南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Advanced Natural Language Processing

# Lecture 3: Word Embedding and Language Model

陈冠华 CHEN Guanhua

Department of Statistics and Data Science

# Content

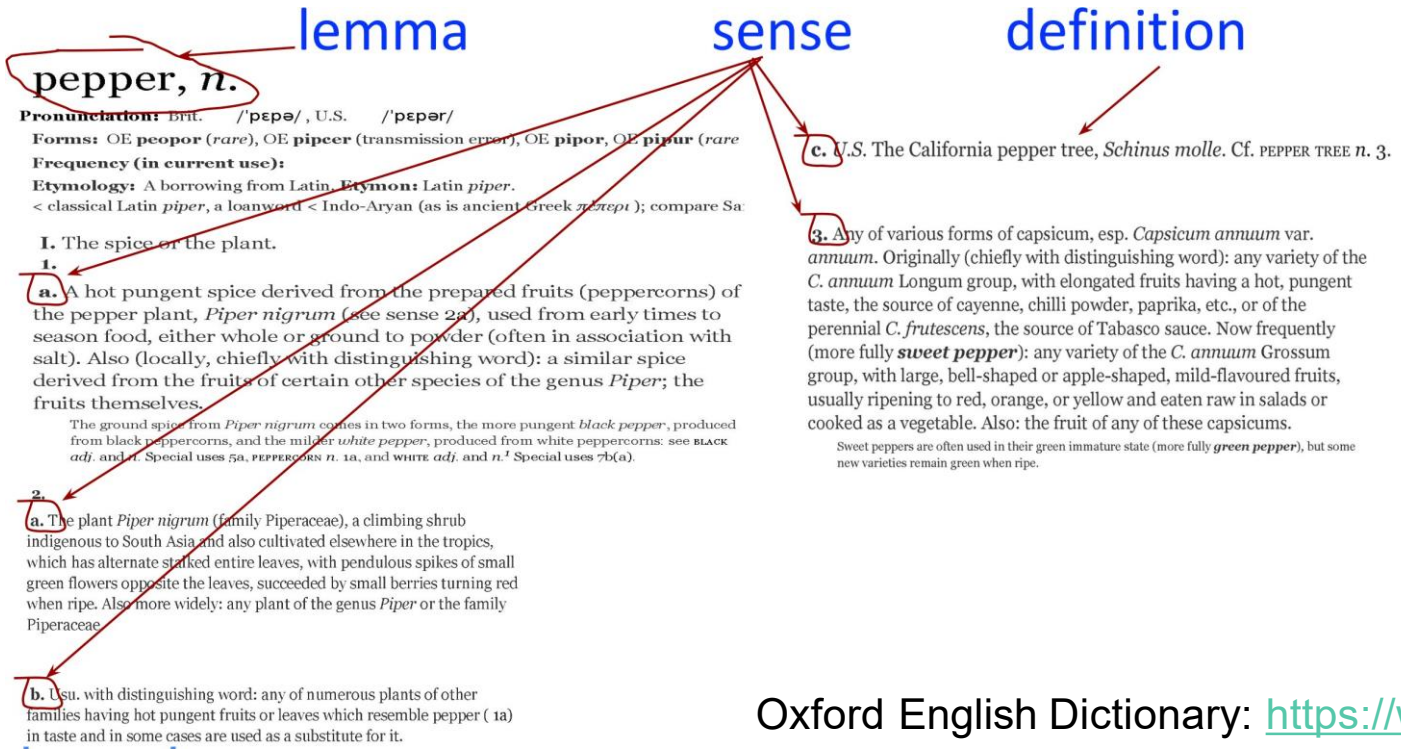
南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Word representations
- Recurrent neural network
- Language model

# Lexical Semantics

How should we represent the meaning of the word?

