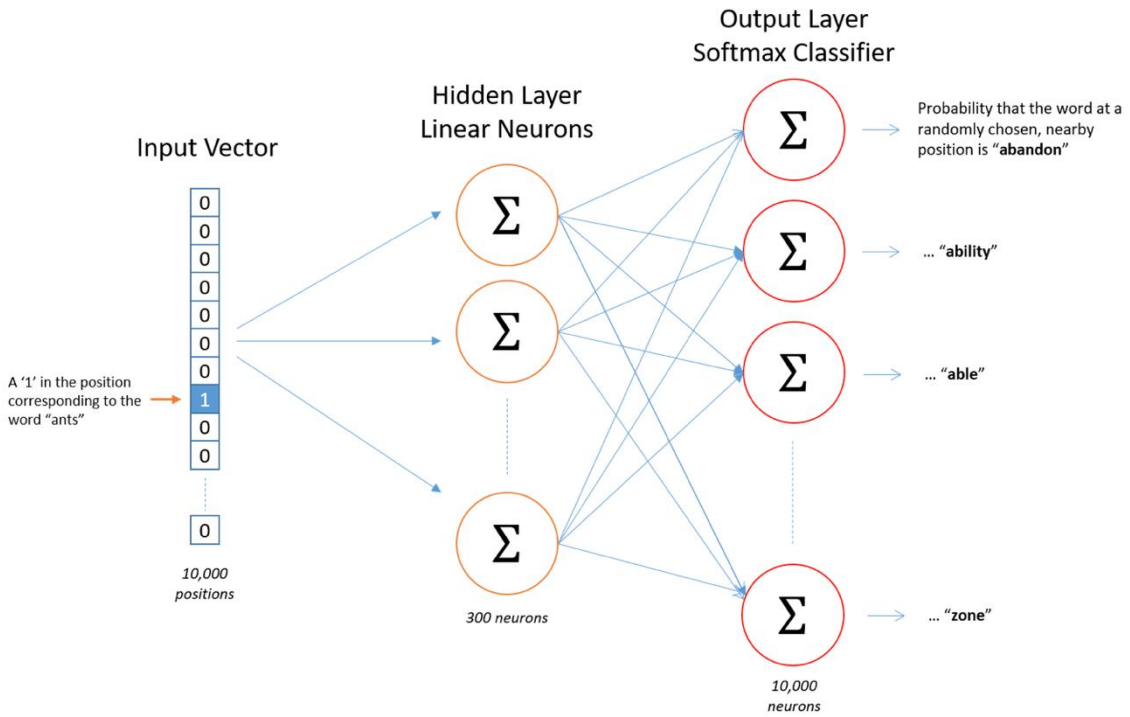
Word2Vec

- **Final step:** optimize θ by minimizing $J(\theta)$.

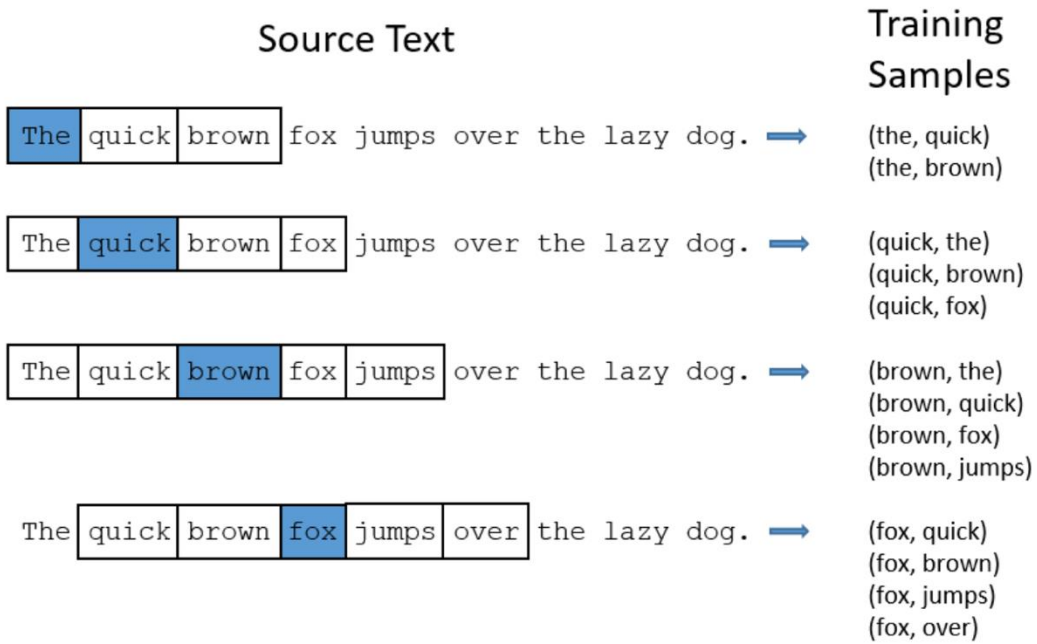
$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{j=-m, j \neq 0}^{m} logP(w_{t+j}|w_t; \theta)$$

Training Objective



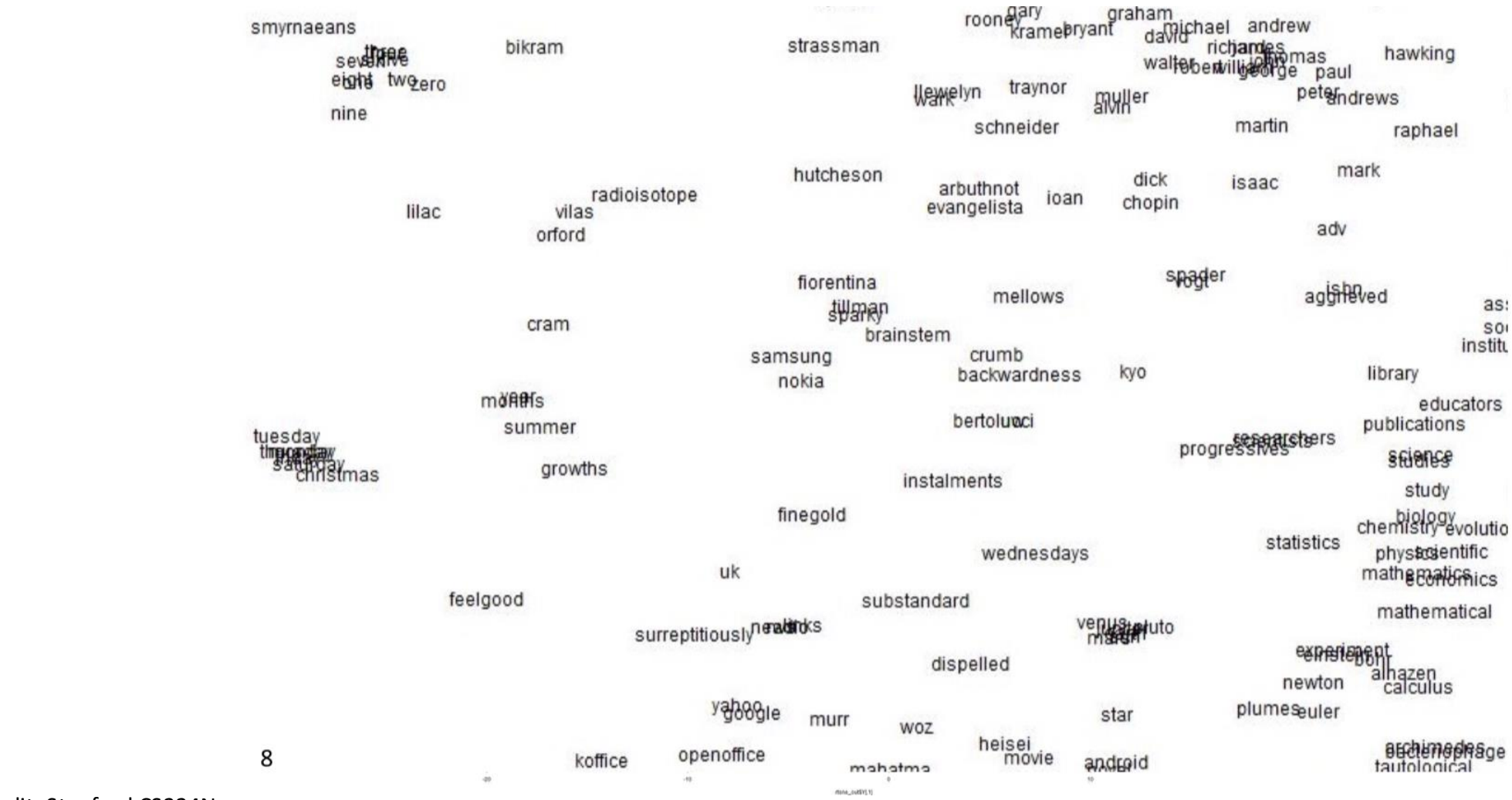
Use word-pair as training examples.



