國立臺灣大學
National Taiwan University

Sequence Modeling & Embeddings

Yun-Nung (Vivian) Chen
http://vivianchen.idv.tw

N L P

Southeast Asia Machine Learning School 2019

http://seamls.miulab.tw/

Sequence Modeling & Embeddings

Yun-Nung (Vivian) Chen
http://vivianchen.idv.tw

National Taiwan University

- Word Representation Basics
- Word Embeddings
- Recurrent Neural Network

National Taiwan University

Sequence Modeling & Embeddings

Yun-Nung (Vivian) Chen
http://vivianchen.idv.tw

- Word Representation Basics
- Word Embeddings
- Recurrent Neural Network

Learning Target Function

- Classification Task

$$f(x) = y \quad \Rightarrow \quad f : R^N \to R^M$$

- x: input object to be classified → a N-dim vector
- y: class/label → a M-dim vector

Assume both x and y can be represented as fixed-size vectors

"This is awesome!" ⟶ +

"It sucks." ⟶ -

How do we represent the meaning of the word?

Meaning Representations

- Definition of “Meaning”
  - the idea that is represented by a word, phrase, etc.
  - the idea that a person wants to express by using words, signs, etc.
  - the idea that is expressed in a work of writing, art, etc.

Goal: word representations that capture the relationships between words

Meaning Representations in Computers

- Knowledge-based representation
- Corpus-based representation
  - ✓Atomic symbol
  - ✓Neighbors
    - High-dimensional sparse word vector
    - Low-dimensional dense word vector
      - Method 1 – dimension reduction
      - Method 2 – direct learning

Meaning Representations in Computers

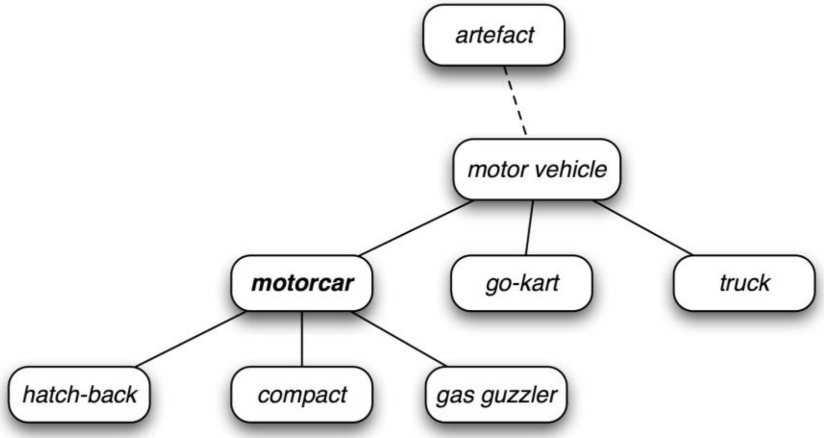
- Knowledge-based representation
- Corpus-based representation
  - ✓ Atomic symbol
  - ✓ Neighbors
    - High-dimensional sparse word vector
    - Low-dimensional dense word vector
      - Method 1 – dimension reduction
      - Method 2 – direct learning