- Words, lemmas, senses, definition




Oxford English Dictionary: https://www.oed.com/

# Lexical Semantics

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

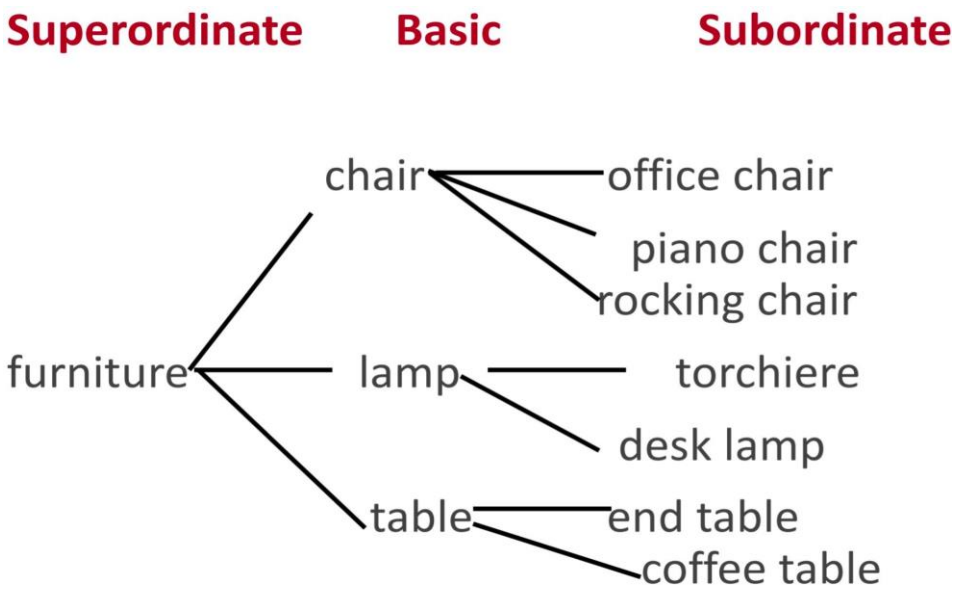
How should we represent the meaning of the word?

- Words, lemmas, senses, definition
- Relationships between words
  - Synonymity: same meaning, e.g., couch/sofa
  - Antonymy: opposite senses, e.g., hot/cold
  - Similarity: similar meanings, e.g., car/bicycle
  - Relatedness: association, e.g., car/gasoline
  - Superordinate/Subordinate: e.g., car/vehicle, mango/fruit

# Lexical Semantics

How should we represent the meaning of the word?

- Words, lemmas, senses, definition
- Relationships between words or senses
- Taxonomy: abstract -> concrete



# Lexical Resources



**WordNet Search - 3.1**
- WordNet home page - Glossary - Help

Word to search for: mouse [Search WordNet]

Display Options: (Select option to change) [Change]

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations
Display options for sense: (gloss) "an example sentence"