Credit: Stanford CS224N

Word2Vec

Visualization



Credit: Stanford CS224N

GloVe

Global Vectors for word representation (GloVe)

- Use global matrix factorization and local context windows to generate word embeddings.
- A count-based model by using nearest neighbors/k-means rather than predictive model (i.e., Word2Vec)

Pretrained GloVe models are available here:
https://nlp.stanford.edu/projects/glove/

Pennington et. al., “GloVe: Global Vectors for Word Representation”

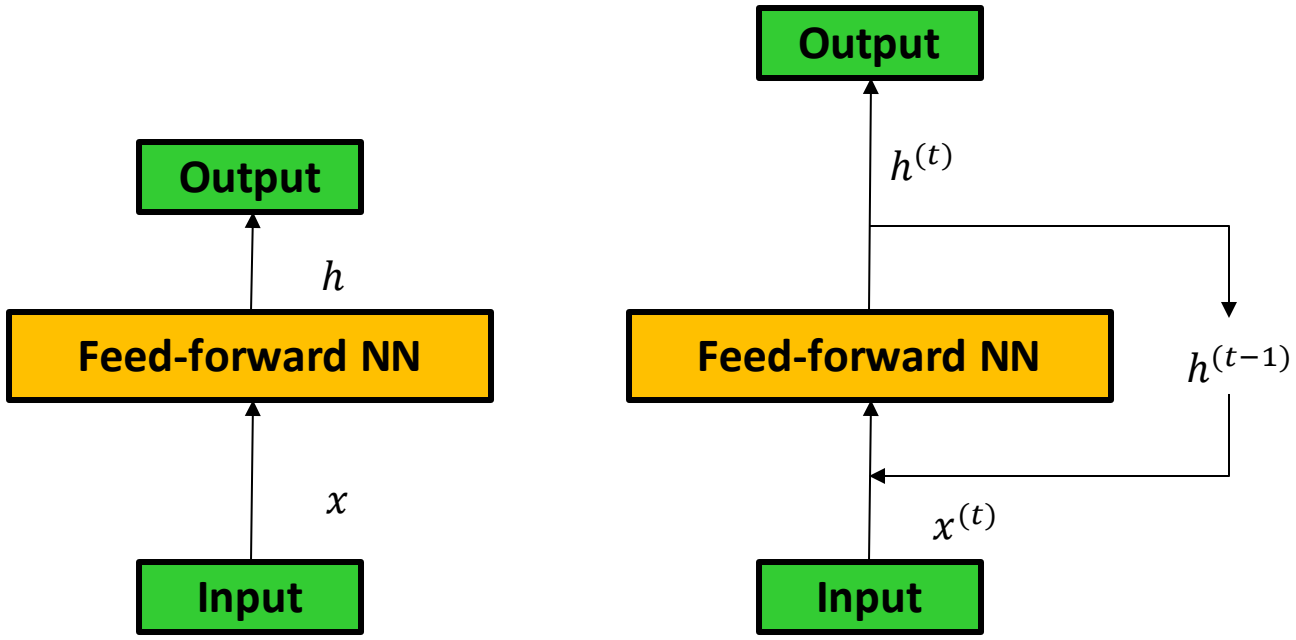
Outline

Lecture 9: NLP & recurrent neural networks

- Natural language processing (NLP)
  - NLP Applications
  - Introduction and Text Normalization
  - N-gram Language Models
  - Word Vectors
- Recurrent neural network (RNN)
  - Vanilla RNN
  - LSTM
  - Bidirectional RNN
- RNN language models
  - Encoder-decoder (Seq2seq) model

Vanilla RNN

- Vanilla RNNs are designed for sequential models.



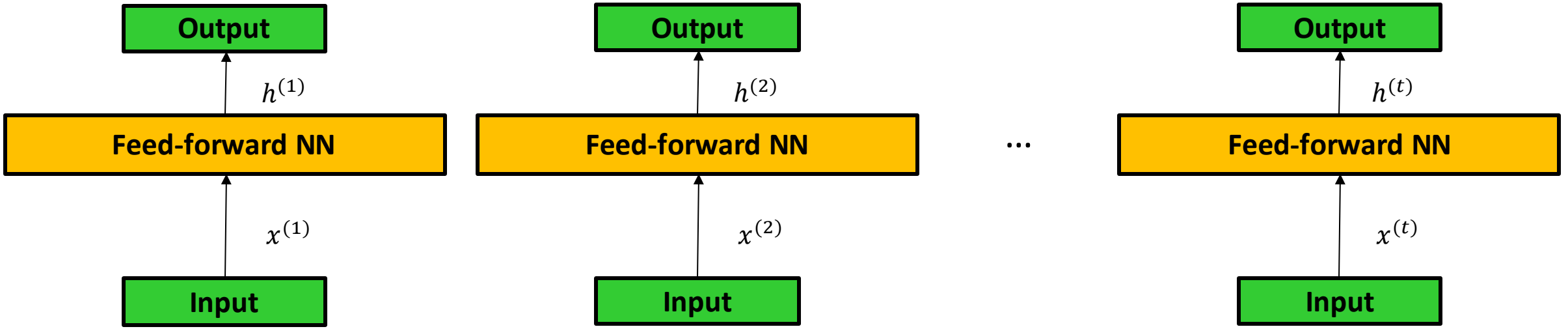
Vanilla NN

Vanilla RNN

Output hypothesis will be an input feature during the forward propagation of next time step.

Vanilla NN

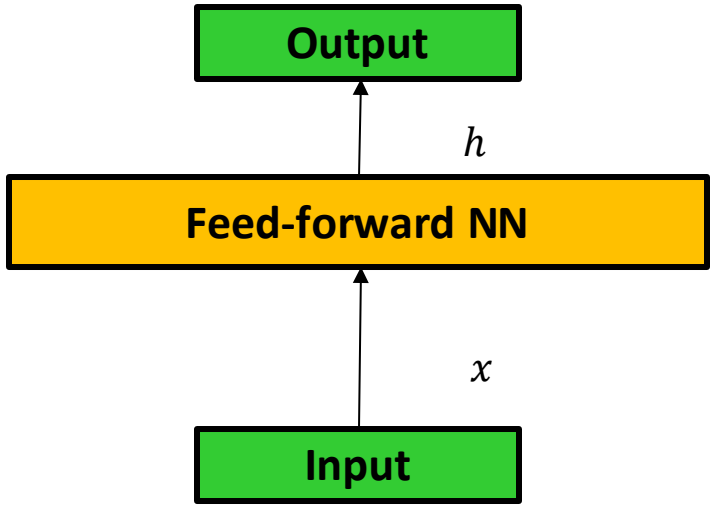
- Vanilla NNs are not able to handle sequential models well.
- They have to take ‘glimpses’ into each frame in the time dimension.