Knowledge-Based Representation

- Hypernyms (is-a) relationships of WordNet

```
from nltk.corpus import wordnet as wn
panda = wn.synset('panda.n.01')
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```



Issues:

- newly-invented words
- subjective
- annotation effort
- difficult to compute word similarity

Meaning Representations in Computers

- Knowledge-based representation
- Corpus-based representation
  - ✓Atomic symbol
  - ✓Neighbors
    - High-dimensional sparse word vector
    - Low-dimensional dense word vector
      - Method 1 – dimension reduction
      - Method 2 – direct learning

Corpus-Based Representation

- Atomic symbols: ***one-hot*** representation

car $[0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ \ldots\ 0]$

car

Issues: difficult to compute the similarity (i.e. comparing “car” and “motorcycle”)

$\underset{\text{car}}{[0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ \ldots\ 0]}$ AND $\underset{\text{motorcycle}}{[0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ \ldots\ 0]} = 0$

Idea: words with similar meanings often have similar neighbors

Corpus-Based Representation

- Co-occurrence matrix
  - Neighbor definition: full document v.s. windows

- **full document**: word-document co-occurrence matrix gives general topics

→ "Latent Semantic Analysis", "Latent Dirichlet Allocation"

- **windows**: context window for each word

→ capture syntactic (e.g. POS) and sematic information

Meaning Representations in Computers

- Knowledge-based representation
- Corpus-based representation
  - ✓ Atomic symbol
  - ✓ Neighbors
    - High-dimensional sparse word vector
    - Low-dimensional dense word vector
      - Method 1 – dimension reduction
      - Method 2 – direct learning

Window-Based Co-occurrence Matrix

- Example
  - Window length=1
  - Left or right context
  - Corpus:

I love AI.
I love deep learning.
I enjoy learning.

similarity > 0

Counts	I	love	enjoy	AI	deep	learning
I	0	2	1	0	0	0
love	2	0	0	1	1	0
enjoy	1	0	0	0	0	1
AI	0	1	0	0	0	0
deep	0	1	0	0	0	1
learning	0	0	1	0	1	0

Issues:
- matrix size increases with vocabulary
- high dimensional
- sparsity

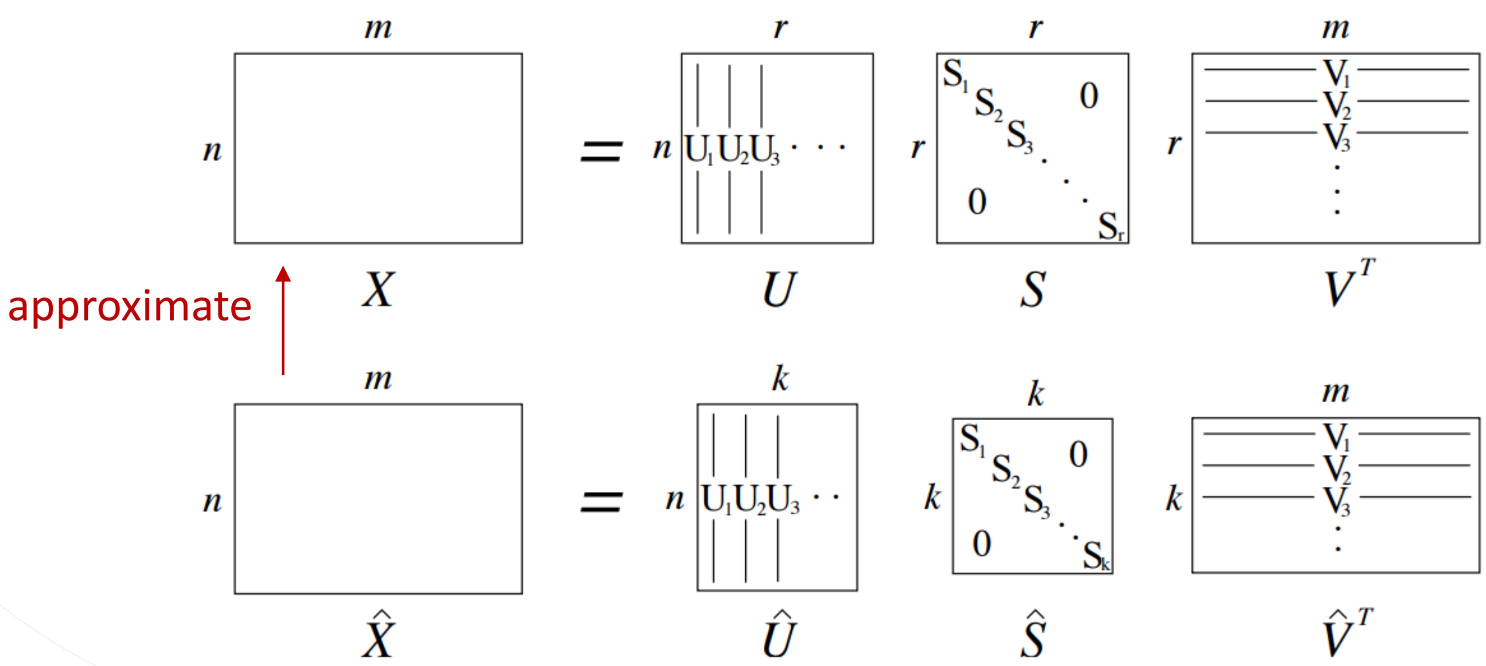
Idea: low-dimensional dense word vector

Meaning Representations in Computers

- Knowledge-based representation
- Corpus-based representation
  - ✓Atomic symbol
  - ✓Neighbors
    - High-dimensional sparse word vector
    - Low-dimensional dense word vector
      - Method 1 – dimension reduction
      - Method 2 – direct learning

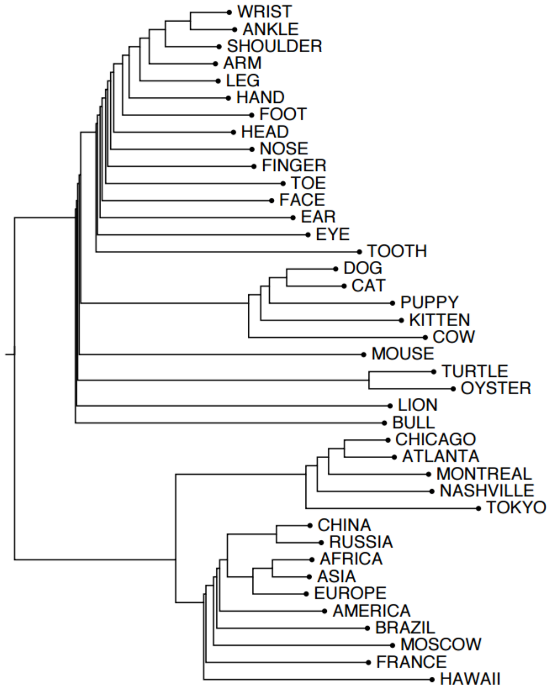
Low-Dimensional Dense Word Vector

- Method 1: dimension reduction on the matrix
- Singular Value Decomposition (SVD) of co-occurrence matrix X

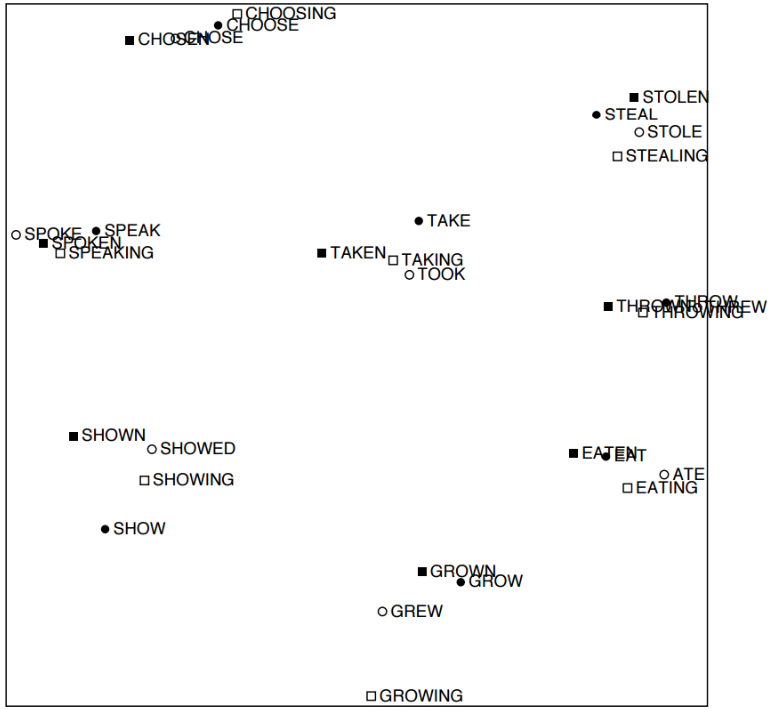


Low-Dimensional Dense Word Vector

- Method 1: dimension reduction on the matrix
- Singular Value Decomposition (SVD) of co-occurrence matrix X



semantic relations



syntactic relations

Issues:
computationally expensive
difficult to add new words

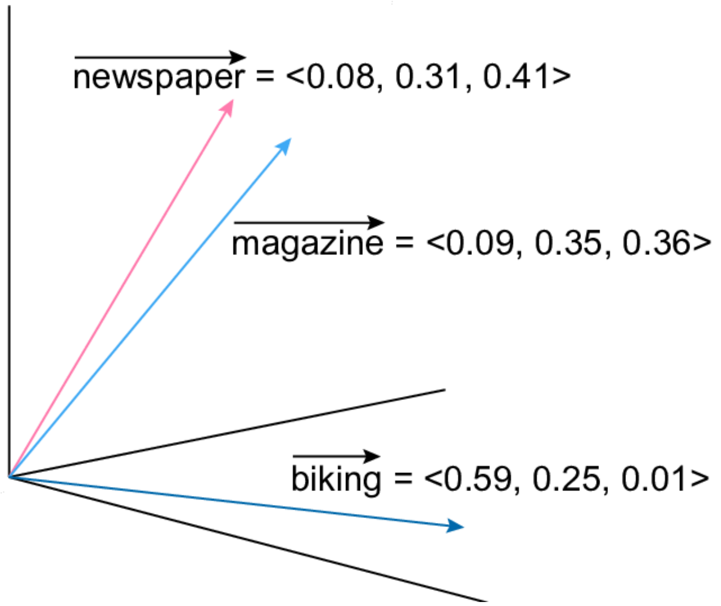
Idea: directly learn low-dimensional word vectors

Meaning Representations in Computers

- Knowledge-based representation
- Corpus-based representation
  - ✓Atomic symbol
  - ✓Neighbors
    - High-dimensional sparse word vector
    - Low-dimensional dense word vector
      - Method 1 – dimension reduction
      - Method 2 – direct learning

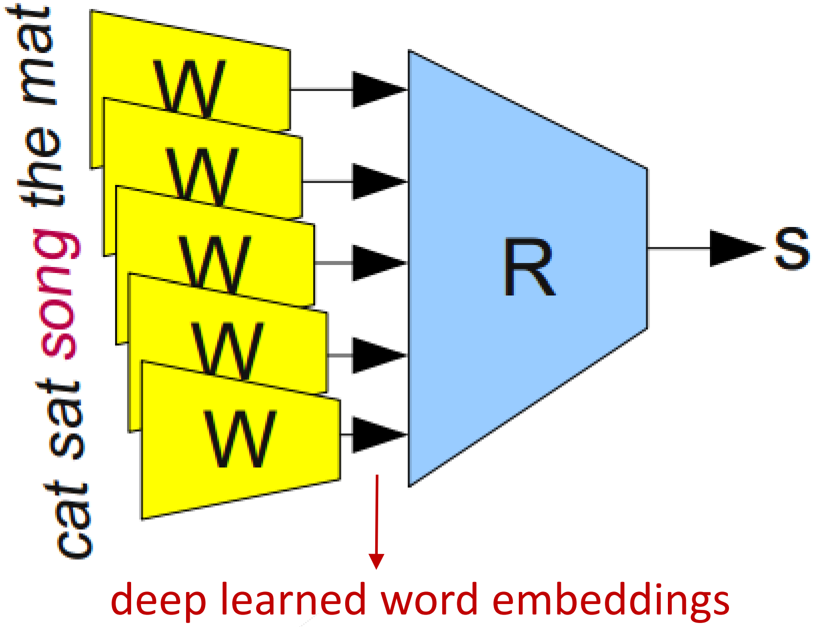
Low-Dimensional Dense Word Vector

- Method 2: directly learn low-dimensional word vectors
  - Learning representations by back-propagation. (Rumelhart et al., 1986)
  - A neural probabilistic language model (Bengio et al., 2003)
  - NLP (almost) from Scratch (Collobert & Weston, 2008)
  - Widely-used models: word2vec (Mikolov et al. 2013) and Glove (Pennington et al., 2014)
    - As known as "Word Embeddings"



Major Advantages of Word Embeddings

- Propagate ***any*** information into them via neural networks
  - form the basis for all language-related tasks



The networks, R and Ws, can be updated during model training

Concluding Remarks

- Knowledge-based representation
- Corpus-based representation
  - ✓Atomic symbol
  - ✓Neighbors
    - High-dimensional sparse word vector
    - Low-dimensional dense word vector
      - Method 1 – dimension reduction
      - Method 2 – direct learning

Sequence Modeling & Embeddings

Yun-Nung (Vivian) Chen
http://vivianchen.idv.tw

- Word Representation Basics
- **Word Embeddings**
- Recurrent Neural Network

NTU MIULAB

Word2Vec Skip-Gram

Mikolov et al., "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013.
Mikolov et al., "Efficient estimation of word representations in vector space," in *ICLR Workshop*, 2013.

Word2Vec – Skip-Gram Model

- Goal: predict surrounding words within a window of each word
- Objective function: maximize the probability of any context word given the current center word

$$w_1, w_2, \cdots, \underbrace{w_{t-m}, \cdots, w_{t-1}, \underset{w_I}{w_t}, \underset{w_O}{w_{t+1}}, \cdots, w_{t+m}}_{\text{context window}}, \cdots, w_{T-1}, w_T$$

$$p(w_{O,1}, w_{O,2}, \cdots, w_{O,C} \mid w_I) = \prod_{c=1}^{C} p(w_{O,c} \mid w_I)$$

target word vector

$$C(\theta) = -\sum_{w_I} \sum_{c=1}^{C} \log p(w_{O,c} \mid w_I)$$

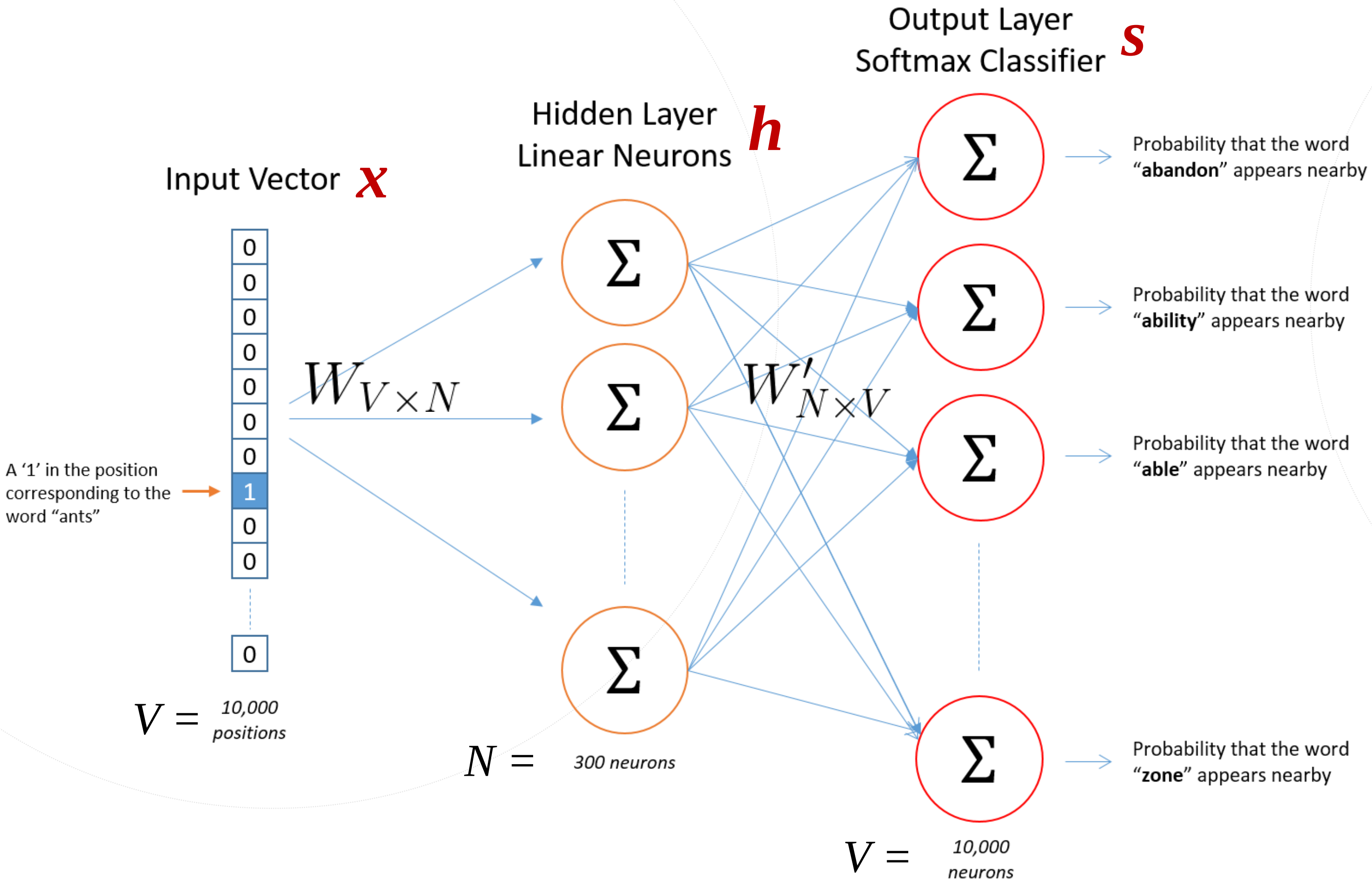