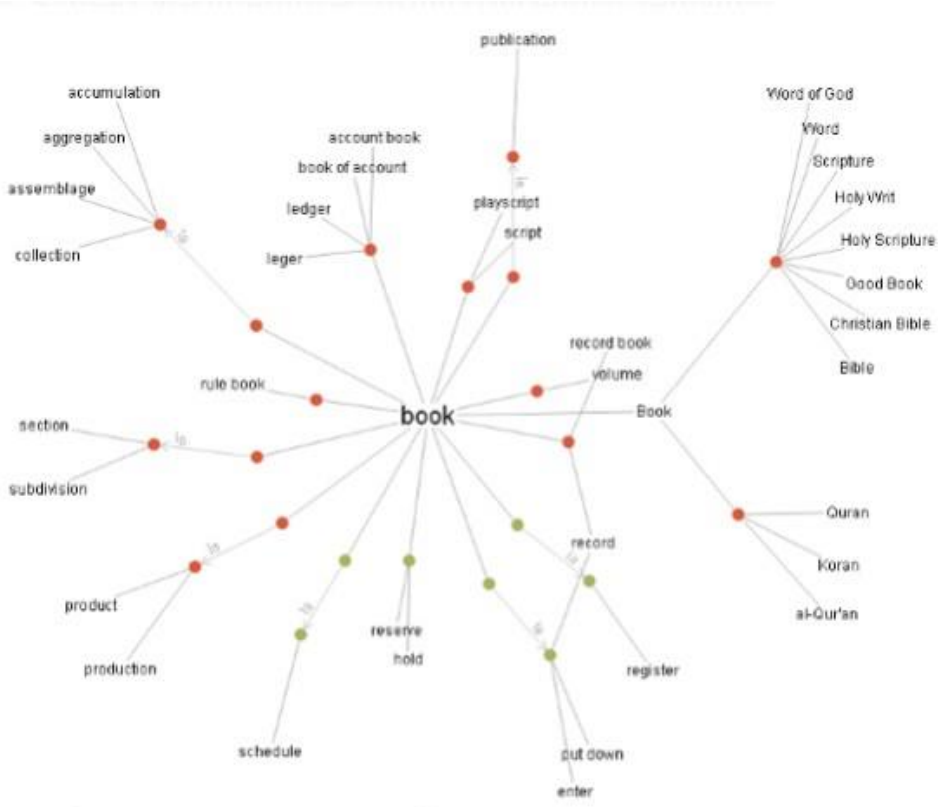
**Noun**

- S: (n) **mouse** (any of numerous small rodents typically resembling diminutive rats having pointed snouts and small ears on elongated bodies with slender usually hairless tails)
- S: (n) shiner, black eye, **mouse** (a swollen bruise caused by a blow to the eye)
- S: (n) **mouse** (person who is quiet or timid)
- S: (n) **mouse**, computer mouse (a hand-operated electronic device that controls the coordinates of a cursor on your computer screen as you move it around on a pad; on the bottom of the device is a ball that rolls on the surface of the pad) *"a mouse takes much more room than a trackball"*

**Verb**

- S: (v) sneak, **mouse**, creep, pussyfoot (to go stealthily or furtively) *"..stead of sneaking around spying on the neighbor's house"*
- S: (v) **mouse** (manipulate the mouse of a computer)

WordNet Search - 3.1 (princeton.edu)




Huge amounts of human labor to create

Visual Wordnet with D3.js　　wordnet可视化

# Methods to Represent Words

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Theories of language tend to view the data (words, sentences, documents) and abstractions over it as symbolic or categorical.
  - Uses symbols to represent linguistic information

- Machine learning algorithms built on optimization rely more on continuous data.
  - Uses floating-point numbers (vectors)

How to enable machines to understand words?

# One-Hot Word Vector

One-hot vector (独热向量) $w \in R^{|V|}$

- Sparse
- Expensive
- Hard to compute word relationships

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *expert* | [0 | 0 | 0 | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0] |
| *skillful* | [0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** | 0 | 0 | 0 | 0] |

# One-Hot Word Vector

- Use word-word co-occurrence counts to represent the meaning of words!

$$P(w_1, w_2, ...w_n) = \prod_{i=1}^{n} P(w_i|w_{i-1})$$

$$P(w_i|w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha|V|}$$

- cat = the 5th word
- dog = the 10th word
- cats = the 118th word

Each word is just a string or indices $w_i$ in the vocabulary list

# Distributional Hypothesis

**Distributional hypothesis**:

Words that occur in similar **contexts** tend to have similar meanings.

(J.R.Firth 1957)

- "You shall know a word by the company it keeps"
- 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).

| | | |
|---|---|---|
| *…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*".

# Example

What does “Ong Choy” mean?

- Suppose you see these sentences:
  - Ong Choy is delicious sautéed with garlic
  - Ong Choy is superb over rice
  - Ong Choy leaves with salty sauces
- And you’ve also seen these:
  - … water spinach sautéed with garlic over rice
  - Chard stems and leaves are delicious
  - Collard greens and other salty leafy greens

空心菜；通菜

# Model of Meaning Focusing on Similarity

- Each word = a vector
  - Similar words are “nearby in space”
  - The standard way to represent meaning in NLP

# Sparse vs Dense Vectors

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- The vectors in the word-word occurrence matrix are
  - Long: vocabulary size
  - Sparse: most are 0's
- Alternative: we want to represent words as short (50-300 dimensional) & dense (real-valued) vectors
  - The basis for modern NLP systems

$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \quad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \quad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

# Why Dense Vectors?

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Short vectors are easier to use as features in ML systems
- Dense vectors generalize better than explicit counts (points in real space vs points in integer space)
- Sparse vectors can't capture higher-order co-occurrence
  - $w_1$ co-occurs with "car", $w_2$ co-occurs with "automobile"
  - They should be similar but they aren't, because "car" and "automobile" are distinct dimensions
- In practice, they work better!