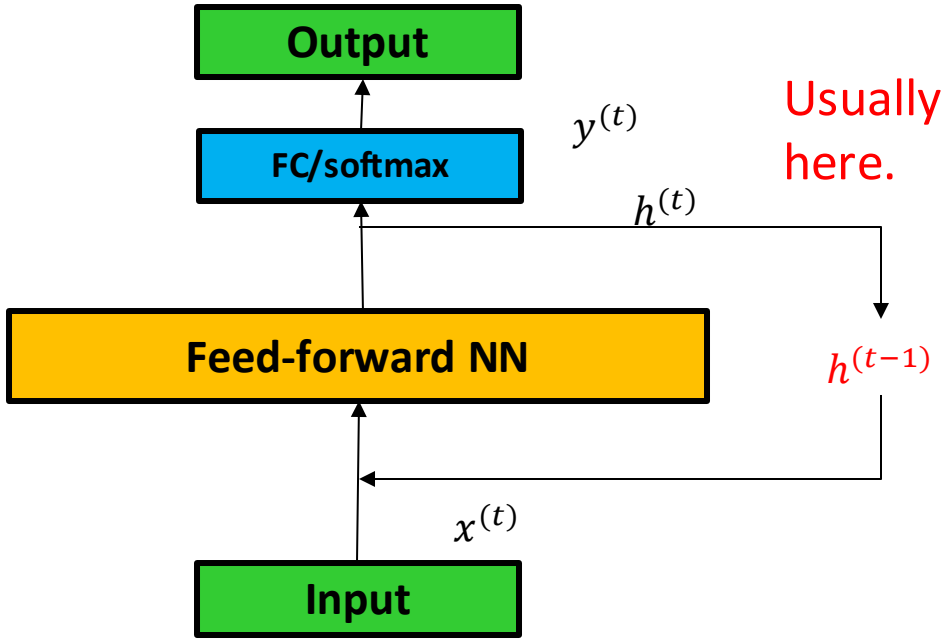
Vanilla NNs cannot share information between time dimension,
so they are prone to overfitting in processing sequential data.

Vanilla RNN

- Vanilla RNNs are designed for sequential models.



Vanilla NN



Usually want to predict something here.

hidden state

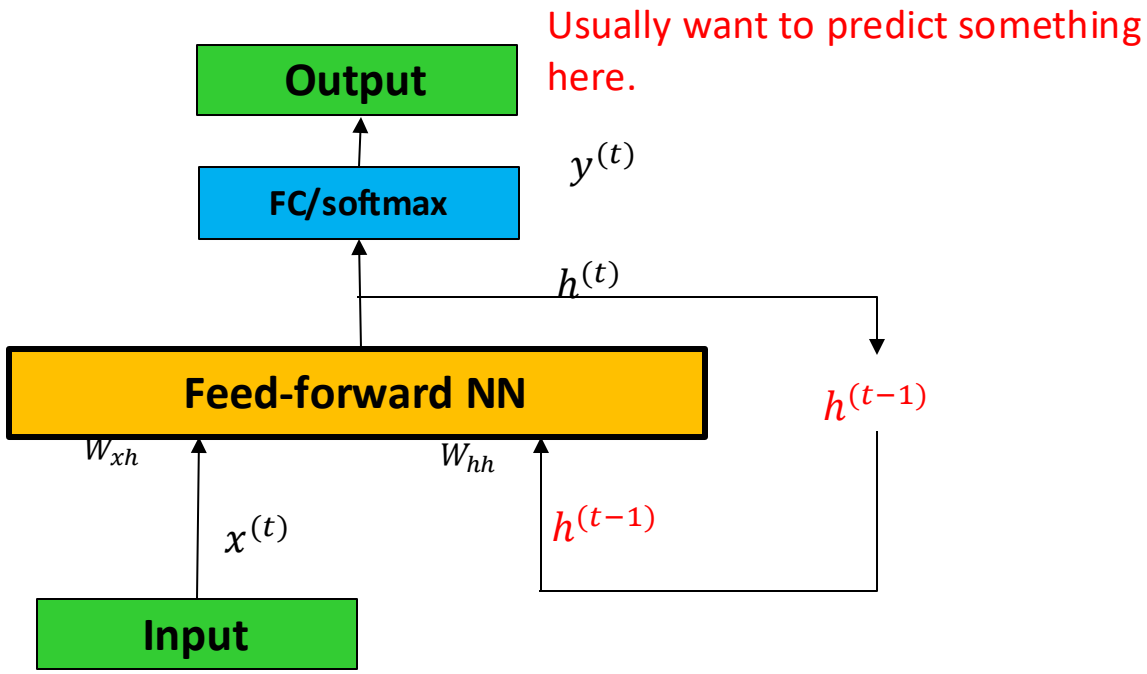
Vanilla RNN

Output hypothesis will be an input feature during the forward propagation of next time step.

Vanilla RNN

Vanilla RNN forward pass

The hidden state at the previous time stamp $h^{(t-1)}$ is also used to generate the current time hidden state $h^{(t)}$.



Forward Propagation

$$h^{(t)} = f_W(h^{(t-1)}, x)$$

Activations $h^{(1)}, h^{(2)}, \ldots, h^{(t)}$ are computed through time.

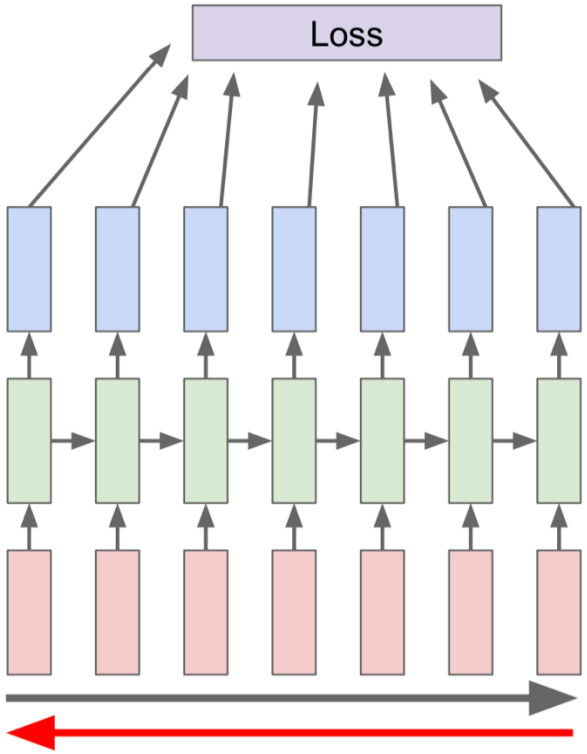
$$h^{(t)} = \tanh(W_{xh}x^{(t)} + W_{hh}h^{(t-1)} + b_h)$$

$$y^{(t)} = \sigma(W_{hy}h^{(t)} + b_y)$$

Vanilla RNN: BPTT

Vanilla RNN backward pass

- Back propagation through time (BPTT)



Forward through the entire sequence to compute loss, and **backward** through the entire sequence to compute the gradient.