$$p(w_O \mid w_I) = \frac{\exp(v'^T_{w_O} v_{w_I})}{\sum_j \exp(v'^T_{w_j} v_{w_I})}$$

outside target word

Benefit: faster, easily incorporate a new sentence/document or add a word to vocab

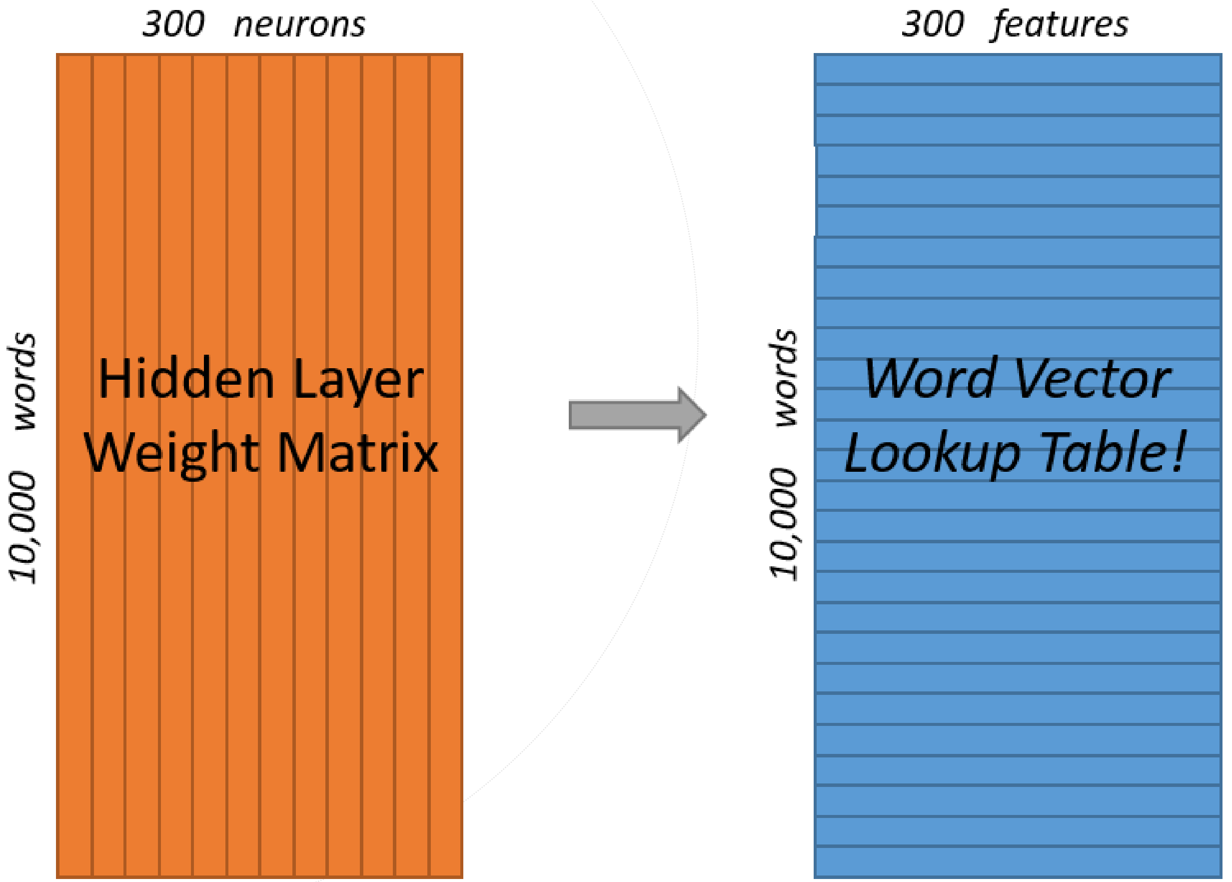
Word2Vec Skip-Gram Illustration

- Goal: predict surrounding words within a window of each word

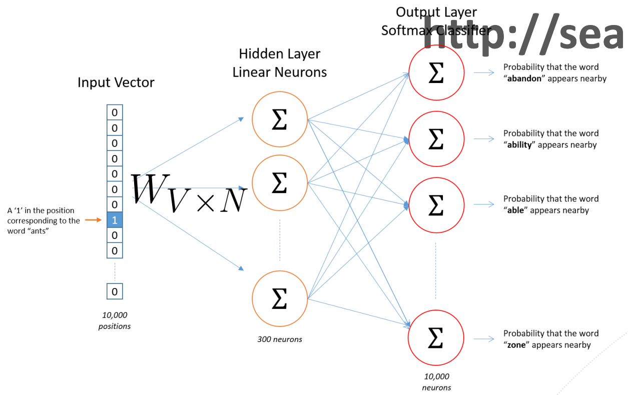


Hidden Layer Matrix → Word Embedding Matrix

$$W_{V \times N}$$



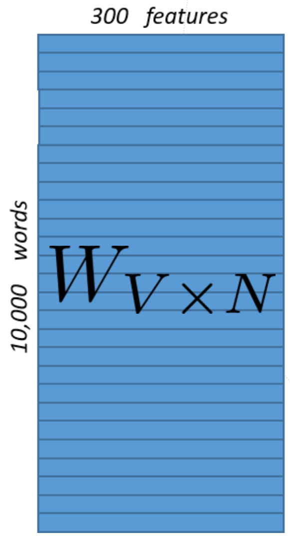
Weight Matrix Relation



- Hidden layer weight matrix = word vector lookup

$$h = x^T W = W_{(k,.)} := v_{w_I}$$

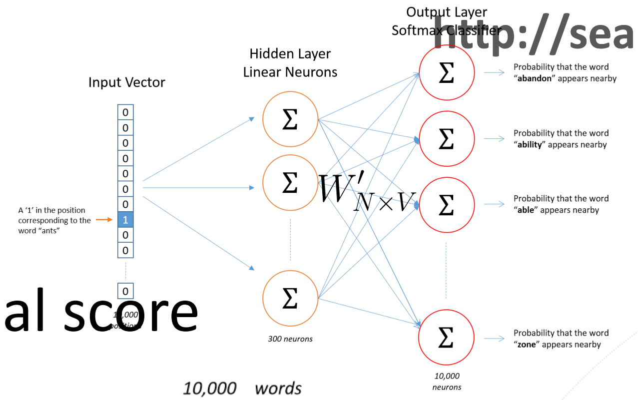
$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = \begin{bmatrix} 10 & 12 & 19 \end{bmatrix}$$

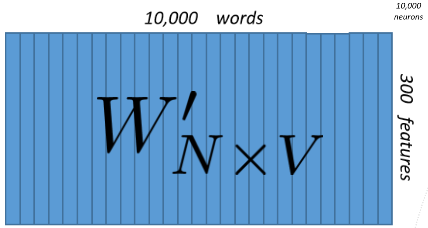


The 300-dim feature representation has the ability of predicting the contexts

Weight Matrix Relation

- Output layer weight matrix = weighted sum as final score



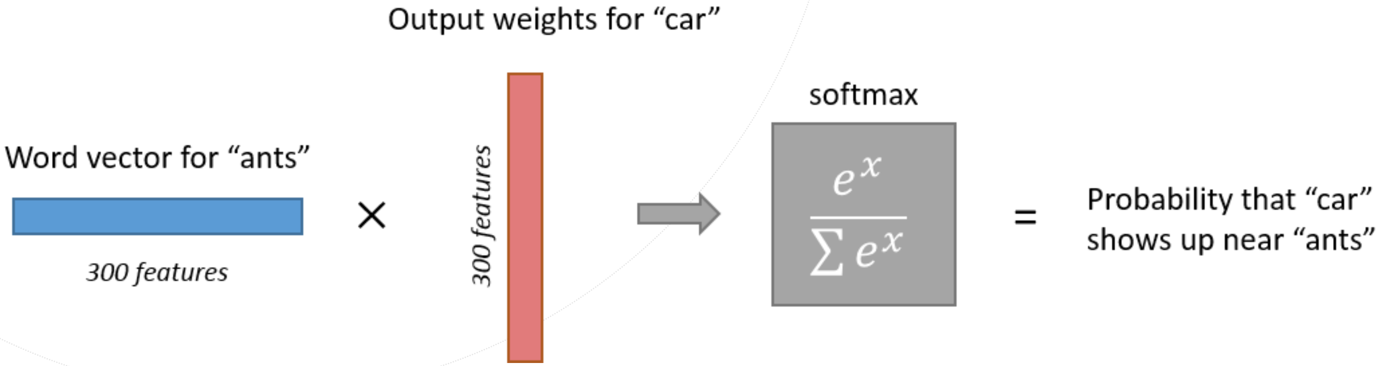


$$s_j = h v'_{w_j}$$

$$p(w_j = w_{O,c} \mid w_I) = y_{j_c} = \boxed{\frac{\exp(s_{j_c})}{\sum_{j'=1}^{V} \exp(s_{j'})}}$$

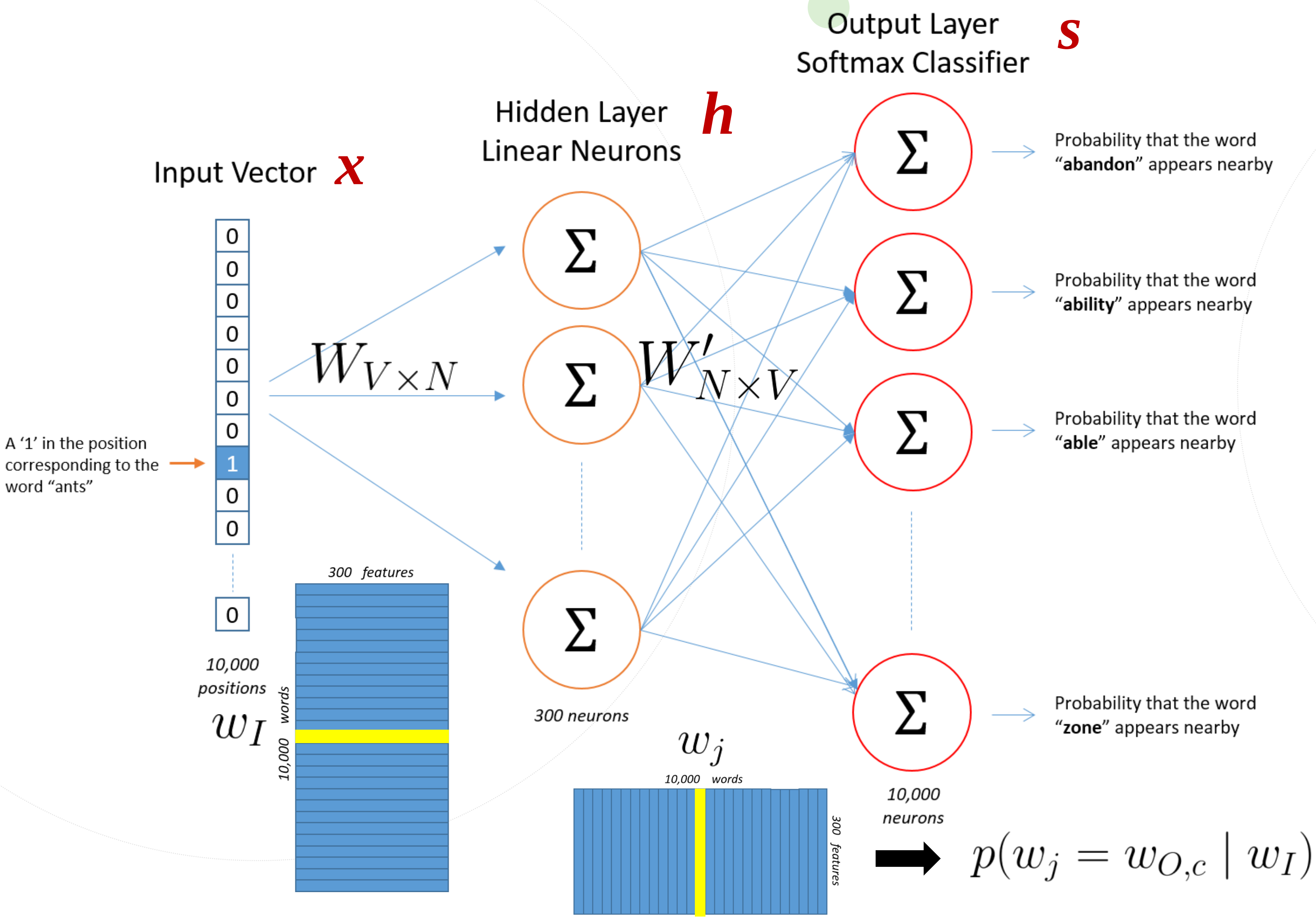
softmax

within the context window



Each vocabulary entry has two vectors: as a **target** word and as a **context** word

Word2Vec Skip-Gram Illustration

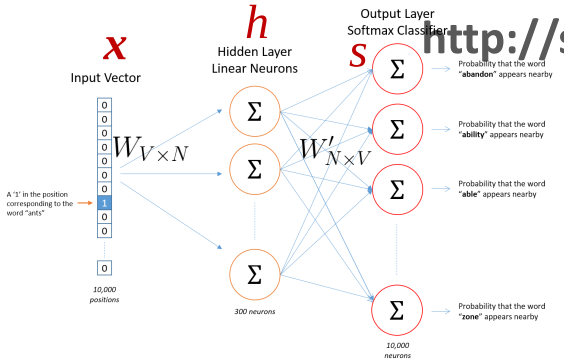


Loss Function

- Given a target word (w_I)

$$
\begin{aligned}
C(\theta) &= -\log p(w_{O,1}, w_{O,2}, \cdots, w_{O,C} \mid w_I) \\
&= -\log \prod_{c=1}^{C} \frac{\exp(s_{j_c})}{\sum_{j'=1}^{V} \exp(s_{j'})} \\
&= -\sum_{c=1}^{C} s_{j_c} + C \log \sum_{j'=1}^{V} \exp(s_{j'})
\end{aligned}
$$

SGD Update for W'



- Given a target word (w_I)

$$\frac{\partial C(\theta)}{\partial w'_{ij}} = \sum_{c=1}^{C} \frac{\partial C(\theta)}{\partial s_{j_c}} \frac{\partial s_{j_c}}{\partial w'_{ij}} = \sum_{c=1}^{C} (y_{j_c} - t_{j_c}) \cdot h_i$$

$$\boxed{s_j = {v'_{w_j}}^T \cdot h}$$

$$\frac{\partial C(\theta)}{\partial s_{j_c}} = y_{j_c} - t_{j_c} := e_{j_c}$$

error term

=1, when w_{jc} is within the context window

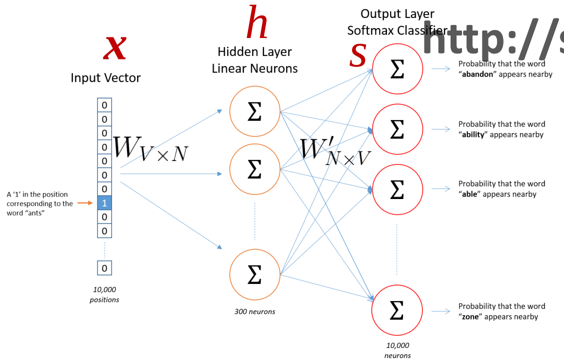
=0, otherwise

$$\boxed{{w'_{ij}}^{(t+1)} = {w'_{ij}}^{(t)} - \eta \cdot \sum_{c=1}^{C} (y_{j_c} - t_{j_c}) \cdot h_i}$$

NTU MIULAB

SGD Update for W

http://seamls.miulab.tw/



$$\frac{\partial C(\theta)}{\partial w_{ki}} = \frac{\partial C(\theta)}{\partial h_i}\frac{\partial h_i}{\partial w_{ki}} = \sum_{j=1}^{V}\sum_{c=1}^{C}(y_{jc} - t_{jc}) \cdot w'_{ij} \cdot x_k$$

$$\boxed{h = x^T W}$$

$$\frac{\partial C(\theta)}{\partial h_i} = \sum_{j=1}^{V}\frac{\partial C(\theta)}{\partial s_j}\frac{\partial s_j}{\partial h_i} = \sum_{j=1}^{V}\sum_{c=1}^{C}(y_{jc} - t_{jc}) \cdot w'_{ij}$$

$$\boxed{s_j = {v'_{w_j}}^T \cdot h}$$

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \eta \cdot \sum_{j=1}^{V}\sum_{c=1}^{C}(y_{jc} - t_{jc}) \cdot w'_{ij} \cdot x_j$$

SGD Update

$$w'^{(t+1)}_{ij} = w'^{(t)}_{ij} - \eta \cdot \sum_{c=1}^{C} (y_{j_c} - t_{j_c}) \cdot h_i$$

$$EI_j = \sum_{c=1}^{C} (y_{j_c} - t_{j_c})$$

$$v'_{w_j}{}^{(t+1)} = v'_{w_j}{}^{(t)} - \eta \cdot EI_j \cdot h$$

$$w^{(t+1)}_{ij} = w^{(t)}_{ij} - \eta \cdot \sum_{j=1}^{V} \sum_{c=1}^{C} (y_{j_c} - t_{j_c}) \cdot w'_{ij} \cdot x_j$$

$$EH_i = \sum_{j=1}^{V} EI_j \cdot w'_{ij} \cdot x_j$$

$$v^{(t+1)}_{w_I} = v^{(t)}_{w_I} - \eta \cdot EH^T$$

large vocabularies or large training corpora → expensive computations

limit the number of output vectors that must be updated per training instance

Negative Sampling

- Idea: only update a sample of output vectors

$$C(\theta) = -\log \sigma({v'_{w_O}}^T v_{w_I}) + \sum_{w_j \in \mathcal{W}_{\text{neg}}} \log \sigma({v'_{w_j}}^T v_{w_I})$$

$$\boxed{{v'_{w_j}}^{(t+1)} = {v'_{w_j}}^{(t)} - \eta \cdot EI_j \cdot h}$$

$$\boxed{EI_j = \sigma({v'_{w_j}}^T v_{w_I}) - t_j}$$

$$\boxed{v_{w_I}^{(t+1)} = v_{w_I}^{(t)} - \eta \cdot EH^T}$$

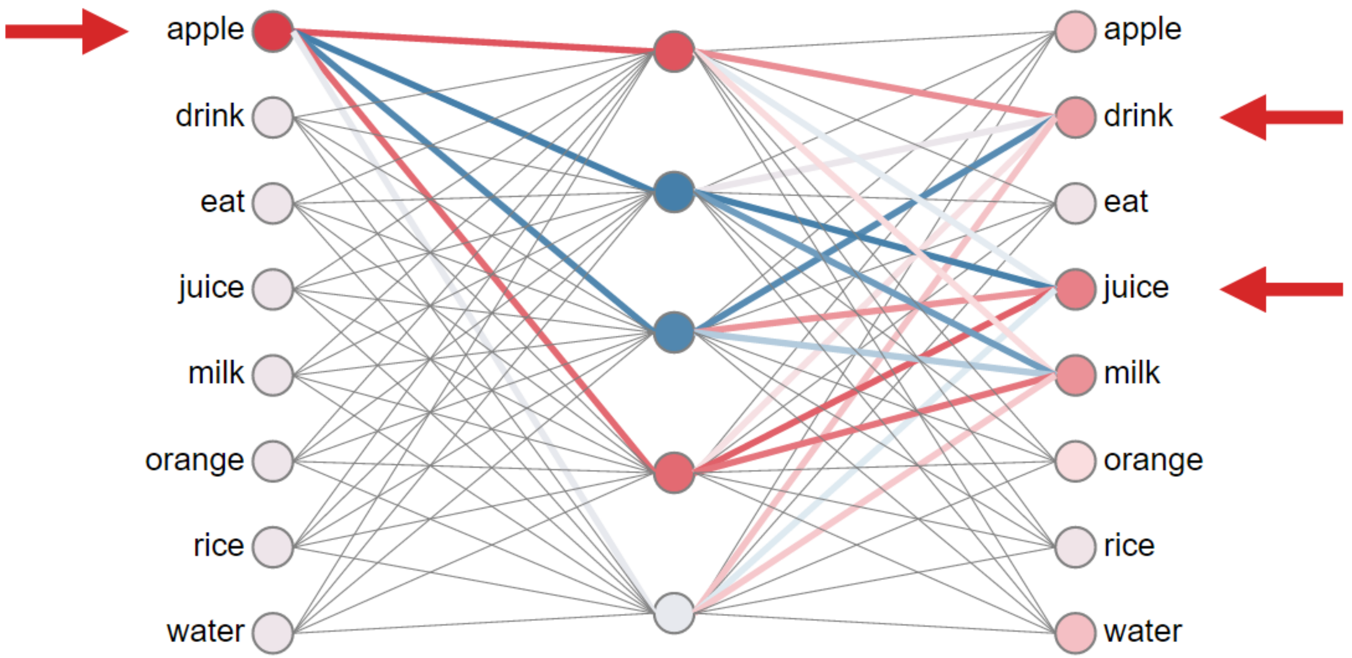
$$\boxed{EH = \sum_{w_j \in \{w_O\} \cup \mathcal{W}_{\text{neg}}} EI_j \cdot v'_{w_j}}$$

$$w_j \in \{w_O\} \cup \mathcal{W}_{\text{neg}}$$

Mikolov et al., "Distributed representations of words and phrases and their compositionality," in NIPS, 2013.

Word2Vec Skip-Gram Visualization https://ronxin.github.io/wevi/

- Skip-gram training data:
  apple|drink^juice,orange|eat^apple,rice|drink^juice,juice|drink^milk,milk|drink^rice,water|drink^milk,juice|orange^apple,juice|apple^drink,milk|rice^drink,drink|milk^water,drink|water^juice,drink|juice^water



Word2Vec Variants

- **Skip-gram**: predicting surrounding words given the target word (Mikolov+, 2013) better

$$p(w_{t-m}, \cdots w_{t-1}, w_{t+1}, \cdots , w_{t+m} \mid w_t)$$

- **CBOW (continuous bag-of-words)**: predicting the target word given the surrounding words (Mikolov+, 2013)

$$p(w_t \mid w_{t-m}, \cdots w_{t-1}, w_{t+1}, \cdots , w_{t+m})$$

- **LM (Language modeling)**: predicting the next words given the proceeding contexts (Mikolov+, 2013) first

$$p(w_{t+1} \mid w_t)$$

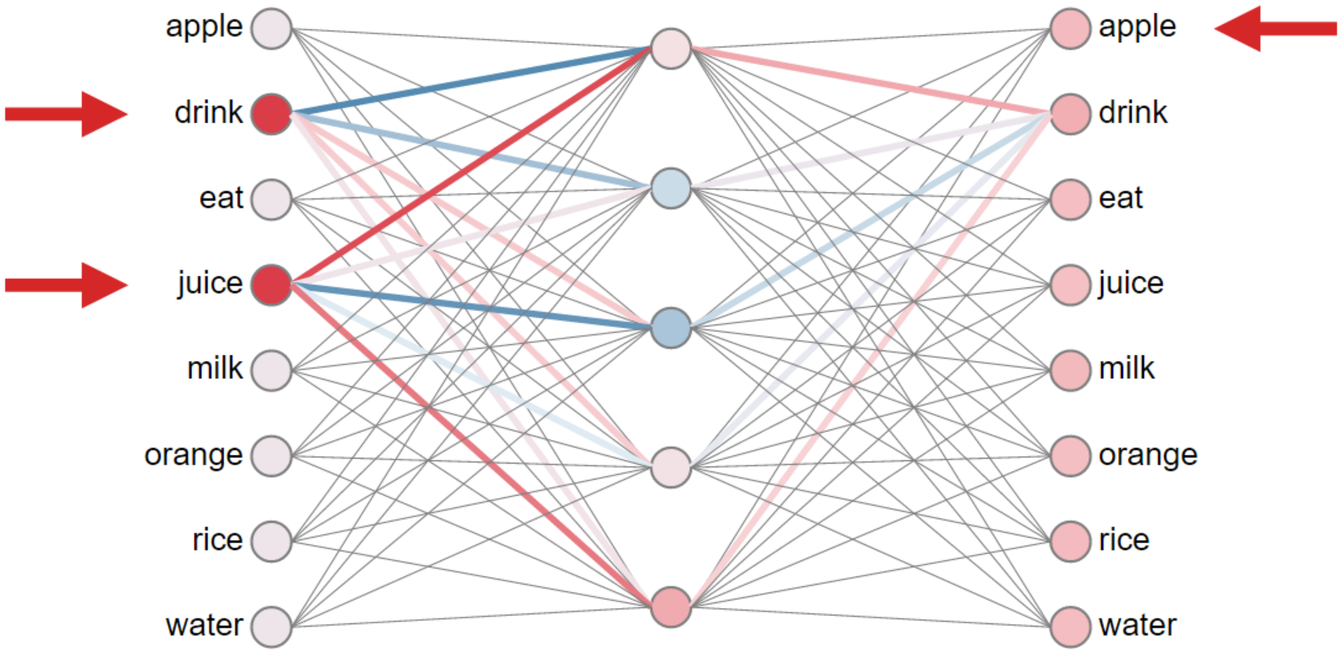
Practice the derivation by yourself!!

Mikolov et al., "Efficient estimation of word representations in vector space," in *ICLR Workshop*, 2013.
Mikolov et al., "Linguistic regularities in continuous space word representations," in *NAACL HLT*, 2013.

Word2Vec CBOW

- Goal: predicting the target word given the surrounding words