# Word Embeddings

- Word embeddings = Learned representations from text for representing words

Input: a large text corpora, $V, d$

- $V$: a pre-defined vocabulary
- $d$: dimension of word vectors (e.g. 300)

Output $f: V \rightarrow R^d$

$$v_{\text{cat}} = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix} \quad v_{\text{dog}} = \begin{pmatrix} -0.124 \\ 0.430 \\ -0.200 \\ 0.329 \end{pmatrix}$$

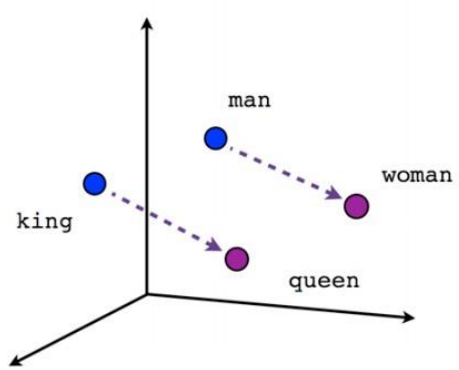
$$v_{\text{the}} = \begin{pmatrix} 0.234 \\ 0.266 \\ 0.239 \\ -0.199 \end{pmatrix} \quad v_{\text{language}} = \begin{pmatrix} 0.290 \\ -0.441 \\ 0.762 \\ 0.982 \end{pmatrix}$$

Each word is represented by a low-dimensional (e.g., d = 300), real-valued vector

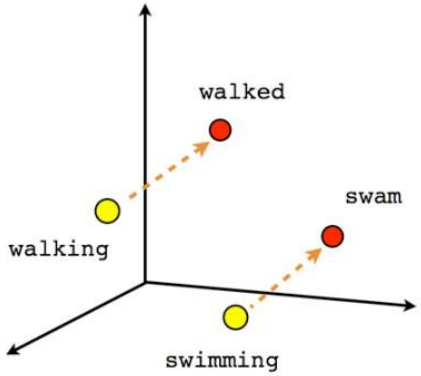
Each coordinate/dimension of the vector doesn't have a particular interpretation