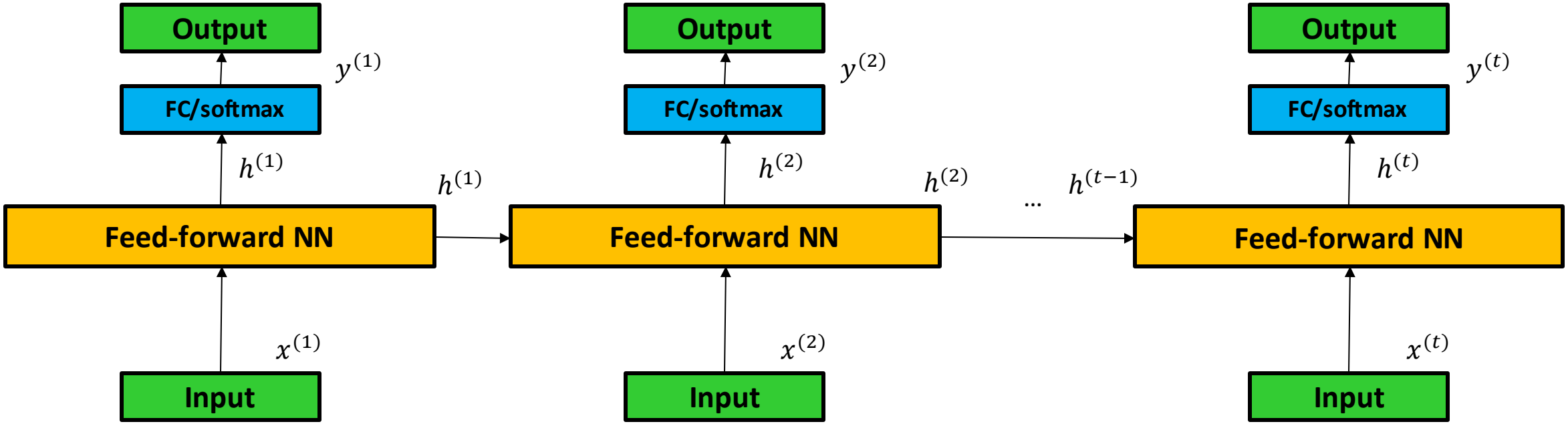
```
Back_Propagation_Through_Time(a, y)   // a[t] is the input at time t. y[t] is the output
    Unfold the network to contain k instances of f
    do until stopping criteria is met:
        x = the zero-magnitude vector;// x is the current context
        for t from 0 to n - k         // t is time. n is the length of the training sequence
            Set the network inputs to x, a[t], a[t+1], ..., a[t+k-1]
            p = forward-propagate the inputs over the whole unfolded network
            e = y[t+k] - p;           // error = target - prediction
            Back-propagate the error, e, back across the whole unfolded network
            Sum the weight changes in the k instances of f together.
            Update all the weights in f and g.
            x = f(x, a[t]);           // compute the context for the next time-step
```

Source: https://en.wikipedia.org/wiki/Backpropagation_through_time
Credit to Stanford CS224N

Vanilla RNN

Unfold Vanilla RNN during computation

- Vanilla RNNs can be unfolded to a stack of feed-forward NNs along the time dimension.

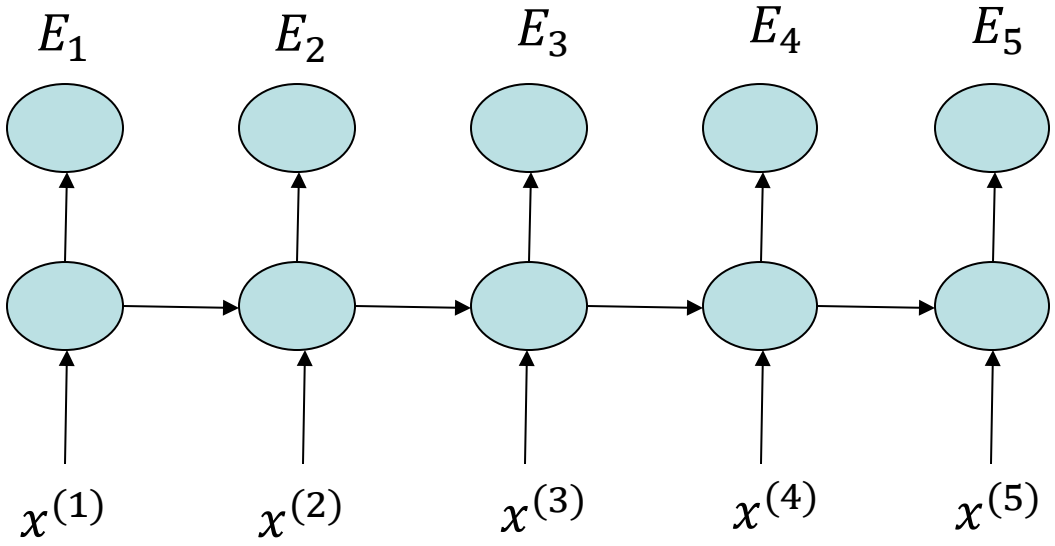


$$h^t = f_W(h_{t-1}, x)$$

Gradient explosion & vanishing

Case study: RNN with t = 5 time steps

Due to the properties of the activation function (sigmoid, tanh), it is easy to see the gradient explosion/vanishing problem.



$$\frac{\partial E_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial E_t}{\partial h^{(t)}} \prod_{j=k+1}^{t} \left(\frac{\partial h^{(j)}}{\partial h^{(j-1)}}\right) \frac{\partial h^{(k)}}{\partial W}$$

$\left\| \frac{\partial h^{(j)}}{\partial h^{(j-1)}} \right\| > 1$: gradient explosion

$\left\| \frac{\partial h^{(j)}}{\partial h^{(j-1)}} \right\| < 1$: gradient vanishing

How to avoid gradient explosion/vanishing?

- Use ReLU
- Use LSTM

We usually use an RMSprop optimizer to train RNN models.

RMSprop (root mean square propagation)

To address the problem in Adagrad, RMSprop uses a moving average of incoming gradients as the denominator.

$$G^{t+1} = \mu G^t + (1-\mu)\left(\nabla_W L(W^t)\right)^2$$

$$W^{t+1} = W^t - \frac{\alpha \nabla_W L(W^t)}{\sqrt{G^{t+1} + \epsilon}}$$

α: Learning rate
μ: Momentum (factor)
G^t: gradient norm at time step t.
ϵ : Small number to avoid zero division

Advantages:

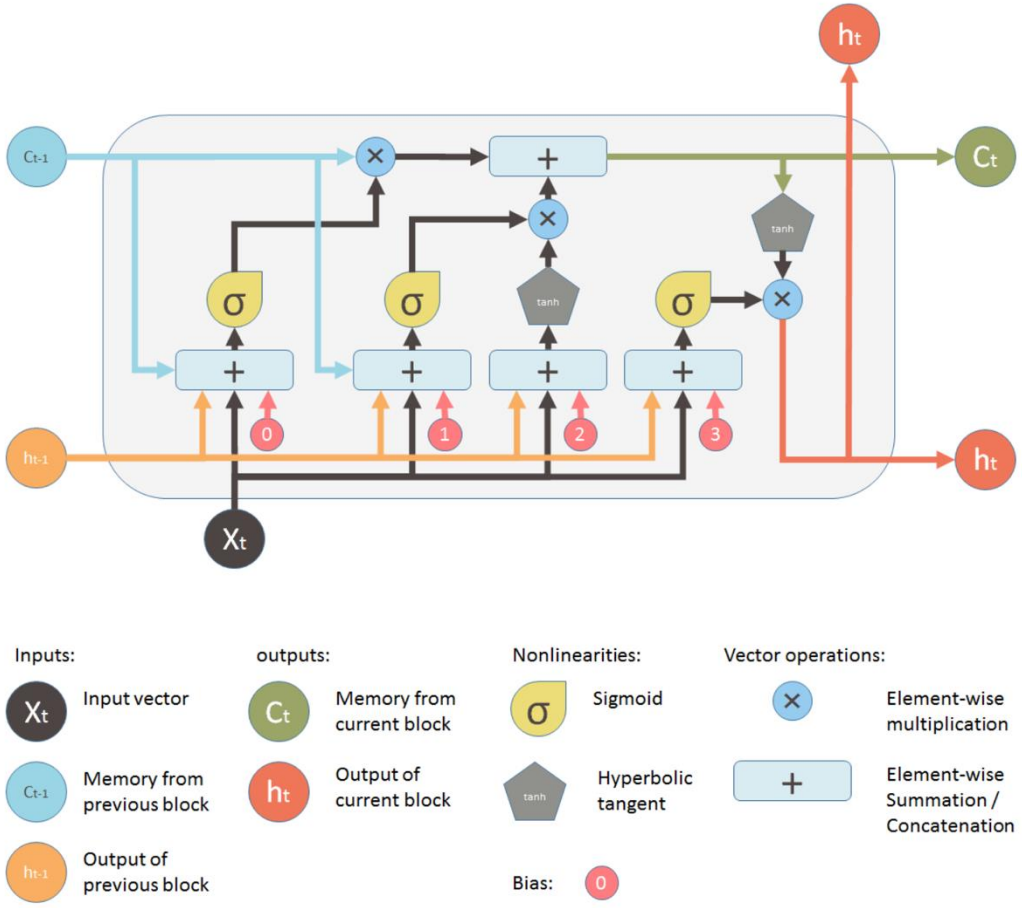
+ Prevent learning rate shrinkage
+ Good for recurrent neural networks

Problem:

- Oscillation at the end of convergence.