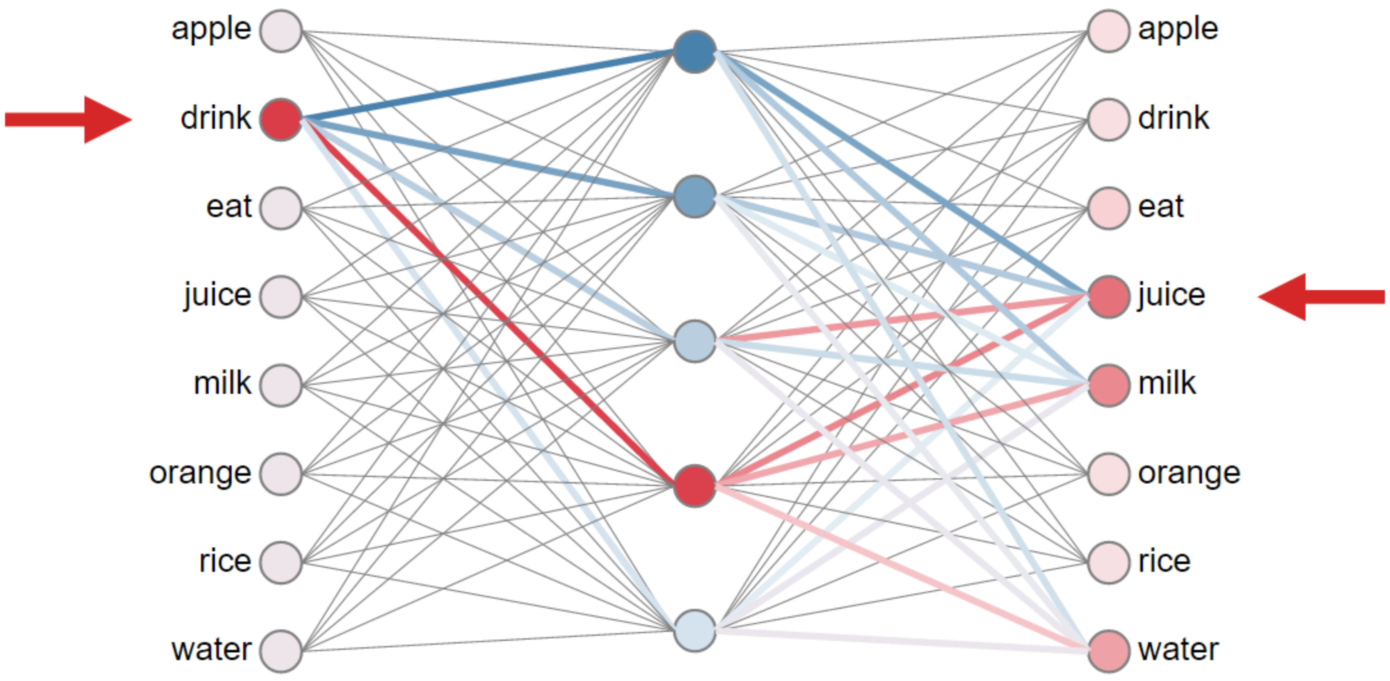
$$p(w_t \mid w_{t-m}, \cdots w_{t-1}, w_{t+1}, \cdots , w_{t+m})$$



Word2Vec LM

- Goal: predicting the next words given the proceeding contexts

$$p(w_{t+1} \mid w_t)$$



Comparison

- Count-based
  - Example
    - LSA, HAL (Lund & Burgess), COALS (Rohde et al), Hellinger-PCA (Lebret & Collobert)
  - Pros
    - ✓ Fast training
    - ✓ Efficient usage of statistics
  - Cons
    - ✓ Primarily used to capture word similarity
    - ✓ Disproportionate importance given to large counts

- Direct prediction
  - Example
    - NNLM, HLBL, RNN, Skipgram/CBOW, (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton; Mikolov et al; Mnih & Kavukcuoglu)
  - Pros
    - ✓ Generate improved performance on other tasks
    - ✓ Capture complex patterns beyond word similarity
  - Cons
    - ✓ Benefits mainly from large corpus
    - ✓ Inefficient usage of statistics

Combining the benefits from both worlds → GloVe

NTU MIULAB

GloVe

Pennington et al., ”GloVe: Global Vectors for Word Representation,” in EMNLP, 2014.

GloVe

- Idea: ratio of co-occurrence probability can encode meaning
- P_{ij} is the probability that word w_j appears in the context of word w_i

$$P_{ij} = P(w_j \mid w_i) = X_{ij}/X_i$$

- Relationship between the words w_i and w_j

	x = solid	x = gas	x = water	x = random
$P(x \mid \text{ice})$	large	small	large	small
$P(x \mid \text{stream})$	small	large	large	small
$\frac{P(x \mid \text{ice})}{P(x \mid \text{stream})}$	large	small	~ 1	~ 1

GloVe

- The relationship of w_i and w_j approximates the ratio of their co-occurrence probabilities with various w_k

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

$$F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

$$F((v_{w_i} - v_{w_j})^T v'_{\tilde{w}_k}) = \frac{P_{ik}}{P_{jk}} \qquad F(\cdot) = \exp(\cdot)$$

$$v_{w_i} \cdot v'_{\tilde{w}_k} = v_{w_i}^T v'_{\tilde{w}_k} = \log P(w_k \mid w_i)$$

GloVe

$$v_{w_i} \cdot v'_{\tilde{w}_j} = v_{w_i}^T v'_{\tilde{w}_j} = \log P(w_j \mid w_i)$$
$$= \log P_{ij} = \log(X_{ij}) - \log(X_i)$$

$$\boxed{P_{ij} = X_{ij}/X_i}$$

$$v_{w_i}^T v'_{\tilde{w}_j} + b_i + \tilde{b}_j = \log(X_{ij})$$

$$C(\theta) = \sum_{i,j=1}^{V} f(P_{ij})(v_{w_i} \cdot v'_{\tilde{w}_j} - \log P_{ij})^2$$

$$C(\theta) = \sum_{i,j=1}^{V} f(X_{ij})(v_{w_i}^T v'_{\tilde{w}_j} + b_i + \tilde{b}_j - \log X_{ij})^2$$

fast training, scalable, good performance even with small corpus, and small vectors

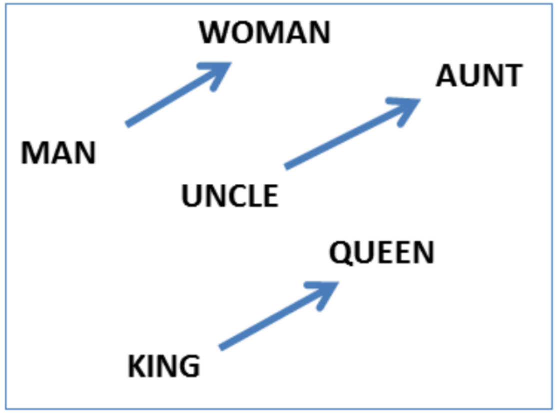
Word Vector Evaluation

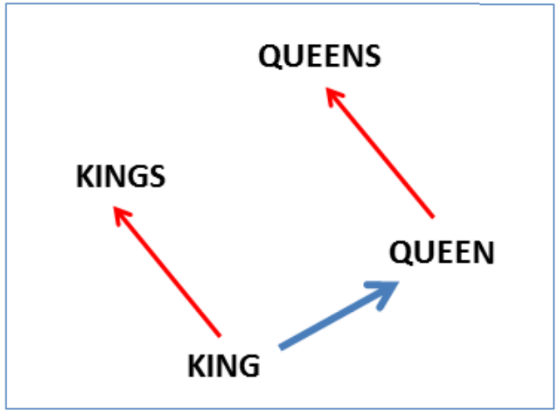
Intrinsic Evaluation – Word Analogies

- Word linear relationship $w_A : w_B = w_C : w_x$

$$x = \arg\max_x \frac{(v_{w_B} - v_{w_A} + v_{w_C})^T v_{w_x}}{\left\| v_{w_B} - v_{w_A} + v_{w_C} \right\|}$$

- **Syntactic** and **Semantic** example questions [link]





Issue: what if the information is there but not linear

Intrinsic Evaluation – Word Analogies

- Word linear relationship $w_A : w_B = w_C : w_x$
- Syntactic and **Semantic** example questions [link]

city---in---state
Chicago : Illinois = Houston : Texas
Chicago : Illinois = Philadelphia : Pennsylvania
Chicago : Illinois = Phoenix : Arizona
Chicago : Illinois = Dallas : Texas
Chicago : Illinois = Jacksonville : Florida
Chicago : Illinois = Indianapolis : Indiana
Chicago : Illinois = Aus8n : Texas
Chicago : Illinois = Detroit : Michigan
Chicago : Illinois = Memphis : Tennessee
Chicago : Illinois = Boston : Massachusetts

Issue: different cities may have same name

capital---country
Abuja : Nigeria = Accra : Ghana
Abuja : Nigeria = Algiers : Algeria
Abuja : Nigeria = Amman : Jordan
Abuja : Nigeria = Ankara : Turkey
Abuja : Nigeria = Antananarivo : Madagascar
Abuja : Nigeria = Apia : Samoa
Abuja : Nigeria = Ashgabat : Turkmenistan
Abuja : Nigeria = Asmara : Eritrea
Abuja : Nigeria = Astana : Kazakhstan

Issue: can change with time

Intrinsic Evaluation – Word Analogies

- Word linear relationship $w_A : w_B = w_C : w_x$
- **Syntactic** and Semantic example questions [link]

superlative

bad : worst = big : biggest
bad : worst = bright : brightest
bad : worst = cold : coldest
bad : worst = cool : coolest
bad : worst = dark : darkest
bad : worst = easy : easiest
bad : worst = fast : fastest
bad : worst = good : best
bad : worst = great : greatest

past tense

dancing : danced = decreasing : decreased
dancing : danced = describing : described
dancing : danced = enhancing : enhanced
dancing : danced = falling : fell
dancing : danced = feeding : fed
dancing : danced = flying : flew
dancing : danced = generating : generated
dancing : danced = going : went
dancing : danced = hiding : hid
dancing : danced = hiding : hit

Intrinsic Evaluation – Word Correlation

- Comparing word correlation with human-judged scores
- Human-judged word correlation [link]

Word 1	Word 2	Human-Judged Score
tiger	cat	7.35
tiger	tiger	10.00
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62

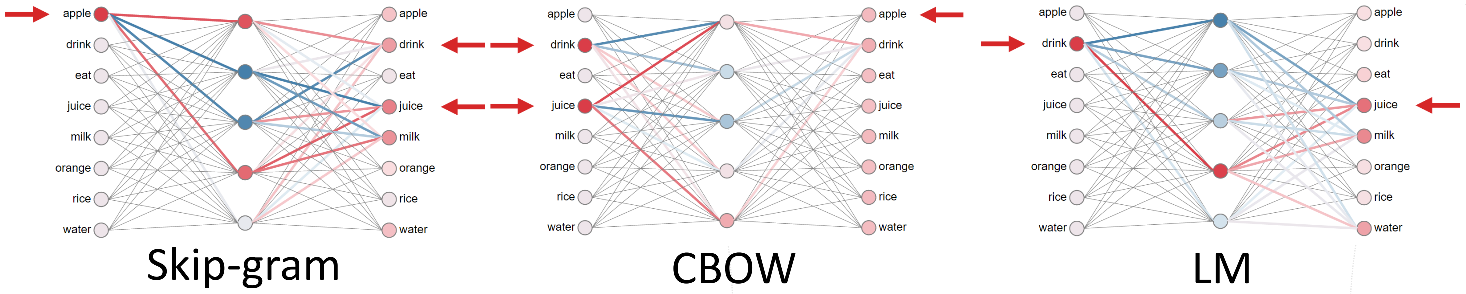
Ambiguity: synonym or same word with different POSs

Extrinsic Evaluation – Subsequent Task

- Goal: use word vectors in neural net models built for subsequent tasks
- Benefit
  - Ability to also classify words accurately
    - Ex. countries cluster together a classifying location words should be possible with word vectors