Long short-term memory (LSTM)

LSTM is designed to address the ‘gradient vanishing’ problem. Memory cells and input/forget/output gates help LSTM give more precise predictions with historical information.

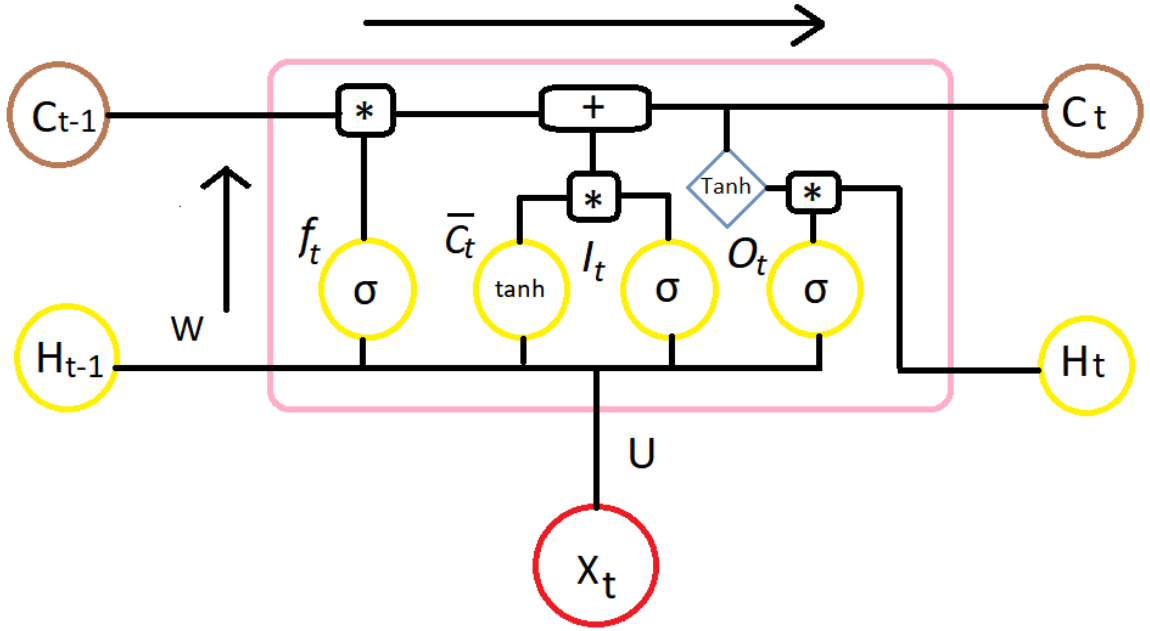


BUT…

- LSTM needs a larger dataset to train.
- LSTM takes a longer time to train and usually has a very large model size.

Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." (1997)

Long short-term memory (LSTM)



LSTM will always try to figure out how much history information it needs to memorize.

Forget gate

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

Input gate

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

Memory cell

$$\bar{C}_t = tanh(W_c[h_{t-1}, x_t] + b_c)$$

$$C_t = f_t \circ C_{t-1} + i_t \circ \bar{C}_t$$

Output gate & hypothesis

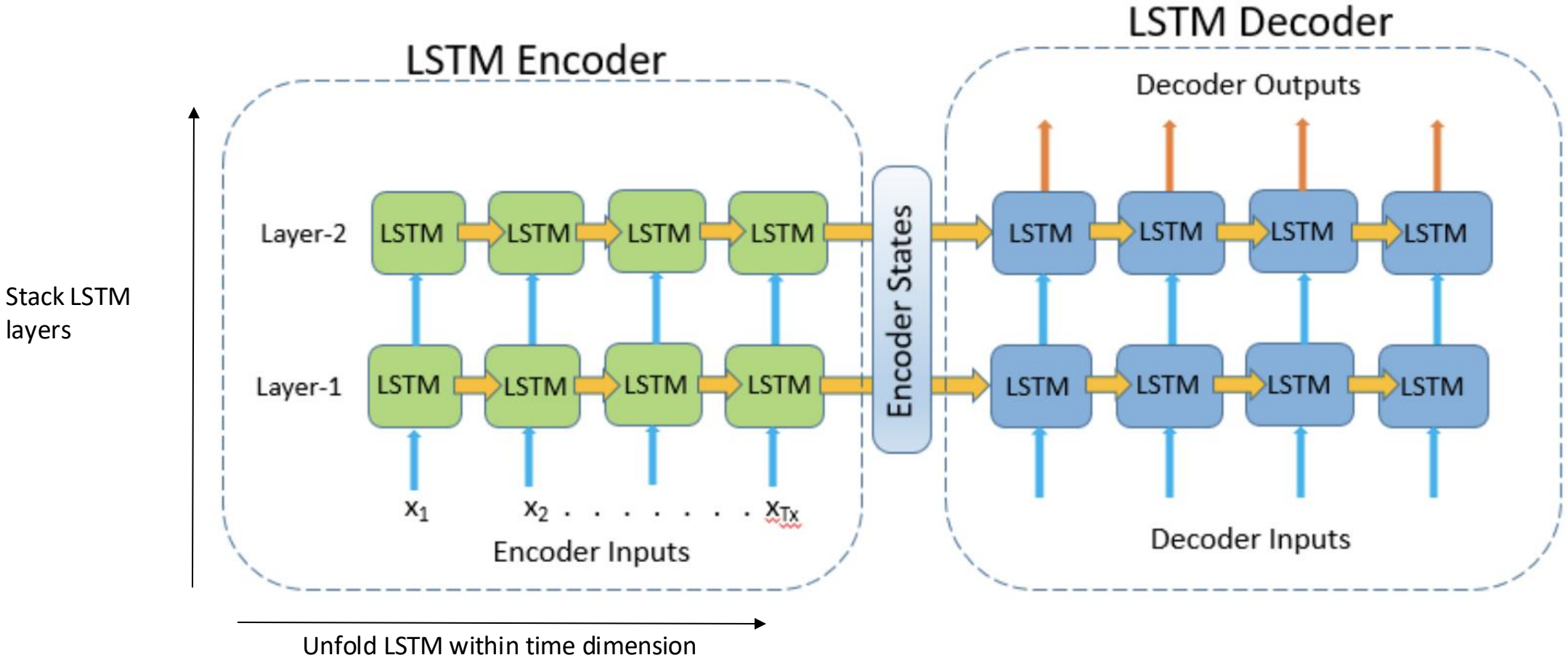
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \circ \tanh(C_t)$$

Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." (1997)

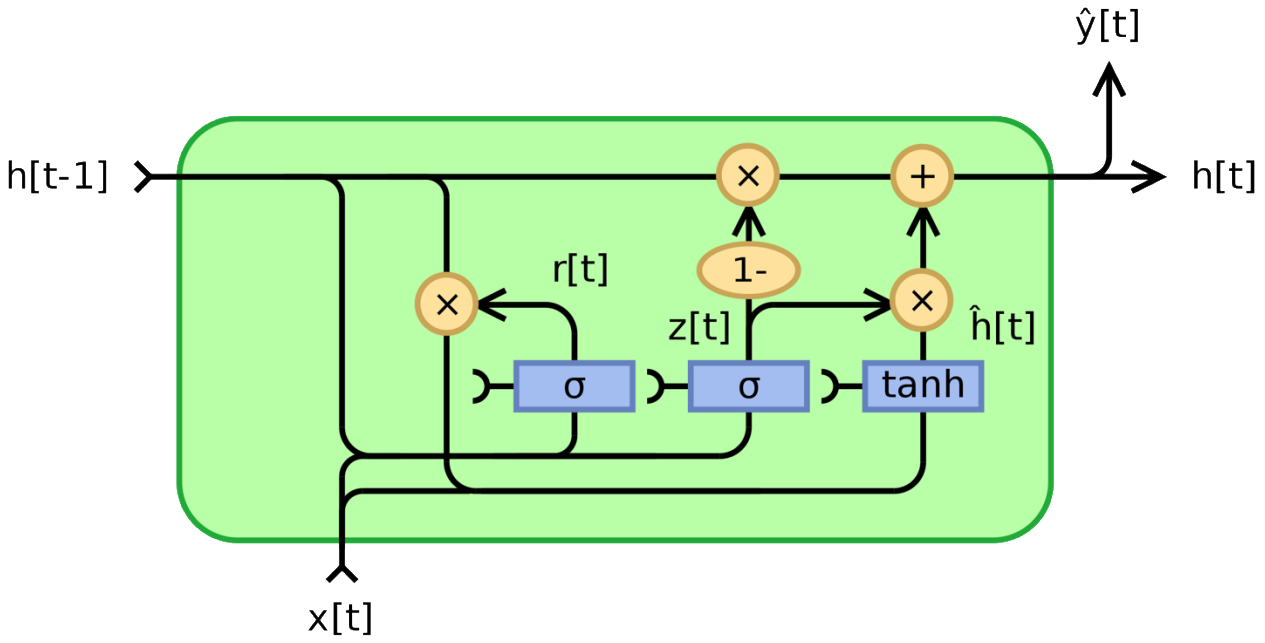
Long short-term memory (LSTM)