  - Incorporate any information into them other tasks
    - Ex. project sentiment into words to find most positive/negative words in corpus

Concluding Remarks

- Low dimensional word vector
  - word2vec



  - GloVe: combining count-based and direct learning
- Word vector evaluation
  - Intrinsic: word analogy, word correlation
  - Extrinsic: subsequent task

Sequence Modeling & Embeddings

Yun-Nung (Vivian) Chen
http://vivianchen.idv.tw

國立臺灣大學
National Taiwan University

- Word Representation Basics
- Word Embeddings
- Recurrent Neural Network

Outline

- Language Modeling
  - N-gram Language Model
  - Feed-Forward Neural Language Model
  - Recurrent Neural Network Language Model (RNNLM)
- Recurrent Neural Network
  - Definition
  - Training via Backpropagation through Time (BPTT)
  - Training Issue
- Applications
  - Sequential Input
  - Sequential Output
    - Aligned Sequential Pairs (Tagging)
    - Unaligned Sequential Pairs (Seq2Seq/Encoder-Decoder)

Outline

- **Language Modeling**
  - N-gram Language Model
  - Feed-Forward Neural Language Model
  - Recurrent Neural Network Language Model (RNNLM)
- Recurrent Neural Network
  - Definition
  - Training via Backpropagation through Time (BPTT)
  - Training Issue
- Applications
  - Sequential Input
  - Sequential Output
    - Aligned Sequential Pairs (Tagging)
    - Unaligned Sequential Pairs (Seq2Seq/Encoder-Decoder)

Language Modeling

- Goal: estimate the probability of a word sequence

$$P(w_1, \cdots, w_m)$$

- Example task
  - determinate whether a sequence is grammatical or makes more sense

recognize speech
or
wreck a nice beach

If P(recognize speech)
> P(wreck a nice beach)

Output =
“recognize speech”

Outline

- Language Modeling
  - **N-gram Language Model**
  - Feed-Forward Neural Language Model
  - Recurrent Neural Network Language Model (RNNLM)
- Recurrent Neural Network
  - Definition
  - Training via Backpropagation through Time (BPTT)
  - Training Issue
- Applications
  - Sequential Input
  - Sequential Output
    - Aligned Sequential Pairs (Tagging)
    - Unaligned Sequential Pairs (Seq2Seq/Encoder-Decoder)

N-Gram Language Modeling

- Goal: estimate the probability of a word sequence

$$P(w_1, \cdots, w_m)$$

- N-gram language model
  - Probability is conditioned on a window of $(n\text{-}1)$ previous words

$$P(w_1, \cdots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \cdots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-(n-1)}, \cdots, w_{i-1})$$

  - Estimate the probability based on the training data

$$P(\text{beach}|\text{nice}) = \frac{C(nice\ beach)}{C(nice)}$$

Count of "nice beach" in the training data

Count of "nice" in the training data

Issue: some sequences may not appear in the training data

N-Gram Language Modeling

- Training data:
  - The dog ran ……
  - The cat jumped ……

P(jumped | dog) = ~~0~~ 0.0001

P(ran | cat) = ~~0~~ 0.0001

give some small probability
→ smoothing

➢ The probability is not accurate.

➢ The phenomenon happens because we cannot collect all the possible text in the world as training data.

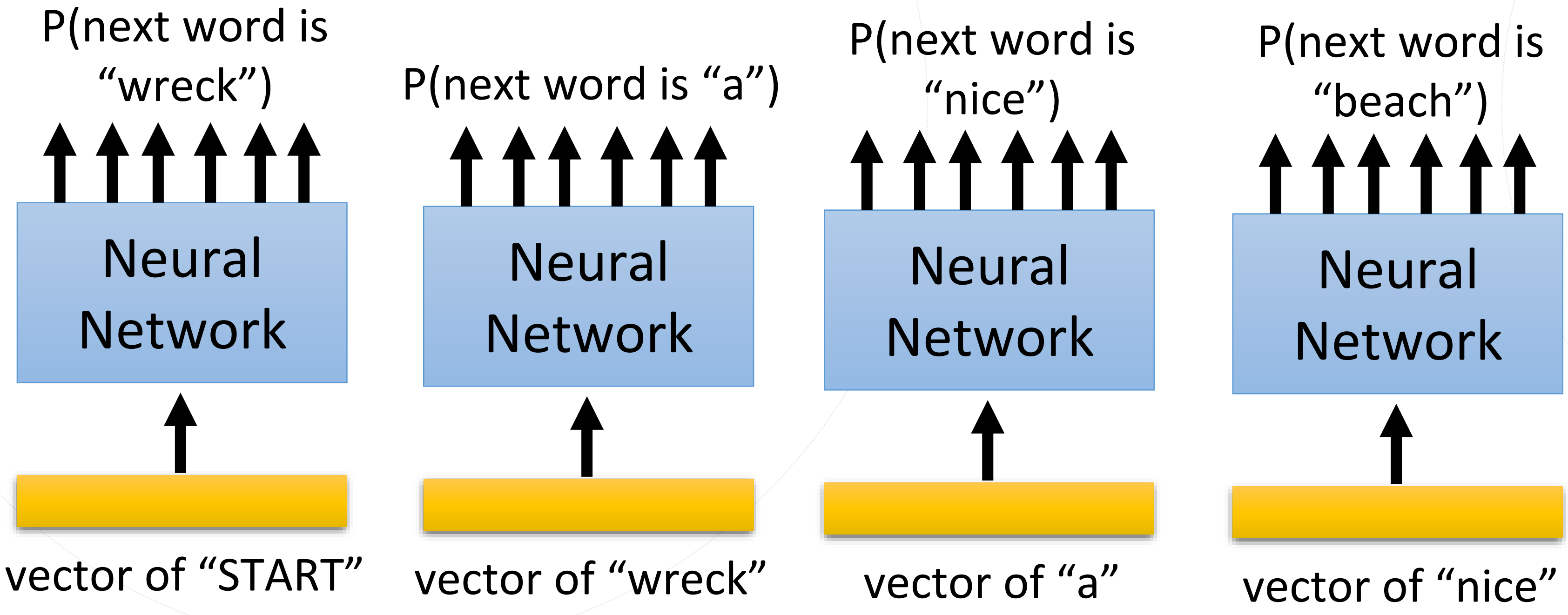
Outline

- Language Modeling
  - N-gram Language Model
  - **Feed-Forward Neural Language Model**
  - Recurrent Neural Network Language Model (RNNLM)
- Recurrent Neural Network
  - Definition
  - Training via Backpropagation through Time (BPTT)
  - Training Issue
- Applications
  - Sequential Input
  - Sequential Output
    - Aligned Sequential Pairs (Tagging)
    - Unaligned Sequential Pairs (Seq2Seq/Encoder-Decoder)

Neural Language Modeling

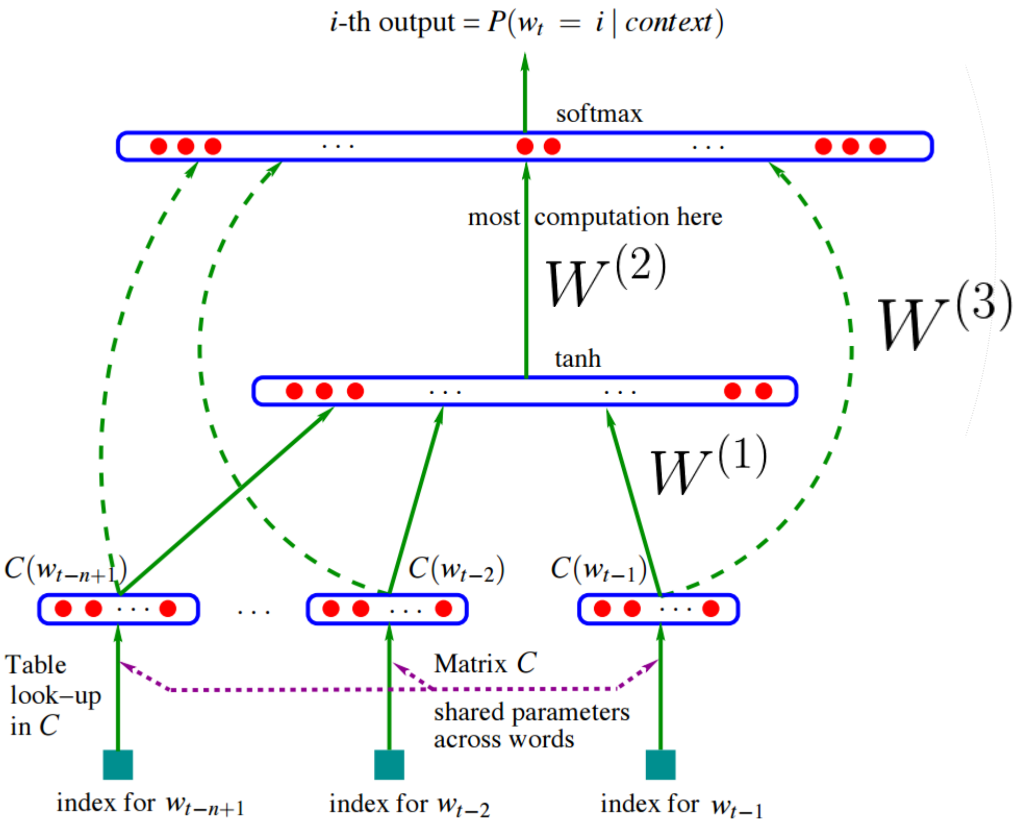
- Idea: estimate $P(w_i \mid w_{i-(n-1)}, \cdots, w_{i-1})$ not from count, but from the NN prediction

P("wreck a nice beach") = P(wreck|START)P(a|wreck)P(nice|a)P(beach|nice)

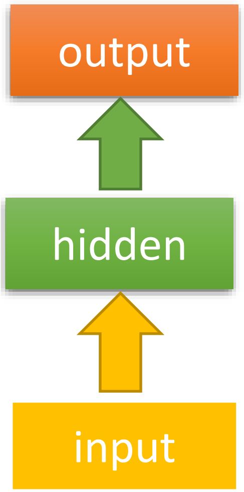


Neural Language Modeling

$$\hat{y} = \text{softmax}(W^{(2)}\sigma(W^{(1)}x + b^{(1)}) + W^{(3)}x + b^{(3)})$$



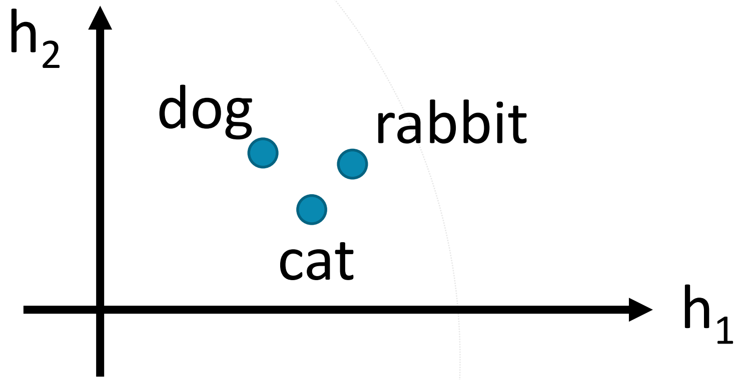
Probability distribution of the next word



context vector

Neural Language Modeling

- The input layer (or hidden layer) of the related words are close



  - If P(jump|dog) is large, P(jump|cat) increase accordingly (even there is not "… cat jump …" in the data)

Smoothing is automatically done

Issue: fixed context window for conditioning

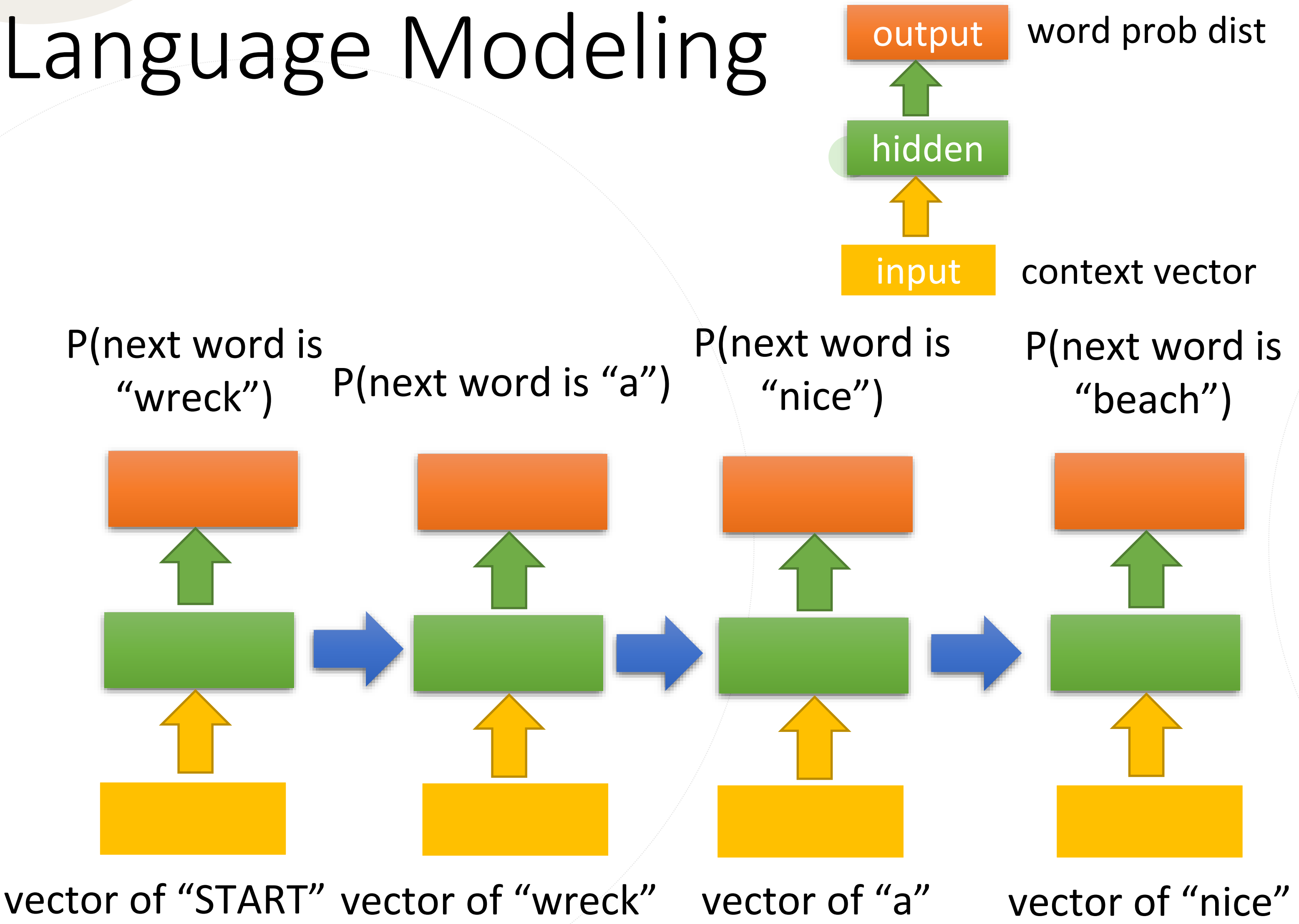
Outline

- Language Modeling
  - N-gram Language Model
  - Feed-Forward Neural Language Model
  - **Recurrent Neural Network Language Model (RNNLM)**
- Recurrent Neural Network
  - Definition
  - Training via Backpropagation through Time (BPTT)
  - Training Issue
- Applications
  - Sequential Input
  - Sequential Output
    - Aligned Sequential Pairs (Tagging)
    - Unaligned Sequential Pairs (Seq2Seq/Encoder-Decoder)

Recurrent Neural Network

- Idea: condition the neural network on <u>all previous words</u> and <u>tie the weights</u> at each time step
- Assumption: temporal information matters

RNN Language Modeling



Idea: pass the information from the previous hidden layer to leverage all contexts

Outline

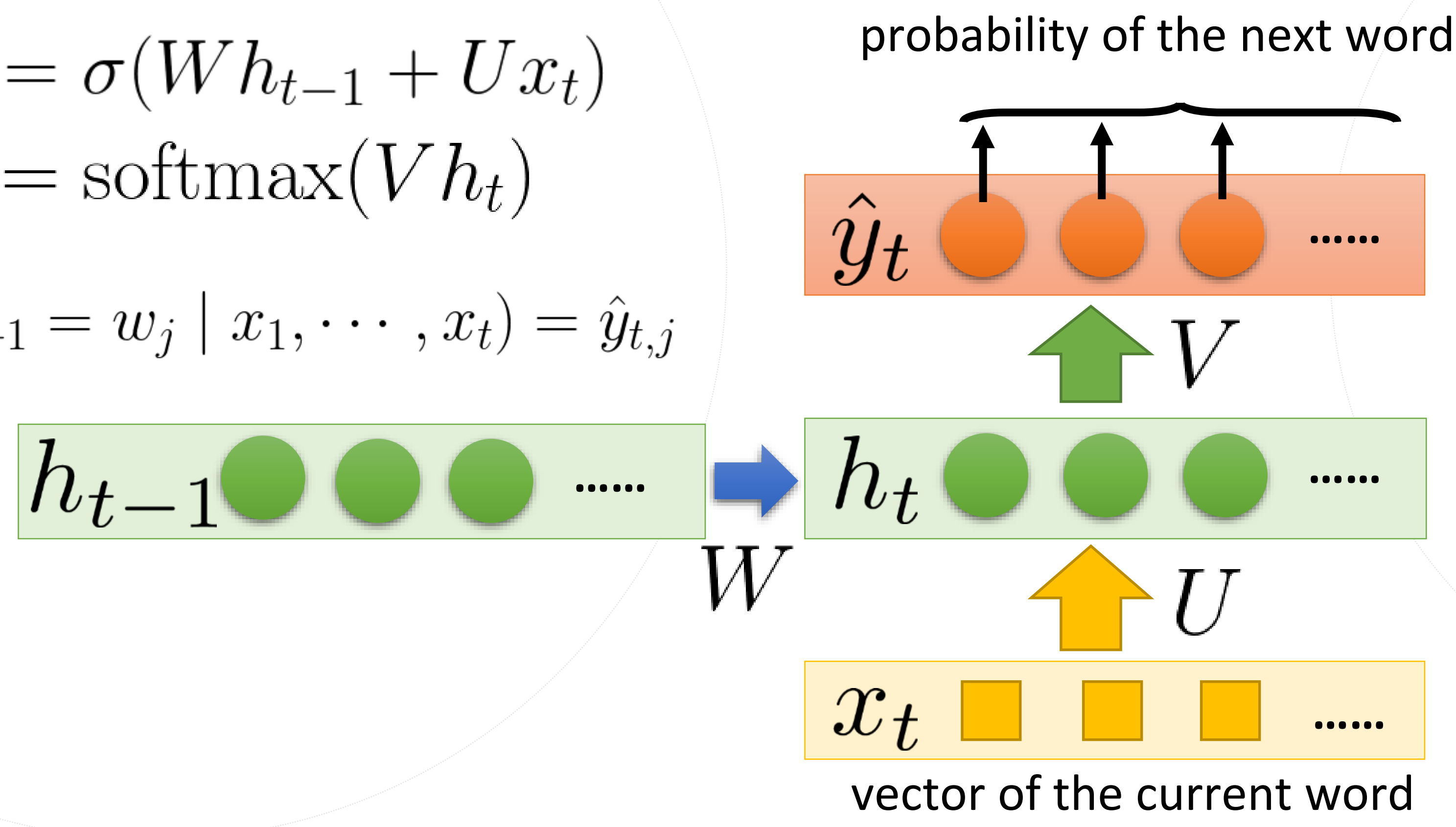
- Language Modeling
  - N-gram Language Model
  - Feed-Forward Neural Language Model
  - Recurrent Neural Network Language Model (RNNLM)
- **Recurrent Neural Network**
  - Definition
  - Training via Backpropagation through Time (BPTT)
  - Training Issue
- Applications
  - Sequential Input
  - Sequential Output
    - Aligned Sequential Pairs (Tagging)
    - Unaligned Sequential Pairs (Seq2Seq/Encoder-Decoder)

RNNLM Formulation

- At each time step,

$$h_t = \sigma(W h_{t-1} + U x_t)$$
$$\hat{y}_t = \mathrm{softmax}(V h_t)$$

$$P(x_{t+1} = w_j \mid x_1, \cdots, x_t) = \hat{y}_{t,j}$$



Outline

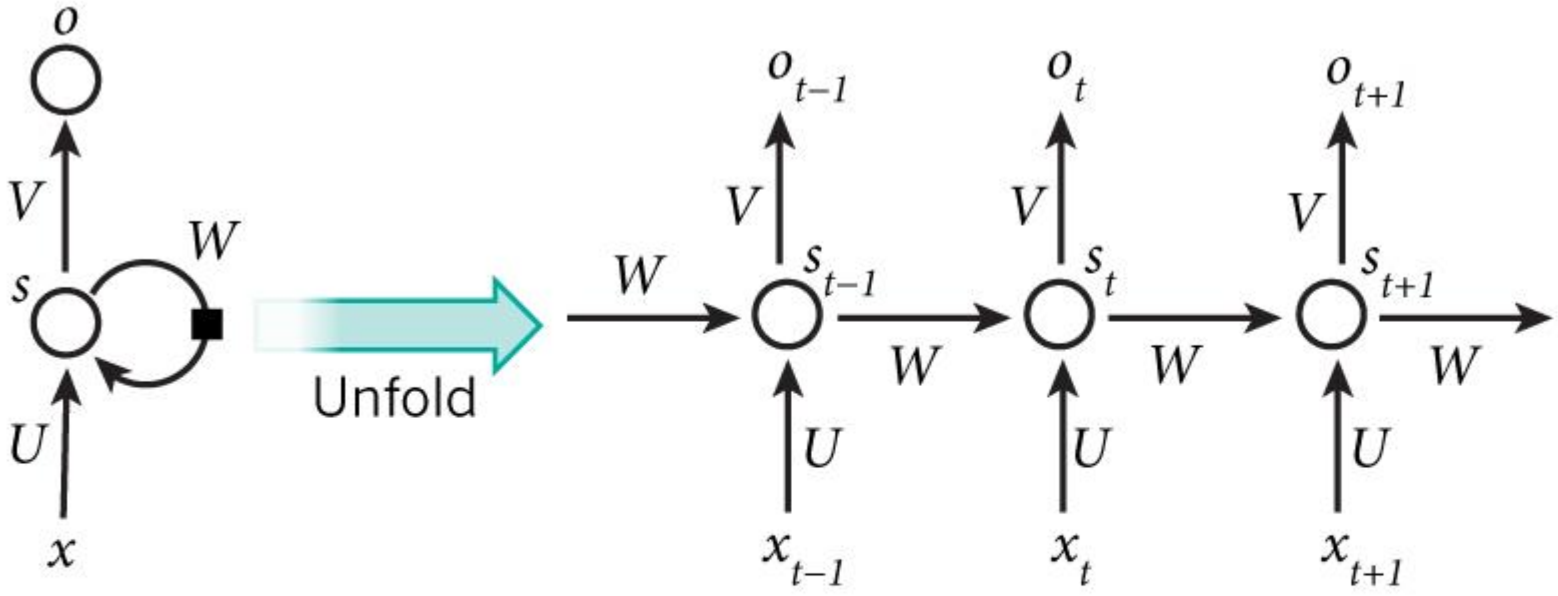
- Language Modeling
  - N-gram Language Model
  - Feed-Forward Neural Language Model
  - Recurrent Neural Network Language Model (RNNLM)
- Recurrent Neural Network
  - **Definition**
  - Training via Backpropagation through Time (BPTT)
  - Training Issue
- Applications
  - Sequential Input
  - Sequential Output
    - Aligned Sequential Pairs (Tagging)
    - Unaligned Sequential Pairs (Seq2Seq/Encoder-Decoder)

Recurrent Neural Network Definition

$$s_t = \sigma(W s_{t-1} + U x_t)$$

$$o_t = \mathrm{softmax}(V s_t)$$

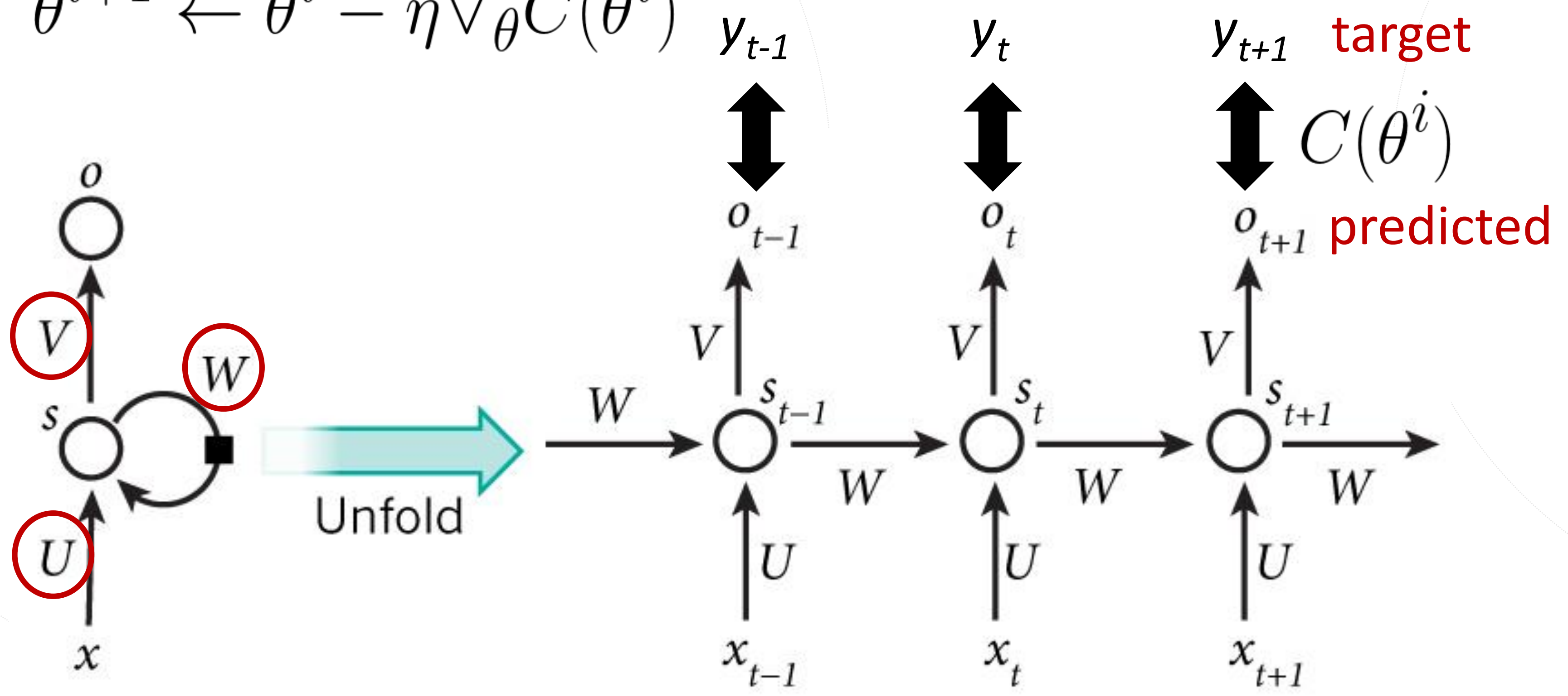
$\sigma(\cdot)$: tanh, ReLU



Model Training

- All model parameters $\theta = \{U, V, W\}$ can be updated by

$$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_\theta C(\theta^i)$$

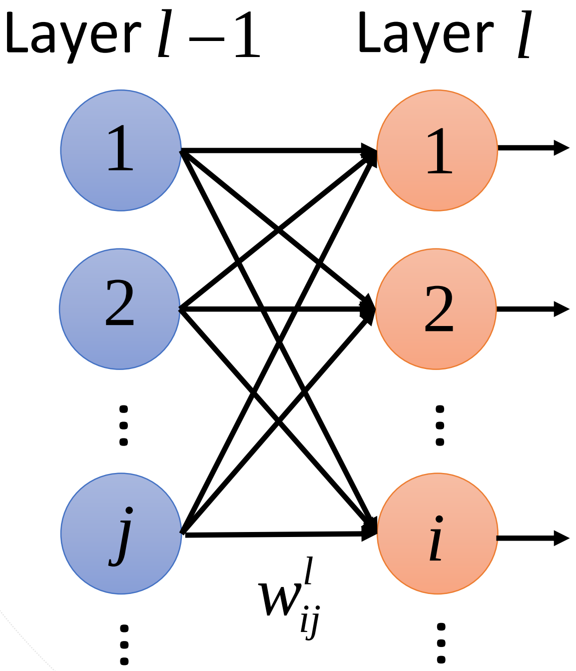


Outline

- Language Modeling
  - N-gram Language Model
  - Feed-Forward Neural Language Model
  - Recurrent Neural Network Language Model (RNNLM)
- Recurrent Neural Network
  - Definition
  - **Training via Backpropagation through Time (BPTT)**
  - Training Issue
- Applications
  - Sequential Input
  - Sequential Output
    - Aligned Sequential Pairs (Tagging)
    - Unaligned Sequential Pairs (Seq2Seq/Encoder-Decoder)

Backpropagation

$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \boxed{\frac{\partial C(\theta)}{\partial z_i^l}}\boxed{\frac{\partial z_i^l}{\partial w_{ij}^l}}$$



$\boxed{\delta_i^l}$ Error signal

$$\begin{cases} a_j^{l-1} & l>1 \\ x_j & l=1 \end{cases}$$

Backward Pass

$$\delta^L = \sigma'(z^L) \odot \nabla C(y)$$
$$\delta^{L-1} = \sigma'(z^{L-1}) \odot (W^L)^T \delta^L$$
$$\vdots$$
$$\delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}$$
$$\vdots$$

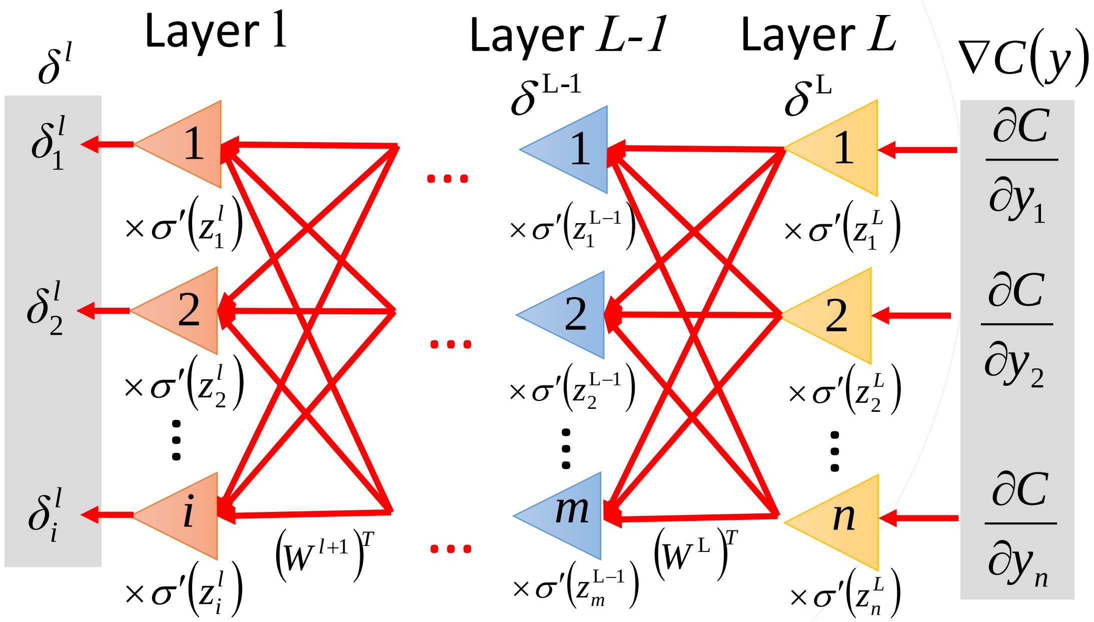
Forward Pass

$$z^1 = W^1 x + b^1$$
$$a^1 = \sigma(z^1)$$
$$\vdots$$
$$z^l = W^l a^{l-1} + b^l$$
$$a^l = \sigma(z^l)$$
$$\vdots$$

Backpropagation

$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \boxed{\frac{\partial C(\theta)}{\partial z_i^l}} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

δ_i^l Error signal



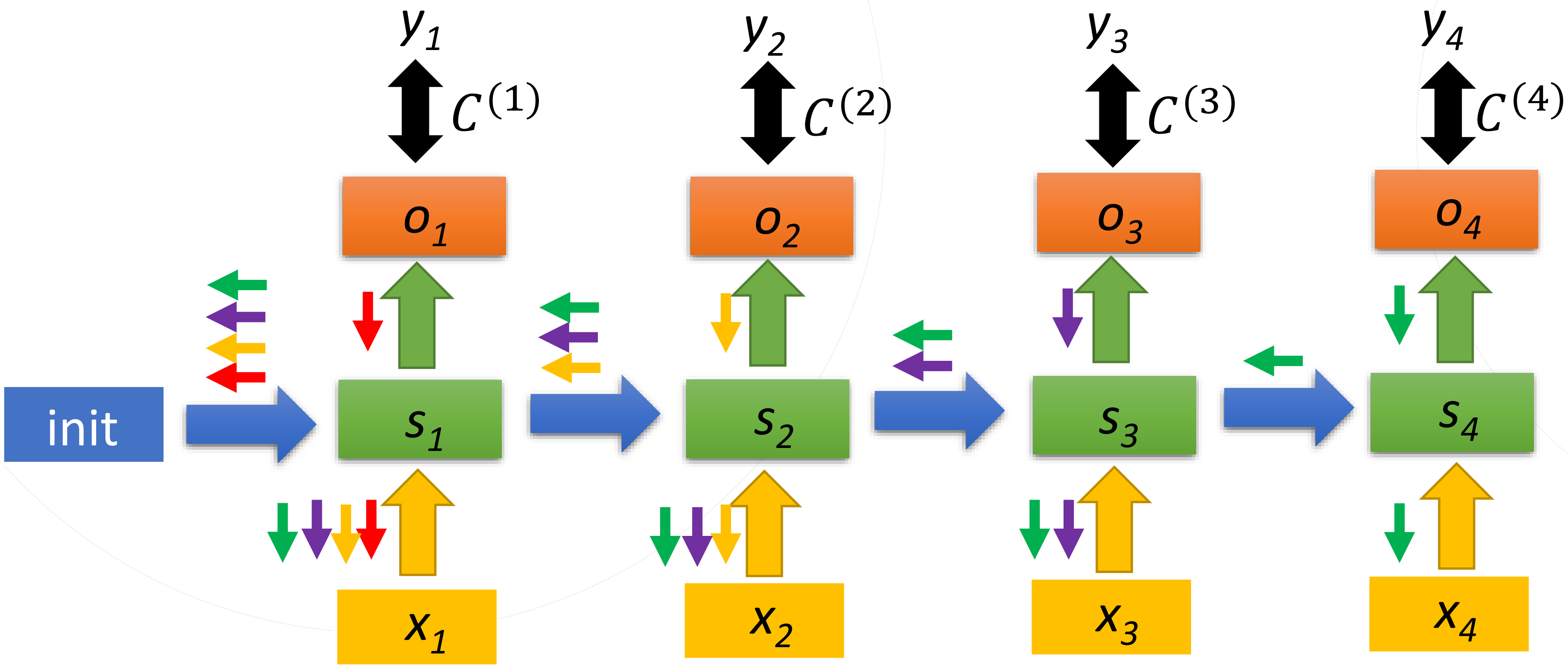
Backward Pass

$$\delta^L = \sigma'(z^L) \odot \nabla C(y)$$
$$\delta^{L-1} = \sigma'(z^{L-1}) \odot (W^L)^T \delta^L$$
$$\vdots$$
$$\delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}$$
$$\vdots$$

Backpropagation through Time (BPTT)

Forward Pass: Compute s_1, s_2, s_3, s_4 ……

Backward Pass: For $C^{(4)}$ For $C^{(3)}$ For $C^{(2)}$ For $C^{(1)}$



Outline

- Language Modeling
  - N-gram Language Model
  - Feed-Forward Neural Language Model
  - Recurrent Neural Network Language Model (RNNLM)
- Recurrent Neural Network
  - Definition
  - Training via Backpropagation through Time (BPTT)
  - **Training Issue**
- Applications
  - Sequential Input
  - Sequential Output
    - Aligned Sequential Pairs (Tagging)
    - Unaligned Sequential Pairs (Seq2Seq/Encoder-Decoder)

RNN Training Issue

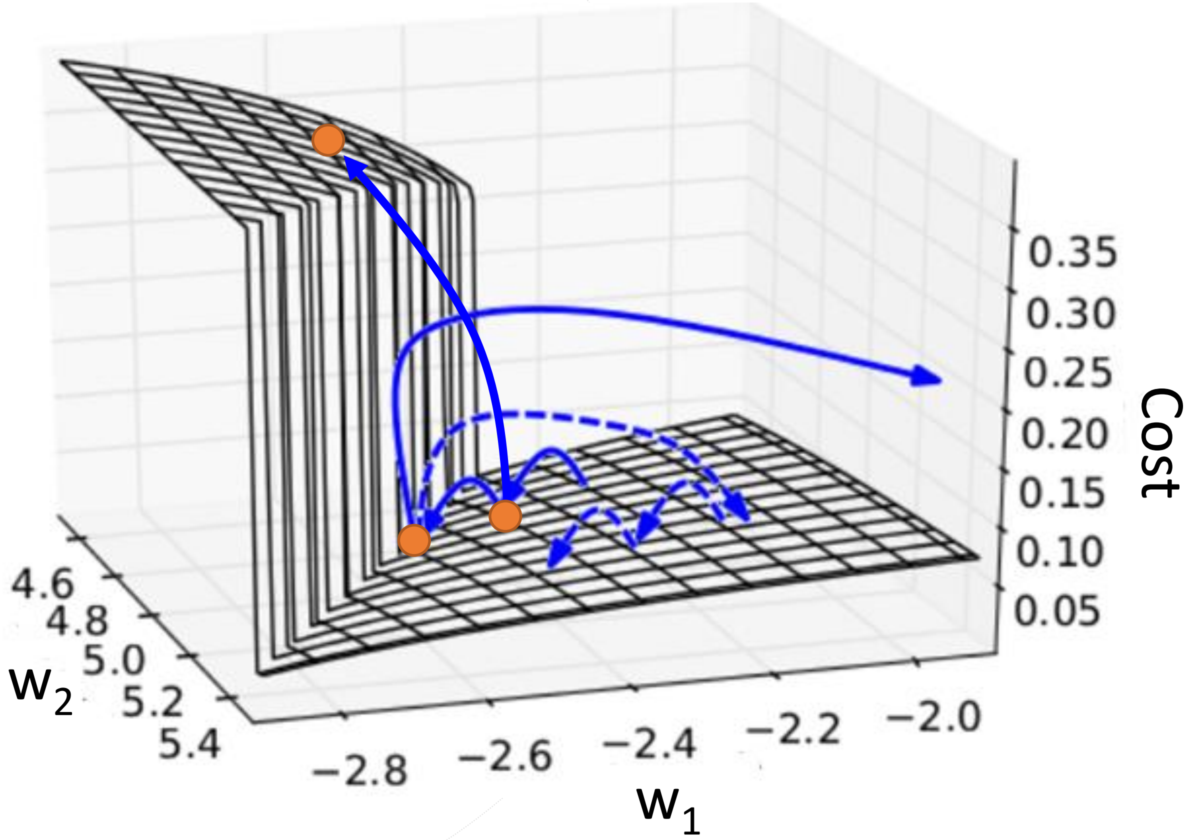
- The gradient is a product of Jacobian matrices, each associated with a step in the forward computation
- Multiply the same matrix at each time step during backprop

$$\delta^l = \sigma'(z^l) \odot \boxed{(W^{l+1})^T} \delta^{l+1}$$

The gradient becomes very small or very large quickly
→ **vanishing or exploding gradient**

Bengio et al., "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. of Neural Networks*, 1994. [link]
Pascanu et al., "On the difficulty of training recurrent neural networks," in *ICML*, 2013. [link]

Rough Error Surface



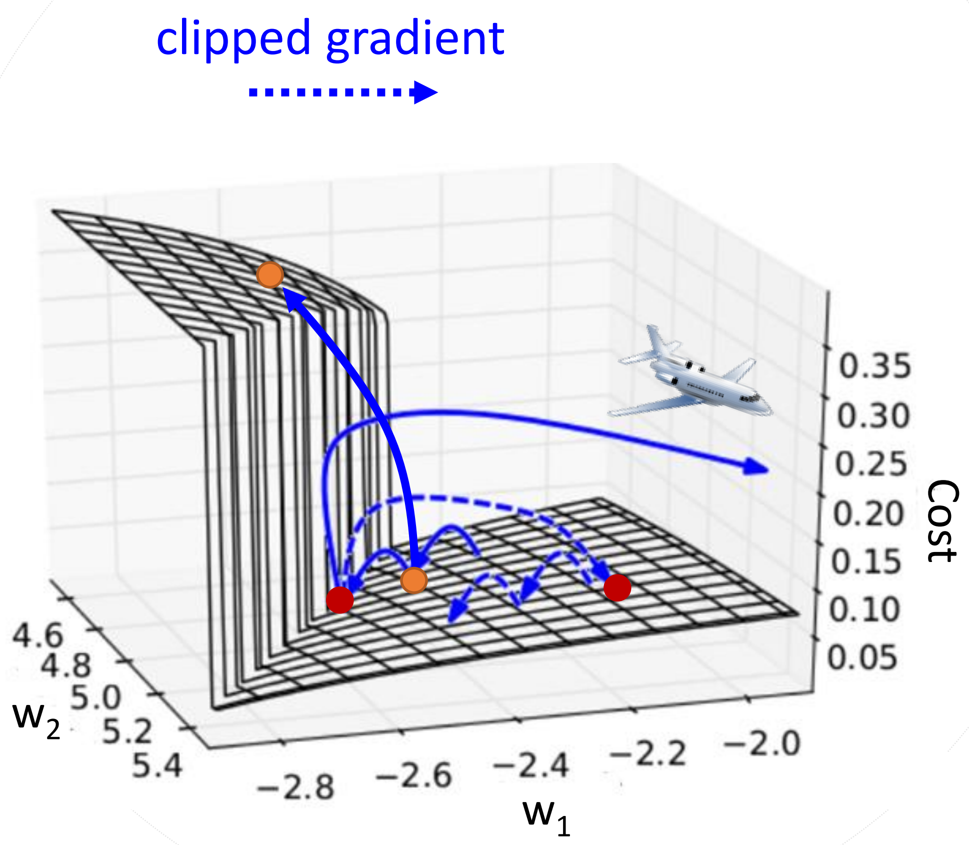
The error surface is either very flat or very steep

Bengio et al., "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. of Neural Networks*, 1994. [link]
Pascanu et al., "On the difficulty of training recurrent neural networks," in *ICML*, 2013. [link]

Possible Solutions

Recurrent Neural Network

Exploding Gradient: Clipping



Idea: control the gradient value to avoid exploding

Algorithm 1 Pseudo-code for norm clipping

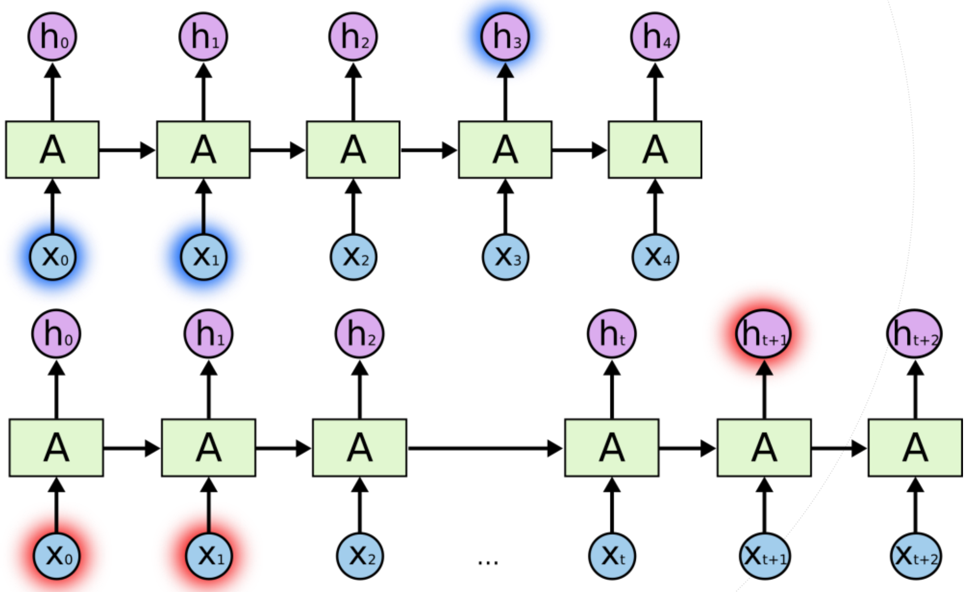
$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

Parameter setting: values from half to ten times the average can still yield convergence

Pascanu et al., "On the difficulty of training recurrent neural networks," in *ICML*, 2013. [link]

Vanishing Gradient: Gating Mechanism

- RNN models temporal sequence information
  - can handle “long-term dependencies” *in theory*



“I grew up in France…
I speak fluent French.”

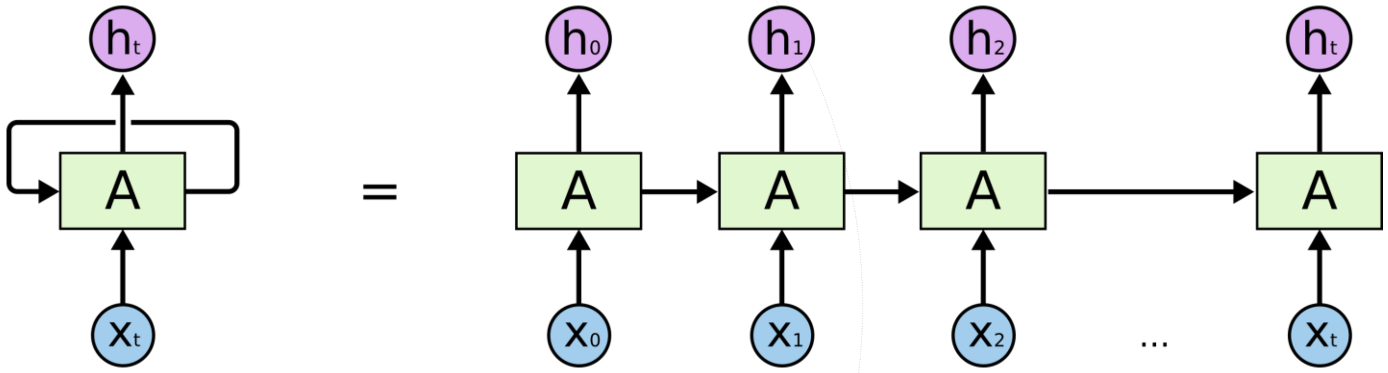
Issue: RNN cannot handle such “long-term dependencies” in practice due to vanishing gradient
→ apply the gating mechanism to directly encode the long-distance information

Long Short-Term Memory

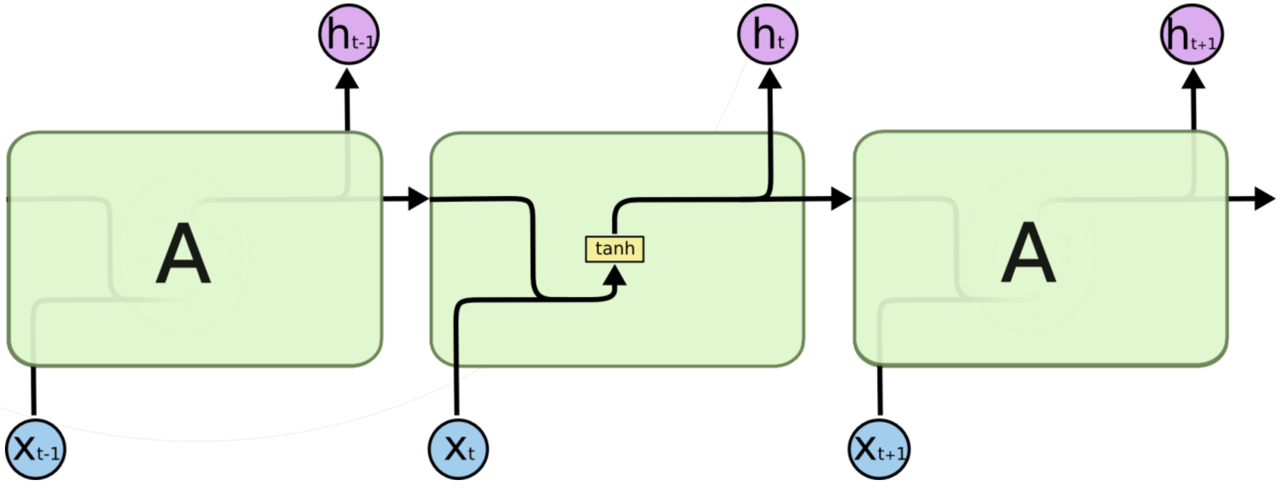
Addressing Vanishing Gradient Problem

Long Short-Term Memory (LSTM)

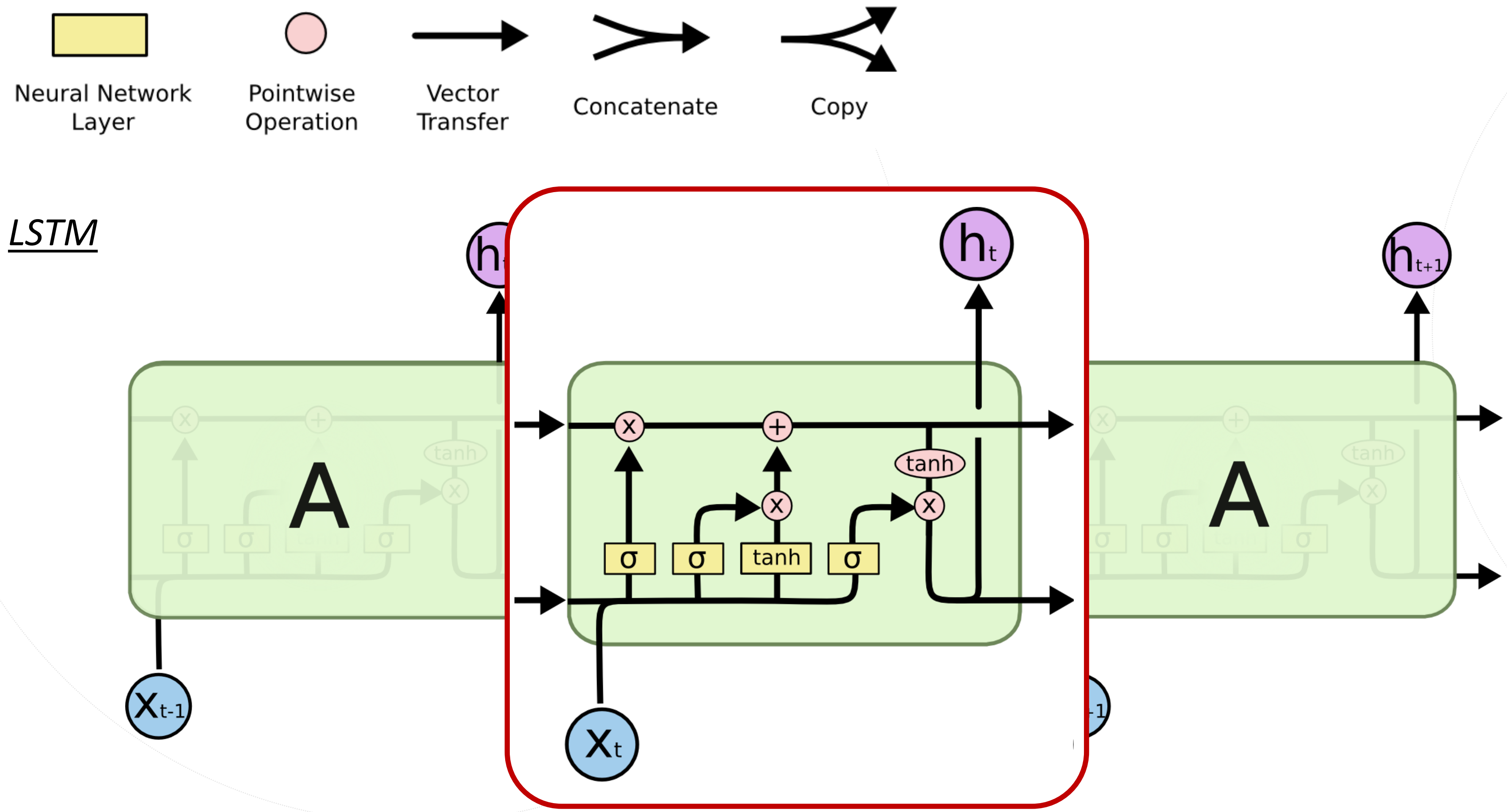