It is a common practice to stack multiple LSTM models to construct very deep LSTM models for language tasks.



Gated recurrent unit (GRU)

- A simplified version of LSTM cell with ability to learn more meaningful word representations.



Update gate

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$

Reset gate

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

Memory content

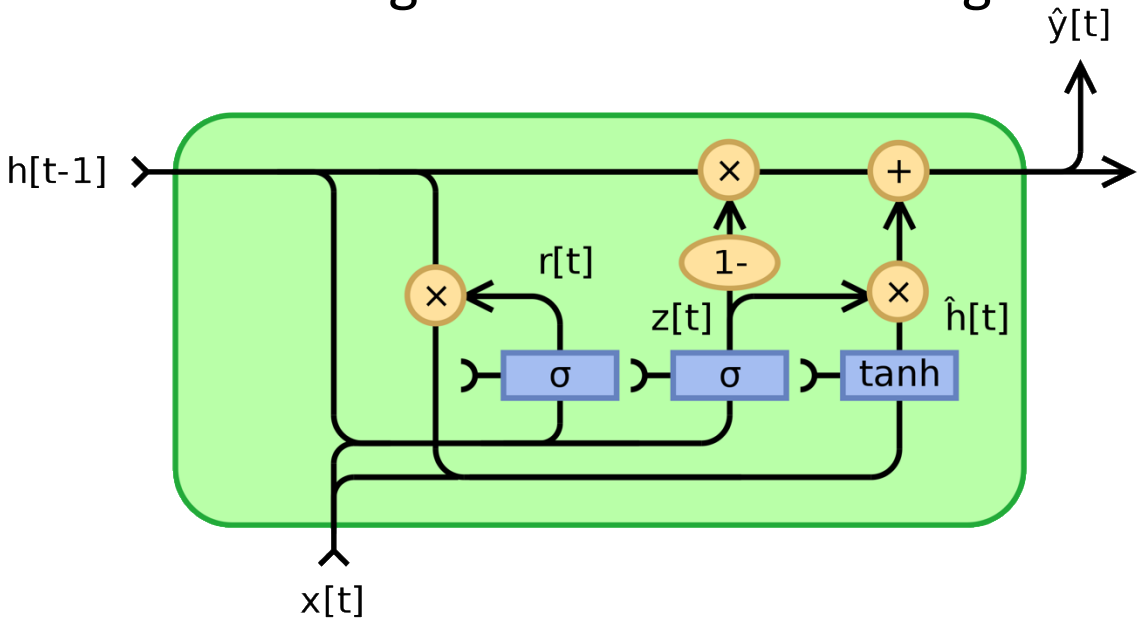
$$\hat{h}_t = \sigma(W^{(h)}x_t + r_t U^{(h)}h_{t-1})$$

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \hat{h}_t$$

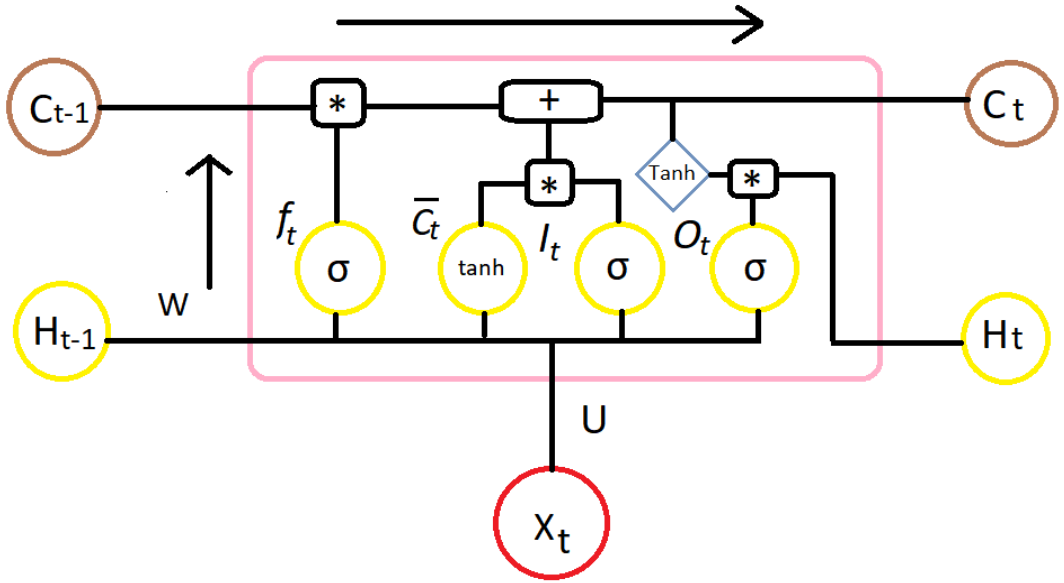
Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling." (2014)

Gated recurrent unit (GRU)

- GRU is an improvement over LSTM as:
  - It merges the input/forget data into one single update gate
  - No extra memory cell (c_t)
  - Simplified control dependencies with fewer gate operations
  - It has a significant lower training cost



GRU

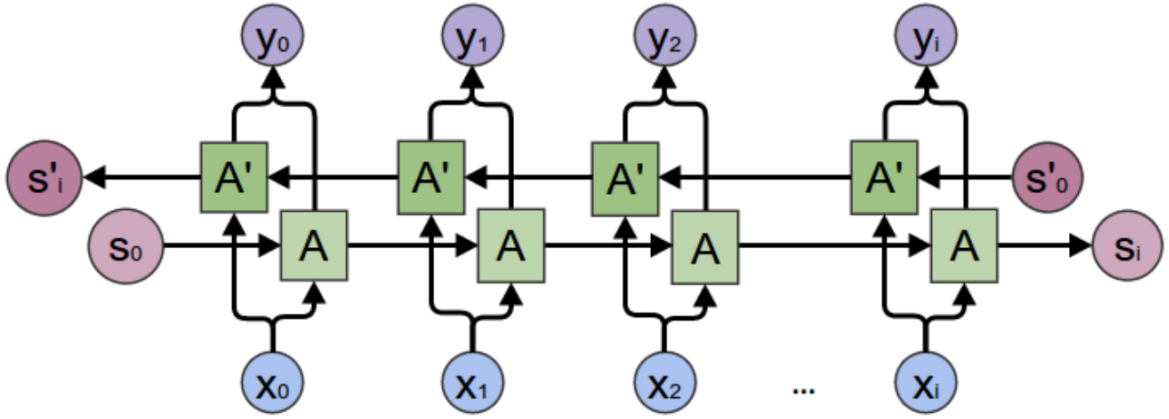


LSTM

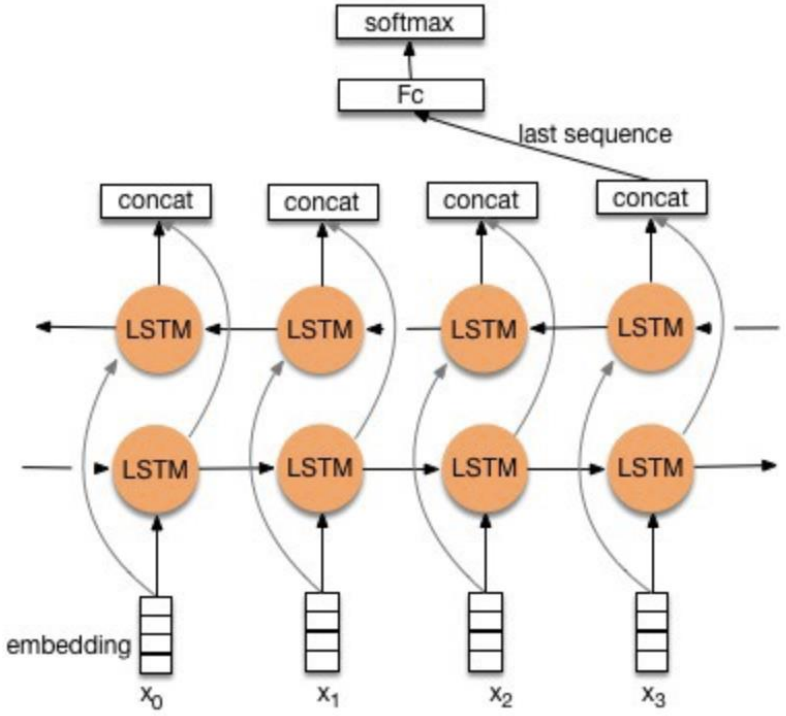
Bidirectional RNNs

Bidirectional RNN concatenates both forward hidden state and backward hidden state during prediction.

It puts two independent recurrent models together and performs two forward pass in the opposite direction.



Bidirectional vanilla RNN



Bidirectional LSTM

Schuster, Mike, and Kuldip K. Paliwal. "Bidirectional recurrent neural networks." (1997)

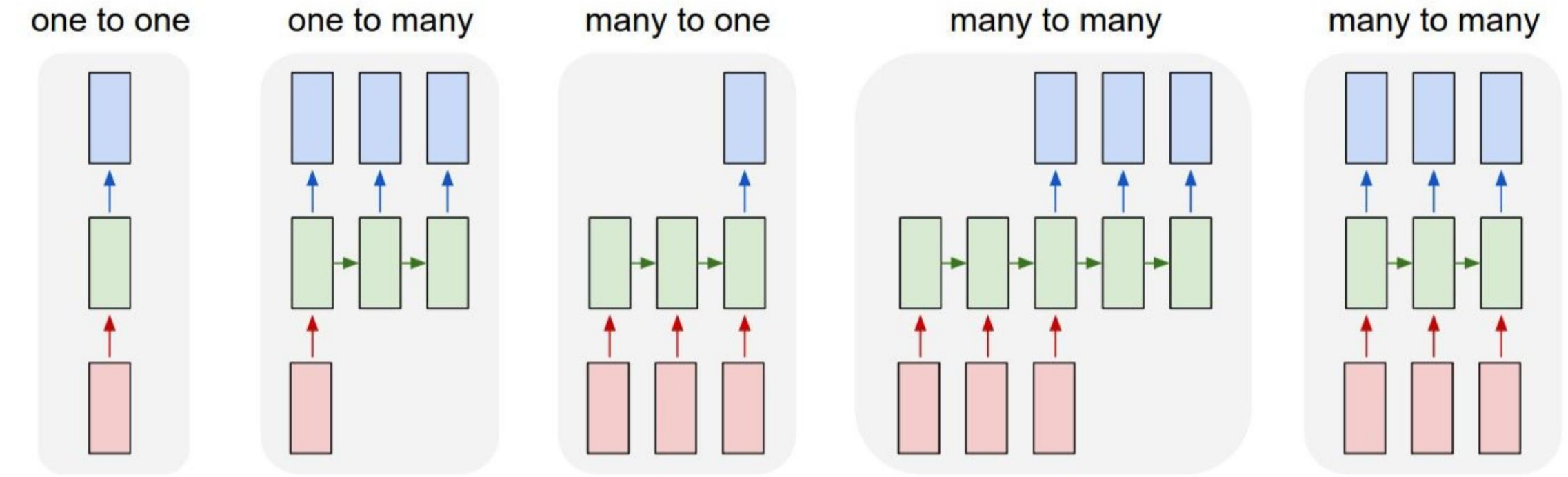
Outline

Lecture 9: NLP & recurrent neural networks

- Natural language processing (NLP)
- Recurrent neural network (RNN)
  - Vanilla RNN
  - LSTM
  - Bidirectional RNN
- RNN language models
  - Encoder-decoder (Seq2seq) model

RNN models

Overview: Types of RNN tasks



Trivial non-sequence model

Sequence generation

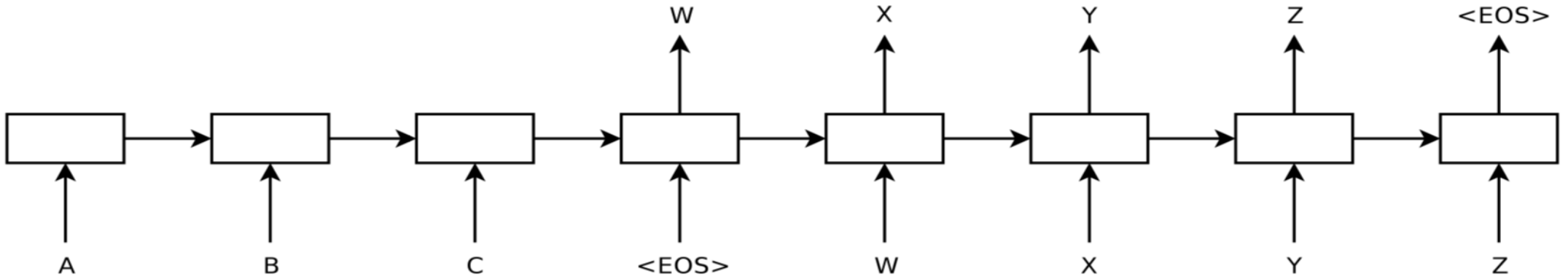
Sentiment classification

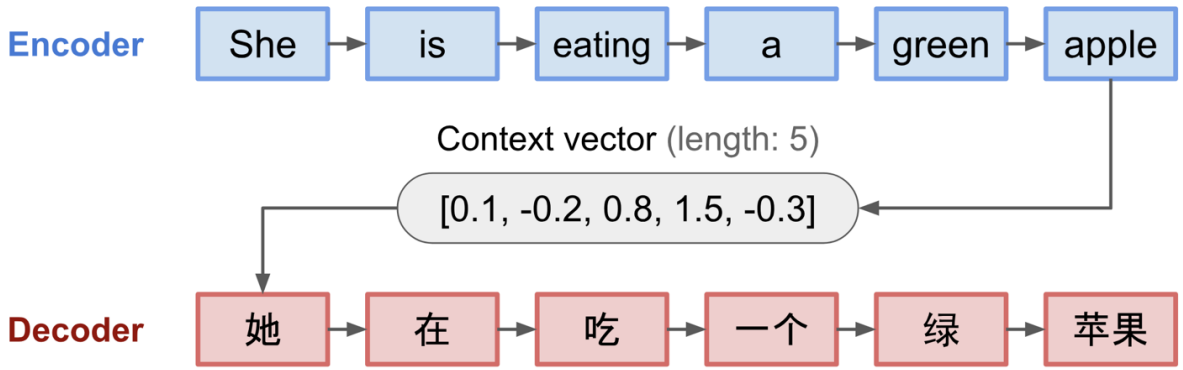
Neural machine translation

Credit: Stanford CS224n

RNN models

Encoder-decoder model (Seq2Seq)