- LSTMs are explicitly designed to avoid the long-term dependency problem



Vanilla RNN

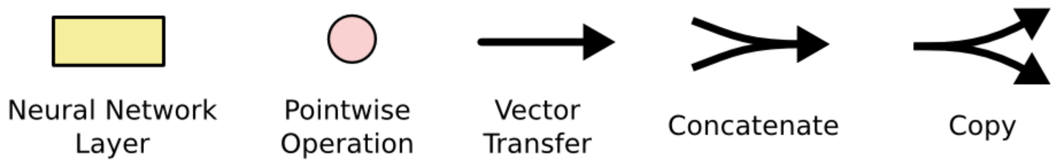


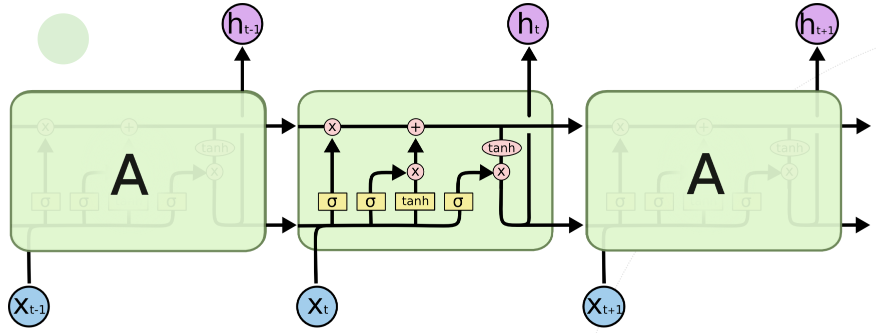
Long Short-Term Memory (LSTM)

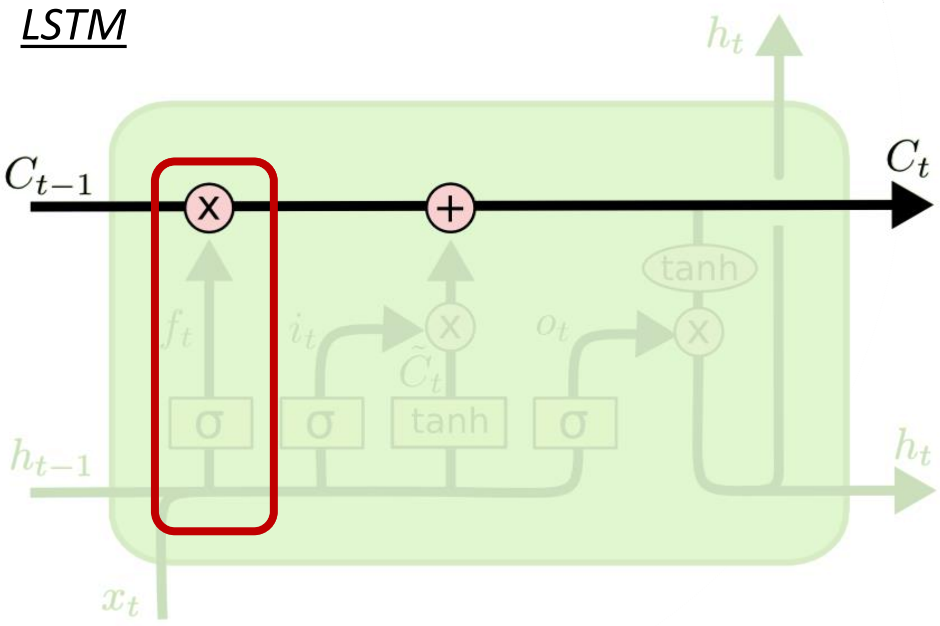


Hochreiter and Schmidhuber, "Long short-term memory," in Neural Computation, 1997. [link]

Long Short-Term Memory (LSTM)





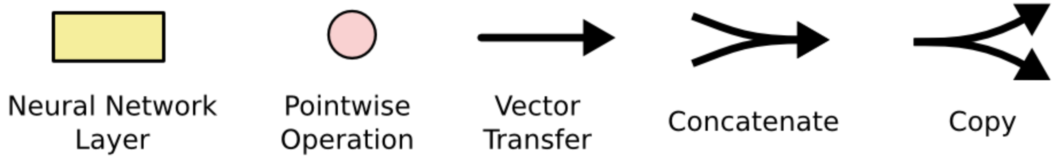


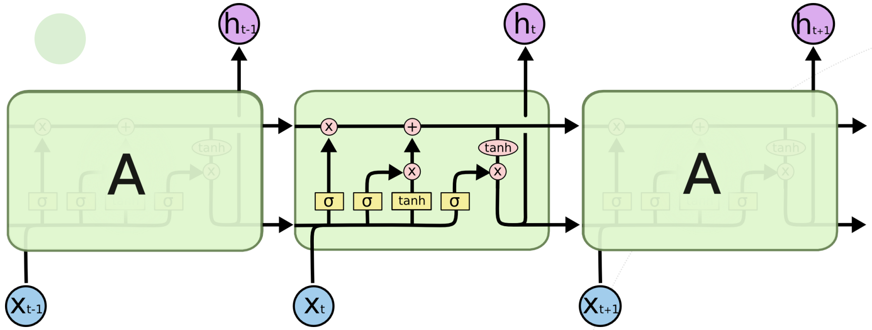
runs straight down the chain with minor linear interactions
→ easy for information to flow along it unchanged

Gates are a way to optionally let information through
→ composed of a sigmoid and a pointwise multiplication operation

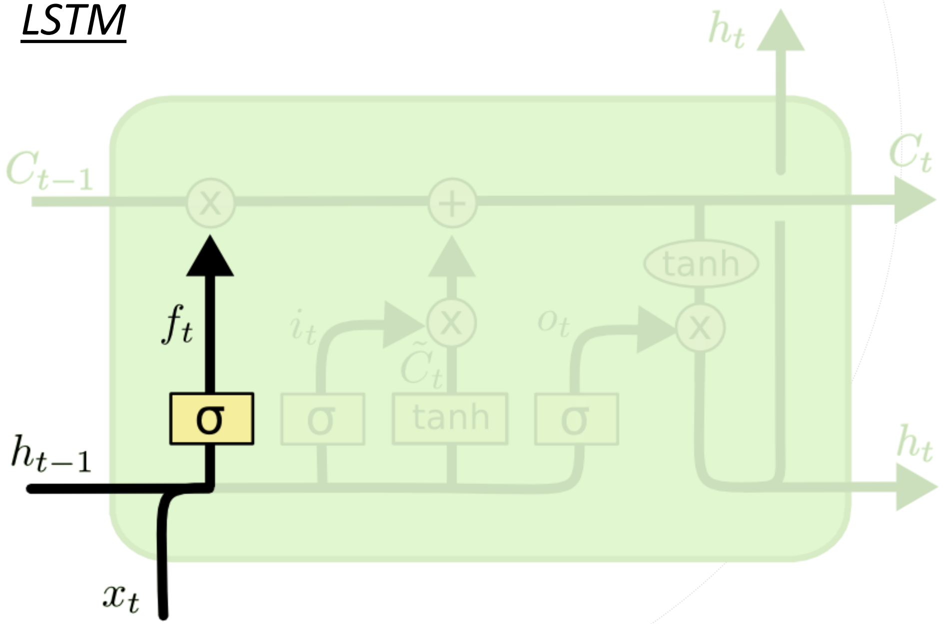
Hochreiter and Schmidhuber, "Long short-term memory," in Neural Computation, 1997. [link]

Long Short-Term Memory (LSTM)





LSTM



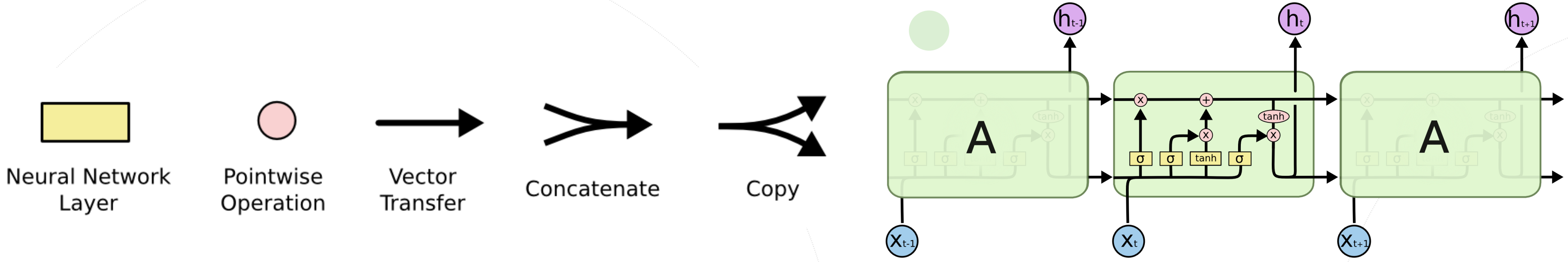
forget gate (a sigmoid layer): decides what information we're going to throw away from the cell state

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

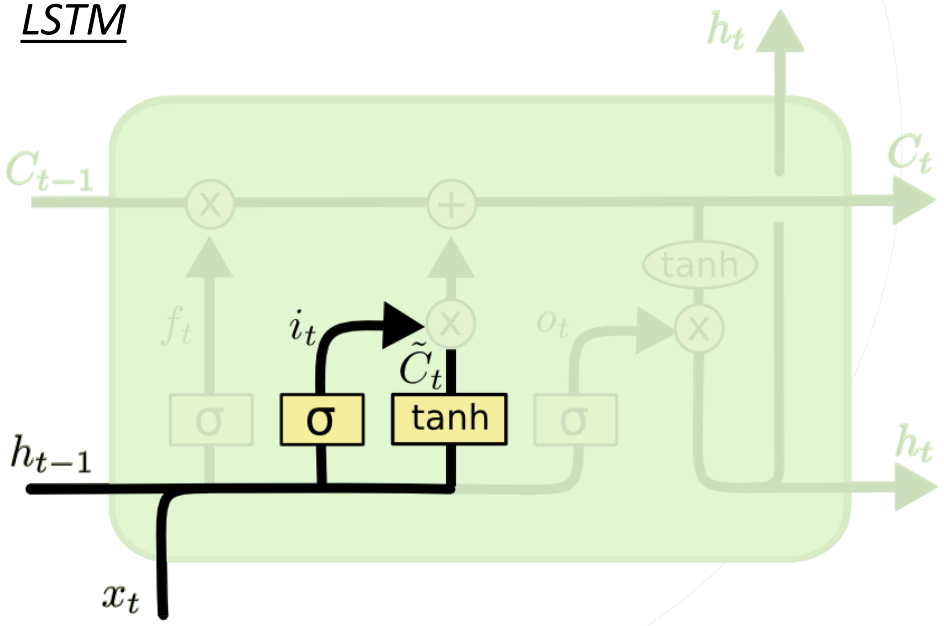
- 1: "completely keep this"
- 0: "completely get rid of this"

Hochreiter and Schmidhuber, "Long short-term memory," in Neural Computation, 1997. [link]

Long Short-Term Memory (LSTM)



LSTM



input gate (a sigmoid layer): decides what new information we're going to store in the cell state

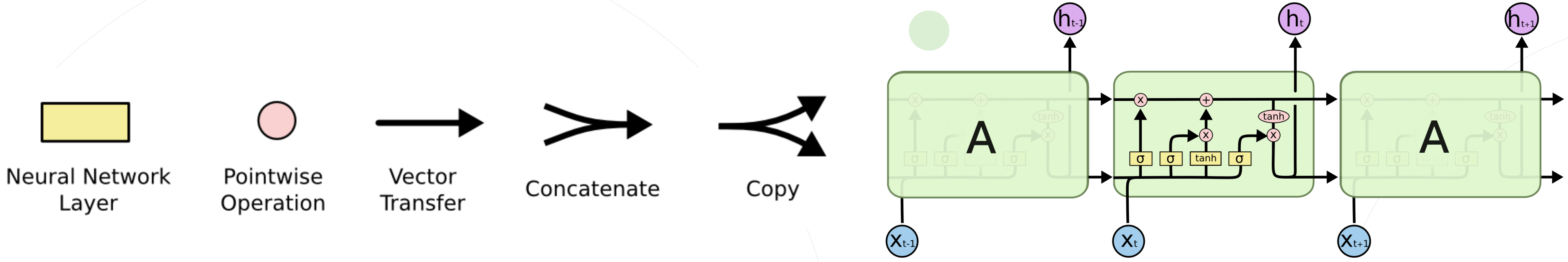
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\boxed{\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)}$$

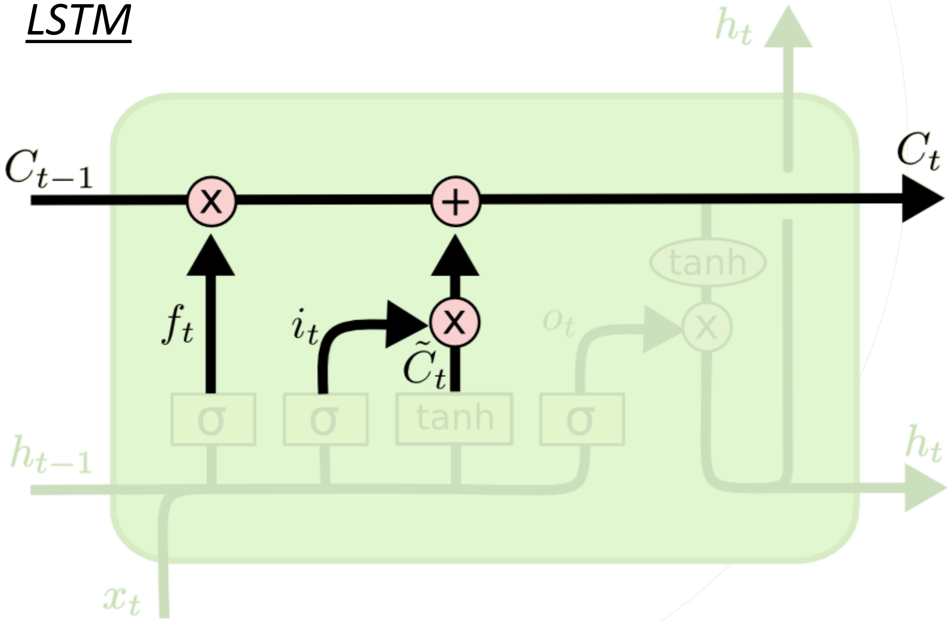
Vanilla RNN

Hochreiter and Schmidhuber, "Long short-term memory," in Neural Computation, 1997. [link]

Long Short-Term Memory (LSTM)



LSTM



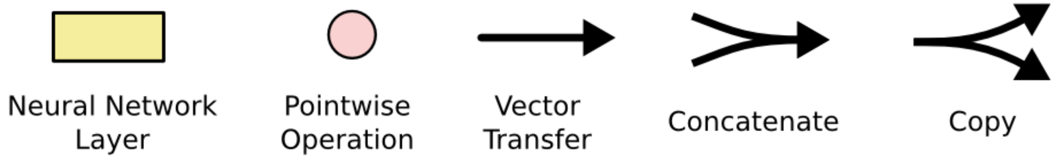
cell state update: forgets the things we decided to forget earlier and add the new candidate values, scaled by how much we decided to update each state value

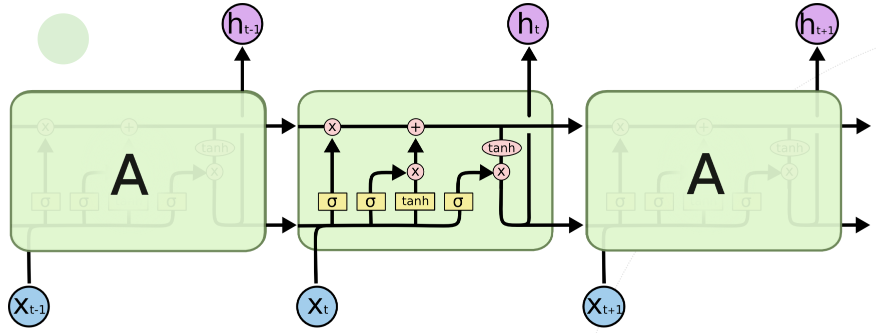
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

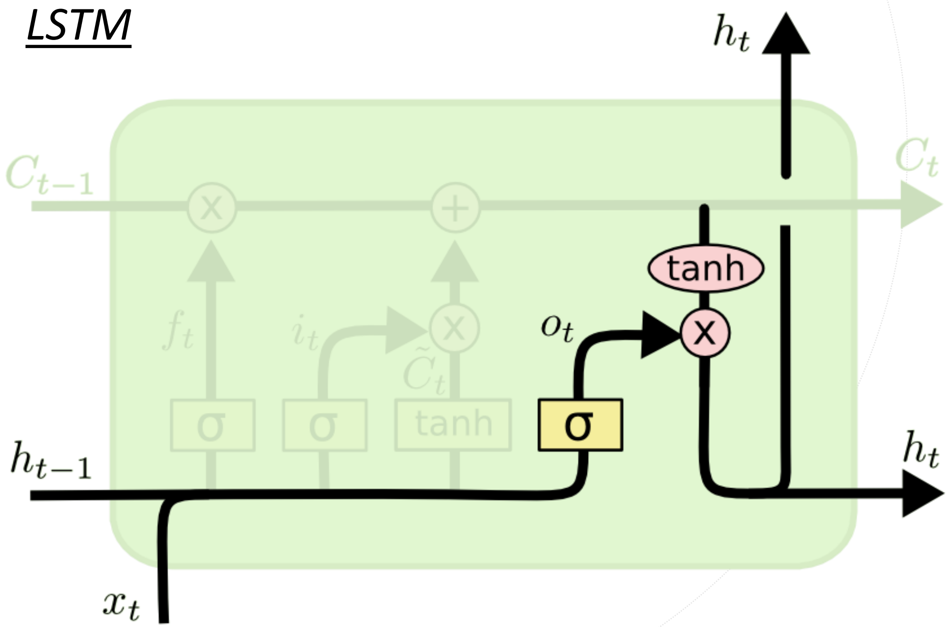
- f_t: decides which to forget
- i_t: decide which to update

Hochreiter and Schmidhuber, "Long short-term memory," in Neural Computation, 1997. [link]

Long Short-Term Memory (LSTM)







output gate (a sigmoid layer):
decides what new information we're going to output

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

$$h_t = o_t * \tanh\left(C_t\right)$$

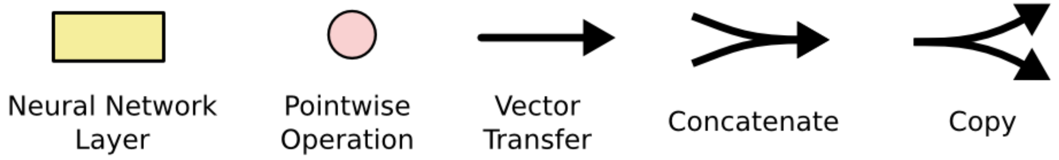
Hochreiter and Schmidhuber, "Long short-term memory," in Neural Computation, 1997. [link]

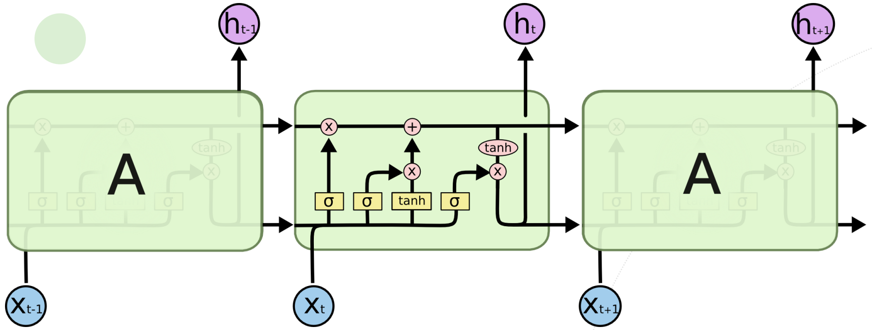
NTU MIULAB

Variants on LSTM

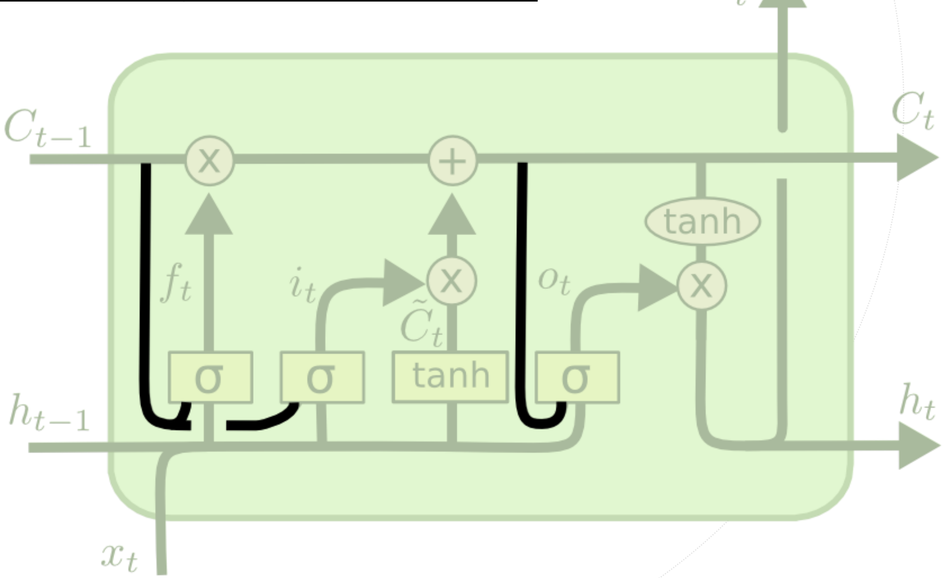
Addressing Vanishing Gradient Problem

LSTM with Peephole Connections





<u>*LSTM with Peephole Conncetions*</u>

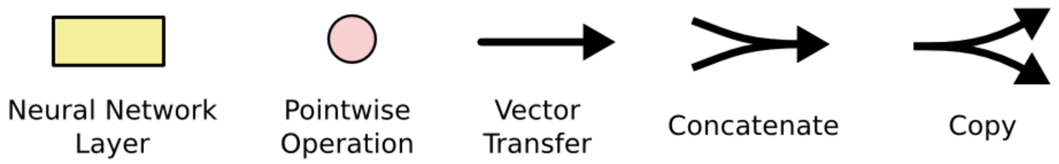


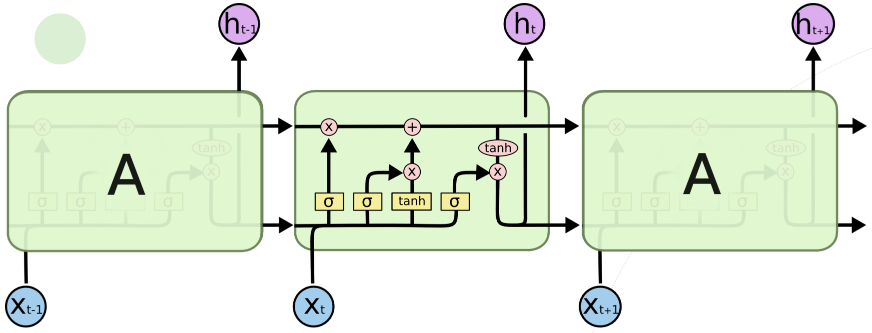
Idea: allow gate layers to look at the cell state

$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] + b_o\right)$$

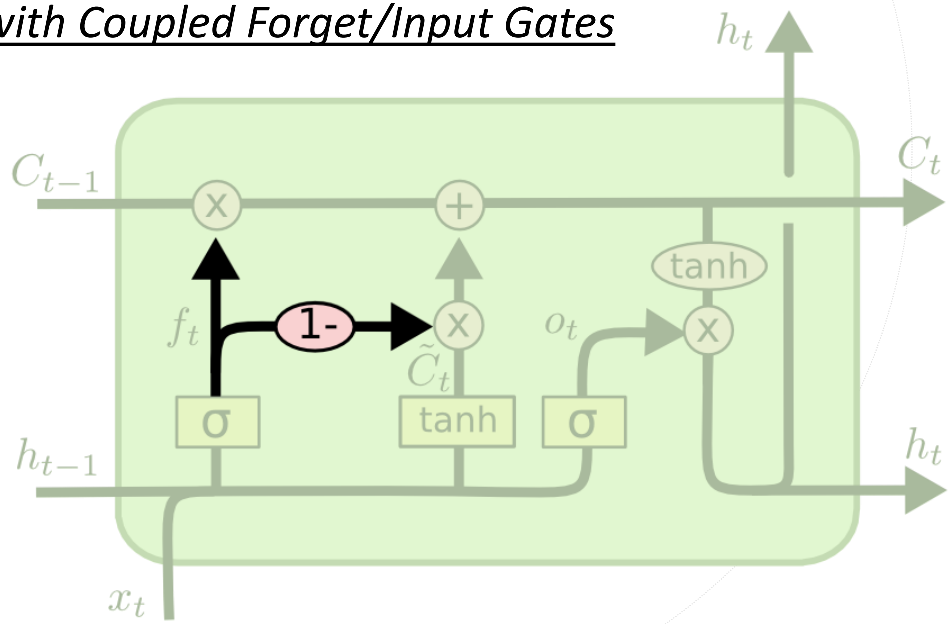
Gers and Schmidhuber, "Recurrent nets that time and count," in IJCNN, 2000. [link]

LSTM with Coupled Forget/Input Gates





LSTM with Coupled Forget/Input Gates



Idea: instead of separately deciding what to forget and what we should add new information to, we make those decisions together

$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_t$$

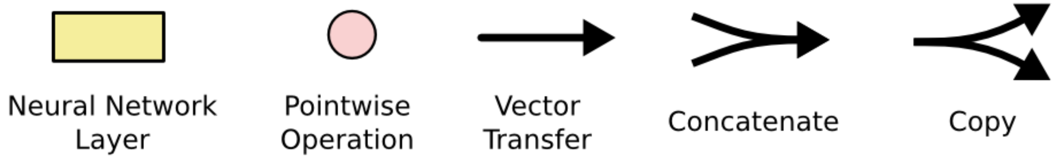
We only forget when we're going to input something in its place, and vice versa.

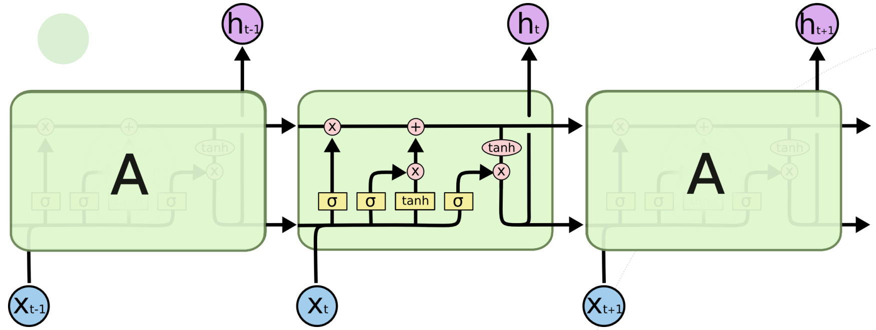
Gers and Schmidhuber, "Recurrent nets that time and count," in IJCNN, 2000. [link]

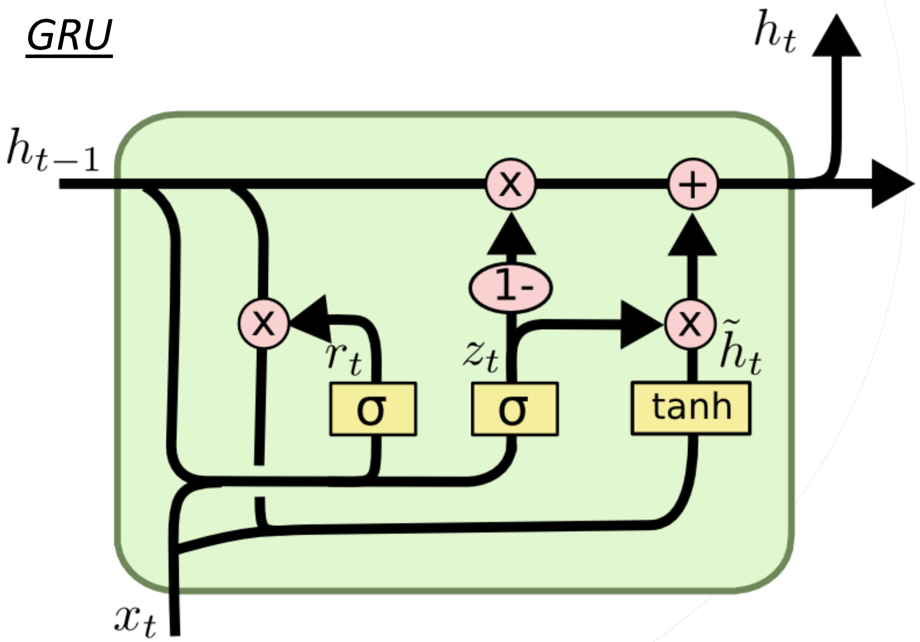
Gated Recurrent Unit

Addressing Vanishing Gradient Problem

Gated Recurrent Unit (GRU)







Idea: combine the forget and input gates into a single "update gate"; merge the cell state and hidden state

update gate: $z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$

reset gate: $r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$r_t=0$: ignore previous memory and only stores the new word information

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

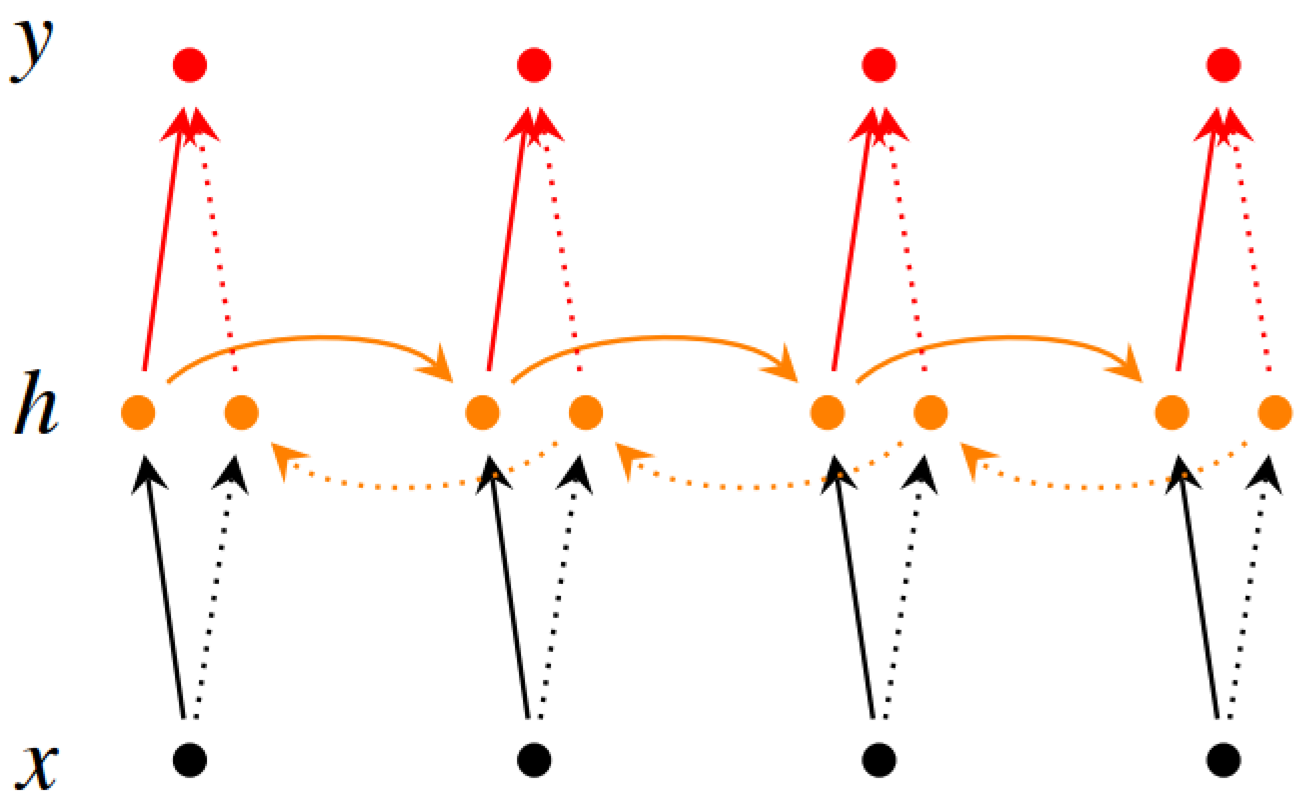
GRU is simpler and has less parameters than LSTM

Cho et al., "Learning phrase representations using RNN encoder-decoder for statistical machine translation," arXiv preprint arXiv:1406.1078, 2014. [link]

Extension

Recurrent Neural Network

Bidirectional RNN



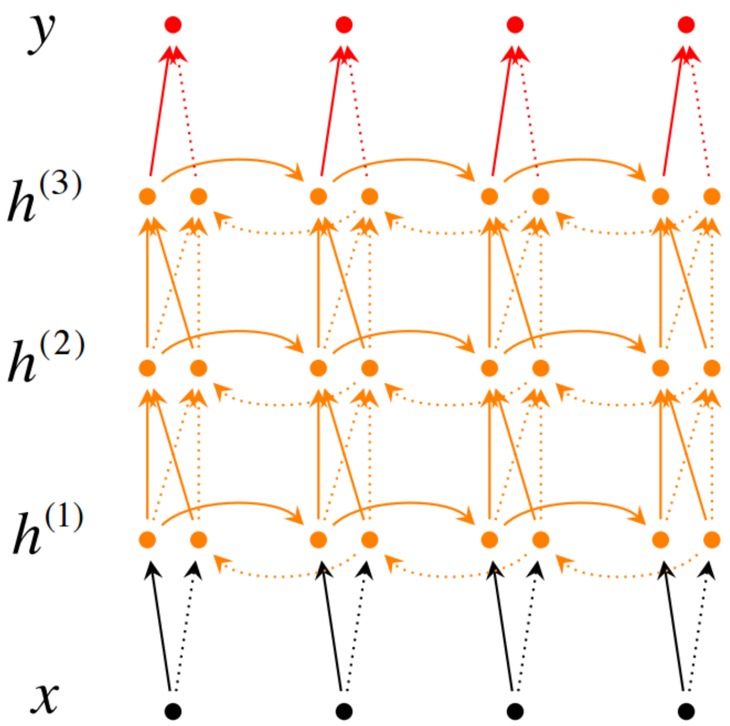
$$\overrightarrow{h}_t = f(\overrightarrow{W} x_t + \overrightarrow{V} \overrightarrow{h}_{t-1} + \overrightarrow{b})$$

$$\overleftarrow{h}_t = f(\overleftarrow{W} x_t + \overleftarrow{V} \overleftarrow{h}_{t+1} + \overleftarrow{b})$$

$$y_t = g(U[\overrightarrow{h}_t ; \overleftarrow{h}_t] + c)$$

$h = [\overrightarrow{h}; \overleftarrow{h}]$ represents (summarizes) the past and future around a single token

Deep Bidirectional RNN



$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\overrightarrow{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

Each memory layer passes an intermediate representation to the next

Outline

- Language Modeling
  - N-gram Language Model
  - Feed-Forward Neural Language Model
  - Recurrent Neural Network Language Model (RNNLM)
- Recurrent Neural Network