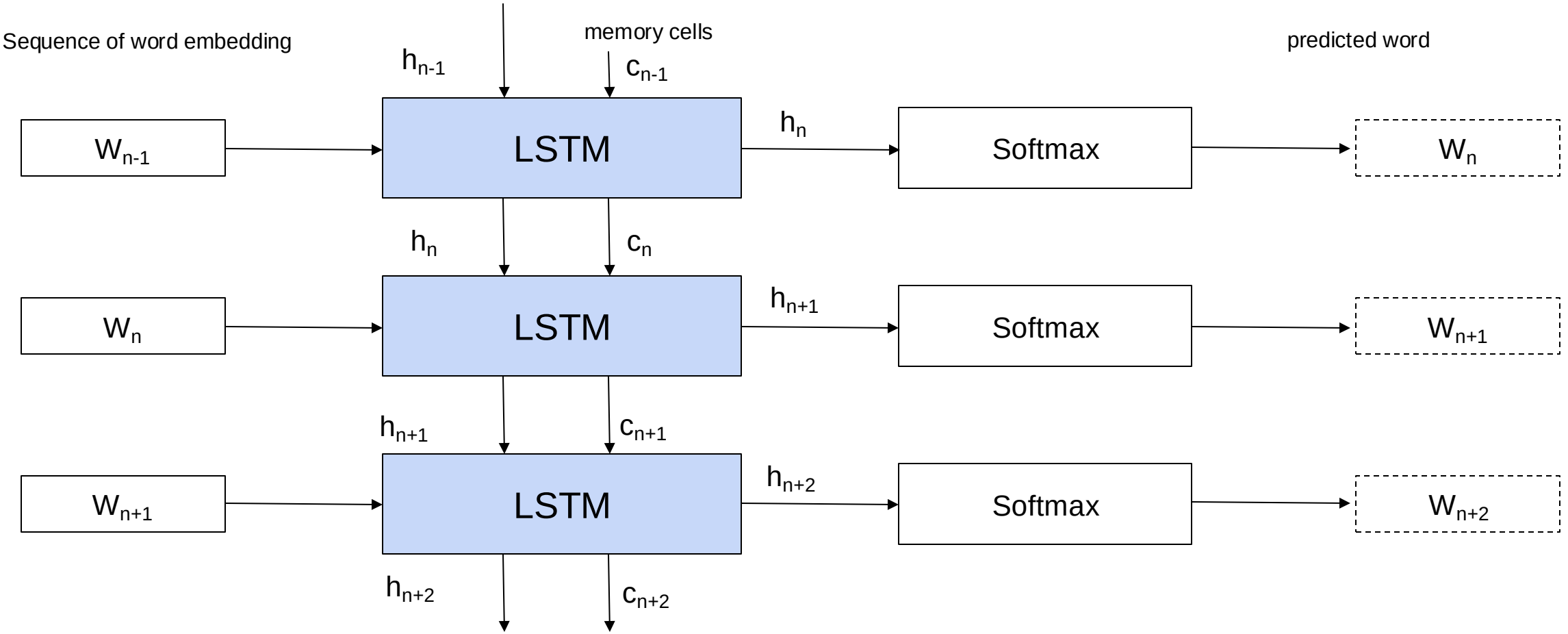


Core idea:

1. Map the input sequence with RNN to a fixed-sized vector.
2. Map the vector back to the target sequence with another RNN.

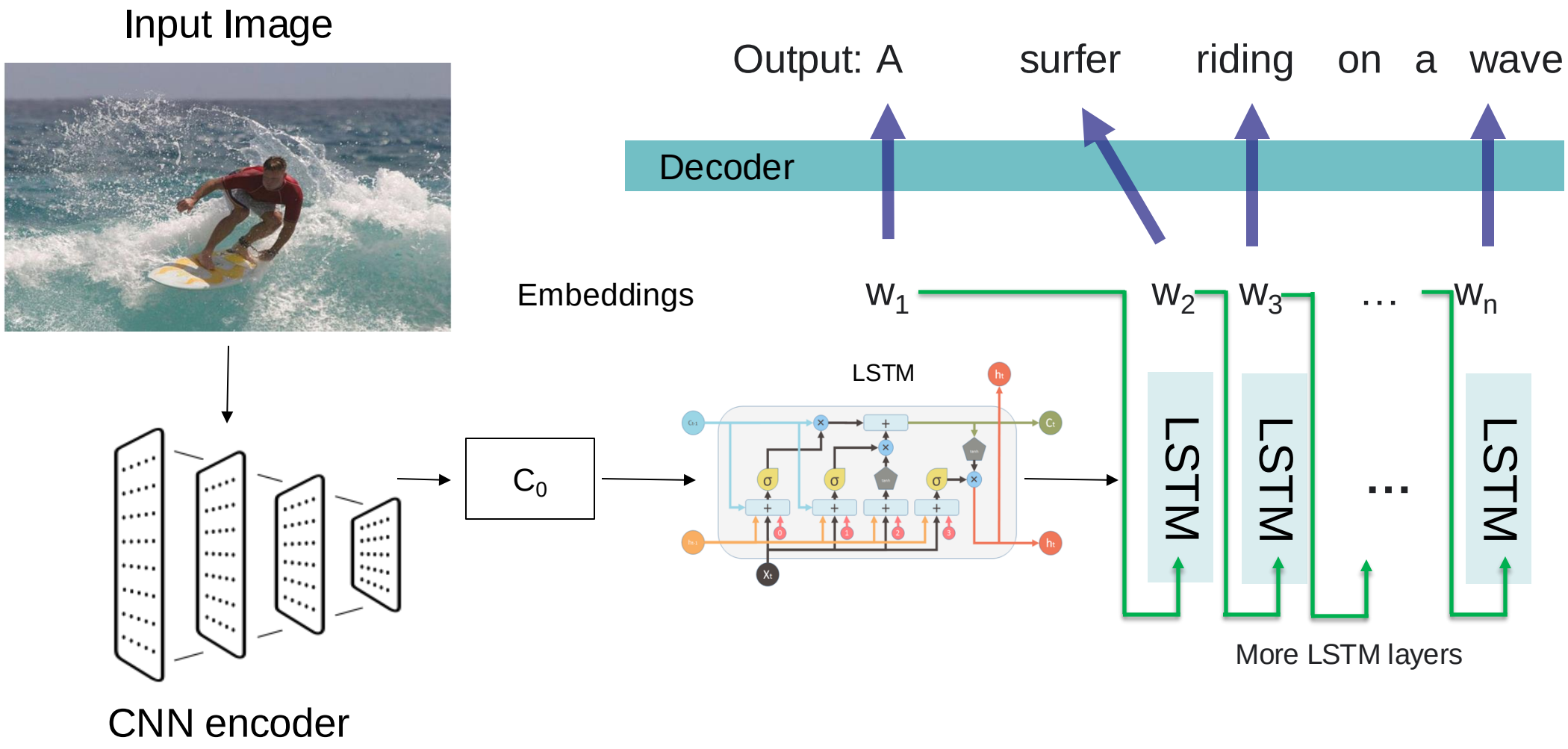
Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." (2014)

LSTM models



$X_{n-1}=[w_{n-1},h_{n-1}]$

Application: Image to Caption



In this lecture, we learned:

- Overview of Natural Language Processing
- Recurrent neural network (RNN)
  - Vanilla RNN
  - LSTM
  - Bidirectional RNN
- RNN language models
  - Encode-Decoder (Seq2Seq) model

Further Reading

- Here is a list of classic and trending papers on RNN and language models:
  - Graves, Alex. "Generating sequences with recurrent neural networks." (2013)
  - Le, Quoc, and Tomas Mikolov. "Distributed representations of sentences and documents." (2014)
  - Weston, Jason, Sumit Chopra, and Antoine Bordes. "Memory networks." (2014)
  - Amodei, Dario, et al. "Deep speech 2: End-to-end speech recognition in English and Mandarin." (2016)

In next lecture, we will learn:

Lecture 10: Attention-based models and LLM Inference

- Self-attention
- Transformer
- Vision Transformer
- Large Language Models