  - Definition
  - Training via Backpropagation through Time (BPTT)
  - Training Issue
- **Applications**
  - Sequential Input
  - Sequential Output
    - Aligned Sequential Pairs (Tagging)
    - Unaligned Sequential Pairs (Seq2Seq/Encoder-Decoder)

How to Frame the Learning Problem?

- The learning algorithm f is to map the input domain X into the output domain Y

$$f : X \to Y$$

- Input domain: word, word sequence, audio signal, click logs
- Output domain: single label, sequence tags, tree structure, probability distribution

Network design should leverage input and output domain properties

Outline

- Language Modeling
  - N-gram Language Model
  - Feed-Forward Neural Language Model
  - Recurrent Neural Network Language Model (RNNLM)
- Recurrent Neural Network
  - Definition
  - Training via Backpropagation through Time (BPTT)
  - Training Issue
- Applications
  - **Sequential Input**
  - Sequential Output
    - Aligned Sequential Pairs (Tagging)
    - Unaligned Sequential Pairs (Seq2Seq/Encoder-Decoder)

Input Domain – Sequence Modeling

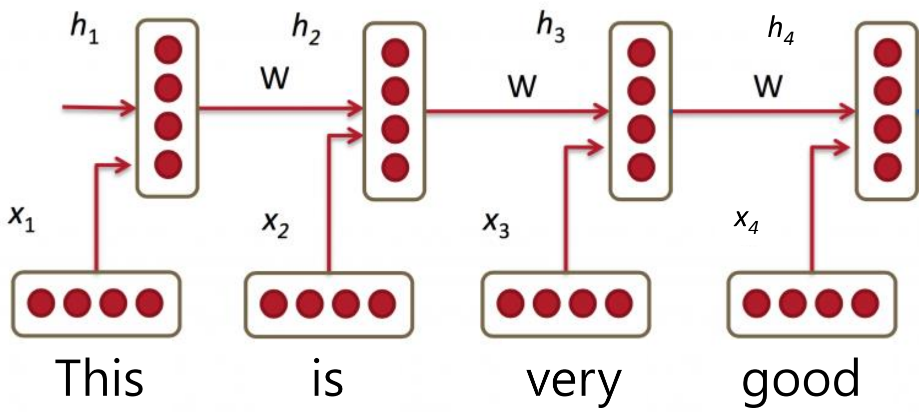
- Idea: aggregate the meaning from all words into a vector
- Method:
  - Basic combination: average, sum
  - Neural combination:
    - ✓ Recursive neural network (RvNN)
    - ✓ Recurrent neural network (RNN)
    - ✓ Convolutional neural network (CNN)

$$
\begin{array}{rl}
 & \overbrace{}^{N\text{-dim}} \\
\text{this} & \begin{bmatrix} 0.2 & 0.6 & 0.3 & \cdots & 0.4 \end{bmatrix} \\
\text{specification} & \begin{bmatrix} 0.9 & 0.8 & 0.1 & \cdots & 0.1 \end{bmatrix} \\
\text{is} & \begin{bmatrix} 0.1 & 0.3 & 0.1 & \cdots & 0.7 \end{bmatrix} \\
\text{good} & \begin{bmatrix} 0.5 & 0.0 & 0.6 & \cdots & 0.4 \end{bmatrix}
\end{array}
$$

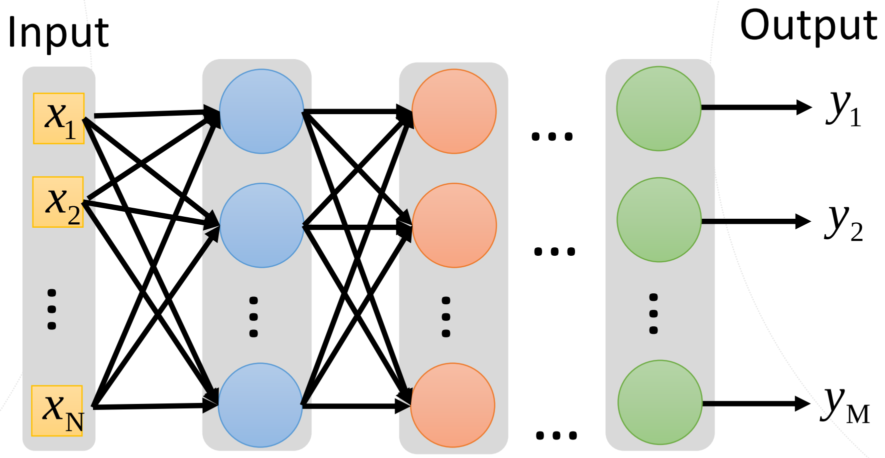
How to compute $\vec{x} = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_N \end{bmatrix}$

Sentiment Analysis

- Encode the sequential input into a vector using RNN



$$\vec{x} = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_N \end{bmatrix}$$



RNN considers temporal information to learn sentence vectors as the input of classification tasks

Outline

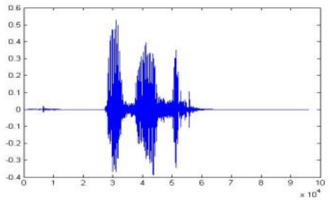
- Language Modeling
  - N-gram Language Model
  - Feed-Forward Neural Language Model
  - Recurrent Neural Network Language Model (RNNLM)
- Recurrent Neural Network
  - Definition
  - Training via Backpropagation through Time (BPTT)
  - Training Issue
- Applications
  - Sequential Input
  - **Sequential Output**
    - Aligned Sequential Pairs (Tagging)
    - Unaligned Sequential Pairs (Seq2Seq/Encoder-Decoder)

Output Domain – Sequence Prediction

- POS Tagging

"I like reading papers." → I/PN like/VBP reading/VBG papers/NNS

- Speech Recognition



→ "Hello"

- Machine Translation

"How are you doing today?" → "你好嗎?"

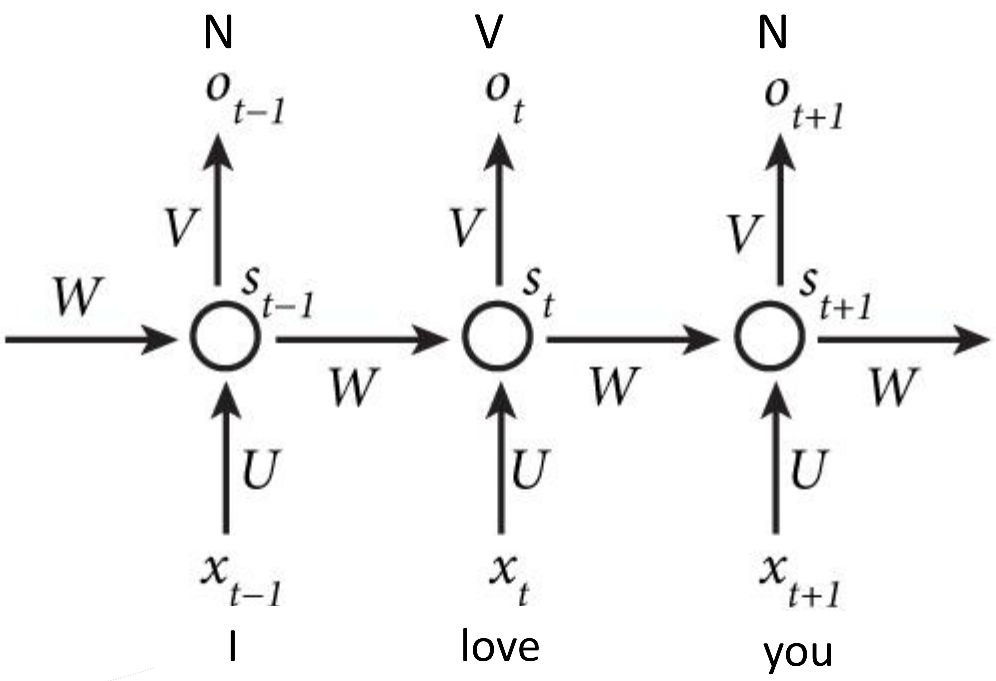
The output can be viewed as a sequence of classification

Outline

- Language Modeling
  - N-gram Language Model
  - Feed-Forward Neural Language Model
  - Recurrent Neural Network Language Model (RNNLM)
- Recurrent Neural Network
  - Definition
  - Training via Backpropagation through Time (BPTT)
  - Training Issue
- Applications
  - Sequential Input
  - Sequential Output
    - **Aligned Sequential Pairs (Tagging)**
    - Unaligned Sequential Pairs (Seq2Seq/Encoder-Decoder)

POS Tagging

- Tag a word at each timestamp
  - Input: word sequence
  - Output: corresponding POS tag sequence



Natural Language Understanding (NLU)

- Tag a word at each timestamp
  - Input: word sequence
  - Output: IOB-format slot tag and intent tag

<START>	just	sent	email	to	bob	about	fishing	this	weekend	<END>
O	O	O	O	O	B-contact_name	O	B-subject	I-subject	I-subject	send_email

→ send_email(contact_name="bob", subject="fishing this weekend")

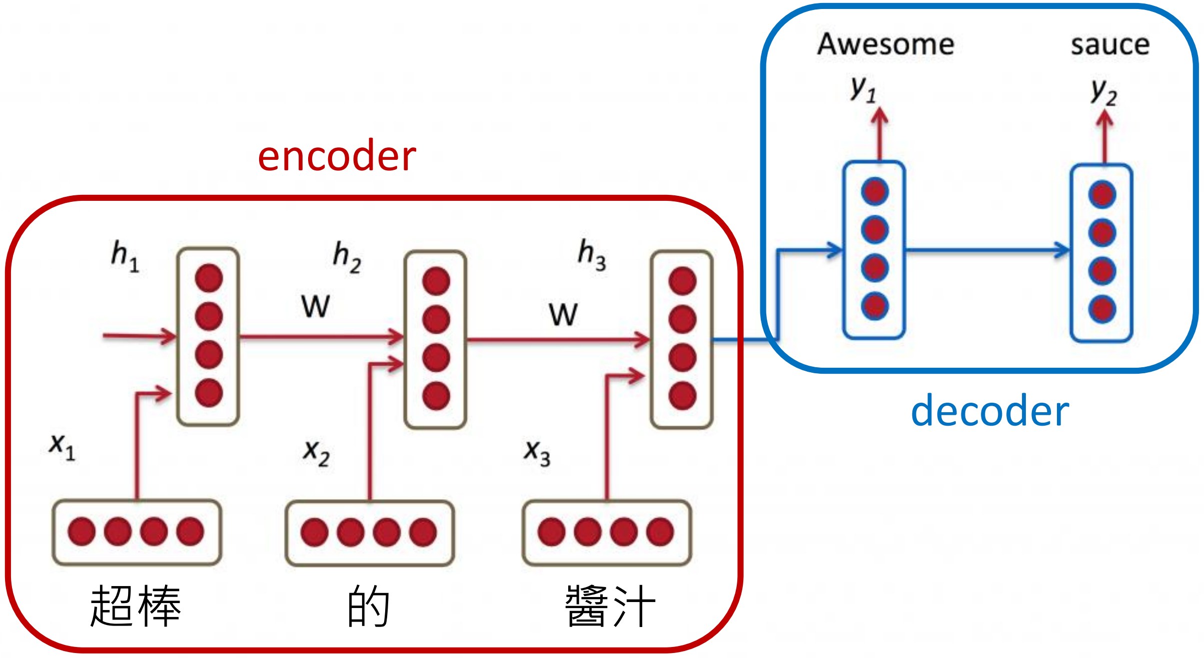
Temporal orders for input and output are the same

Outline

- Language Modeling
  - N-gram Language Model
  - Feed-Forward Neural Language Model
  - Recurrent Neural Network Language Model (RNNLM)
- Recurrent Neural Network
  - Definition
  - Training via Backpropagation through Time (BPTT)
  - Training Issue
- Applications
  - Sequential Input
  - Sequential Output
    - Aligned Sequential Pairs (Tagging)
    - **Unaligned Sequential Pairs (Seq2Seq/Encoder-Decoder)**

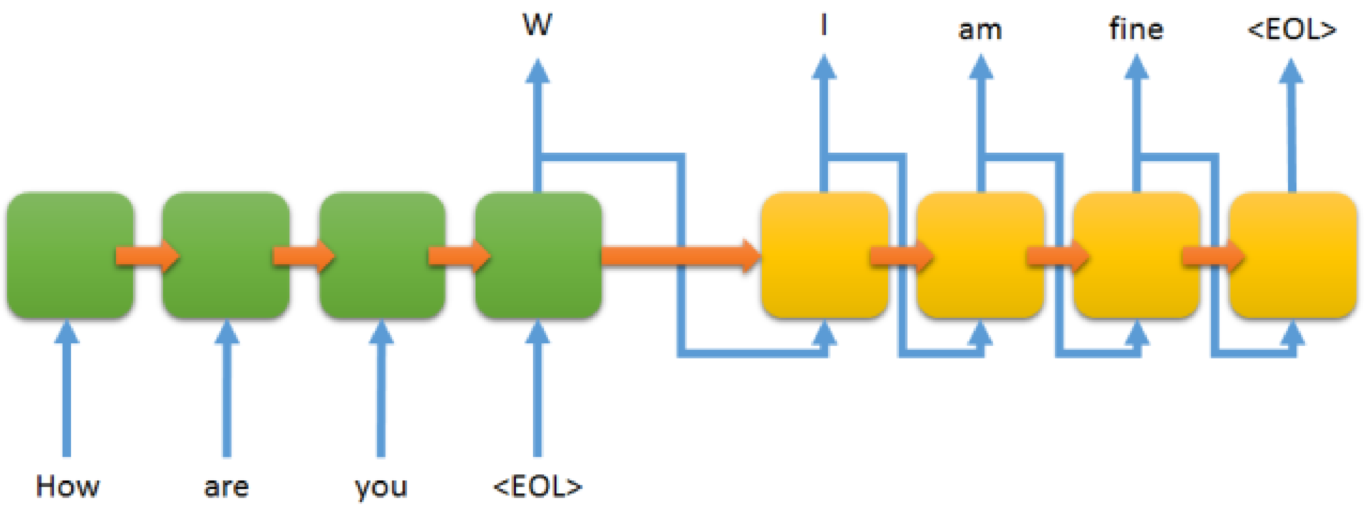
Machine Translation

- Cascade two RNNs, one for encoding and one for decoding
  - Input: word sequences in the source language
  - Output: word sequences in the target language



Chit-Chat Dialogue Modeling

- Cascade two RNNs, one for encoding and one for decoding
  - Input: word sequences in the question
  - Output: word sequences in the response

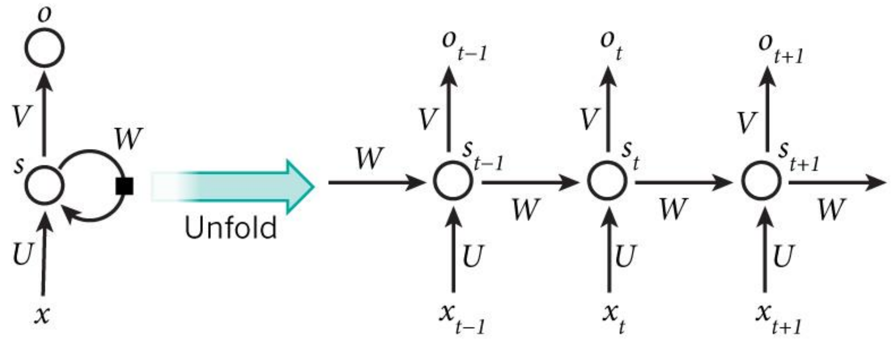


Temporal ordering for input and output may be different

Concluding Remarks

- Language Modeling
  - RNNLM
- Recurrent Neural Networks
  - Definition

$$s_t = \sigma(W s_{t-1} + U x_t)$$
$$o_t = \mathrm{softmax}(V s_t)$$



  - Backpropagation through Time (BPTT)
  - Vanishing/Exploding Gradient
    - Long Short-Term Memory (LSTM)
    - Gated Recurrent Unit (GRU)
- Applications
  - Sequential Input: Sequence-Level Embedding
  - Sequential Output: Tagging / Seq2Seq (Encoder-Decoder)

NTU MIULAB

Q & A

Yun-Nung (Vivian) Chen
yvchen@csie.ntu.edu.tw