# Word Embeddings

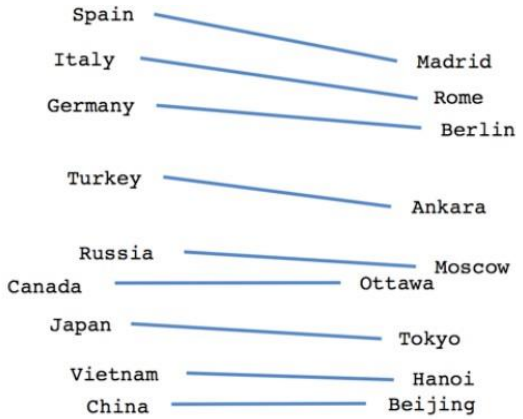
- They have some nice properties




Male-Female




Verb tense




Country-Capital

$$v_{\text{man}} - v_{\text{woman}} \approx v_{\text{king}} - v_{\text{queen}}$$

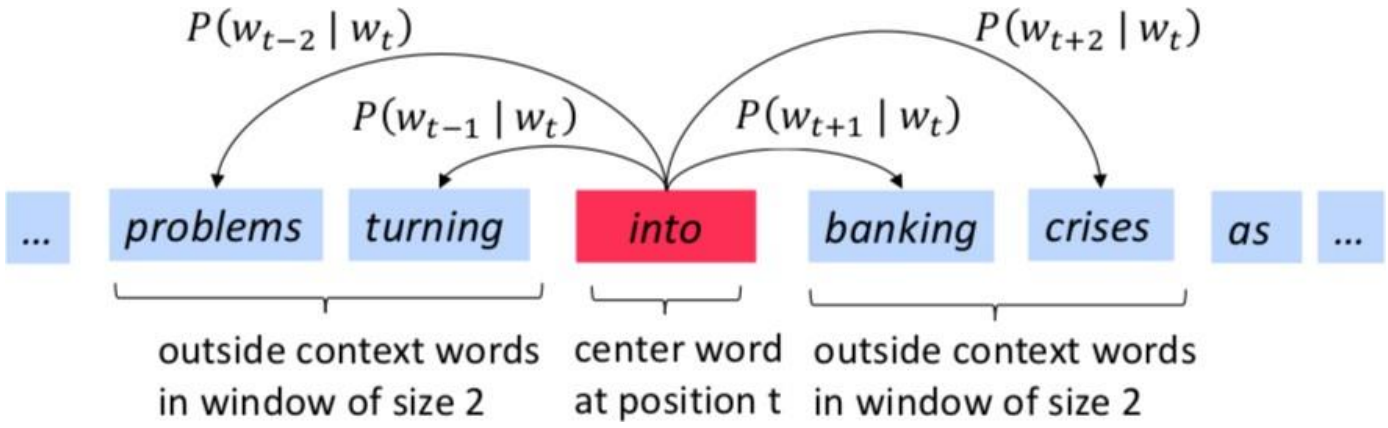
$$v_{\text{Paris}} - v_{\text{France}} \approx v_{\text{Rome}} - v_{\text{Italy}}$$

Word analogy test: $a : a^* :: b : b^*$

$$b^* = \arg\max_{w \in V} \cos(e(w), e(a^*) - e(a) + e(b))$$

# Word2vec

- Key idea: Use each word to predict other words in its context
  - A classification problem
- Assume that we have a large corpus $w_1, w_2, \ldots, w_T \in V$
- Context: a fixed window of size $2m$ ($m = 2$ in the example)



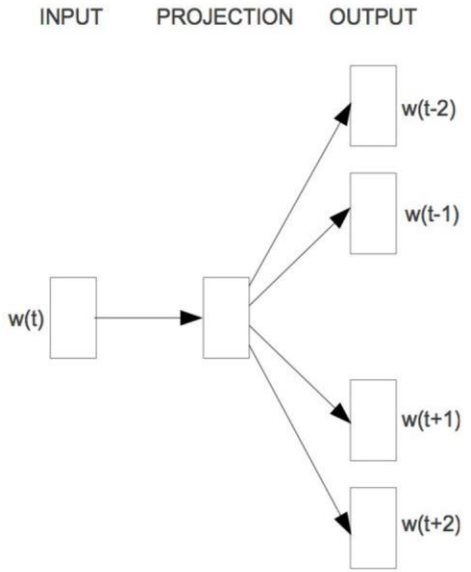
# Word2vec

- Assumption: words under different context have the same vector

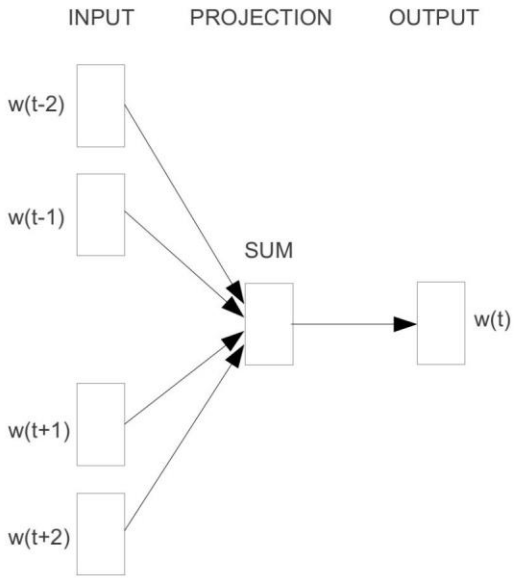
Two word2vec algorithms:

- Skip-gram: $p(c|w)$
- Continuous bag of words (CBOW): $p(w|c)$

  - Context: $c$
  - Target word: $w$



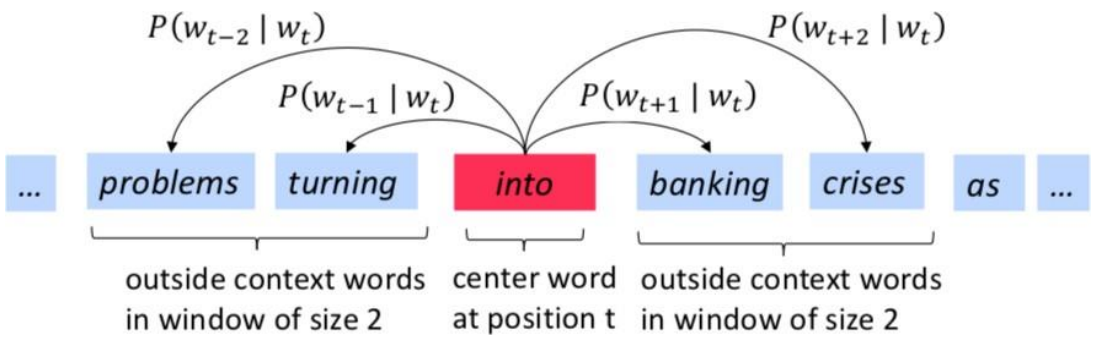

Skip-gram




CBOW

# Skip-Gram Objective

- Assume that we have a large corpus $w_1, w_2, \ldots, w_T \in V$
- Context: a fixed window of size $2m$ ($m = 2$ in the example)
- For each word in the corpus




Convert the training data into:
(into, problems)
(into,turning)
(into,banking)
(into, crises)
(banking, turning)
(banking, into)
(banking, crises)
(banking, as)
…

Our goal is to find parameters that can maximize

$P(\text{problems} \mid \text{into}) \times P(\text{turning} \mid \text{into}) \times P(\text{banking} \mid \text{into}) \times P(\text{crises} \mid \text{into}) \times P(\text{turning} \mid \text{banking}) \times P(\text{into} \mid \text{banking}) \times P(\text{crises} \mid \text{banking}) \times P(\text{as} \mid \text{banking}) \ldots$

# Skip-Gram Objective

- For each word $w_t$ in the corpus, we predict context words within context size $m$, given center word $w_t$:

all the parameters to be optimized

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

- Usually, we optimize the model by minimizing the (average) negative log likelihood

$$\mathcal{L}(\theta) = -\frac{1}{T} \sum_{1}^{T} \sum_{-m \le j \le m, j \ne 0} \log P\left(w_{t+j} \middle| w_t; \theta\right)$$

- Where $T$ is the sentence length, $t + j \in [1, T]$

What is the difference between "likelihood" and "probability"? (stackexchange.com)

# Modeling the Probability

- Use two sets of vectors for each word in the vocabulary
  - $u_a \in R^d$, vector for center word $a \in V$
  - $v_b \in R^d$, vector for context word $b \in V$
- Use inner product $(u_a \cdot v_b)$ and softmax to measure how likely word $a$ appears with context word $b$

$$P(w_{t+j} \mid w_t) = \frac{\exp(u_{w_t} \cdot v_{w_{t+j}})}{\sum_{k \in V} \exp(u_{w_t} \cdot v_k)}$$

$P(\cdot \mid a)$ is a probability distribution defined over V: $\sum_{w \in V} P(w \mid a) = 1$

**A 4-dimensional embedding**

| | | | | |
|---|---|---|---|---|
| **cat** => | 1.2 | -0.1 | 4.3 | 3.2 |
| **mat** => | 0.4 | 2.5 | -0.9 | 0.5 |
| **on** => | 2.1 | 0.3 | 0.1 | 0.4 |
| ... | ... | | | |

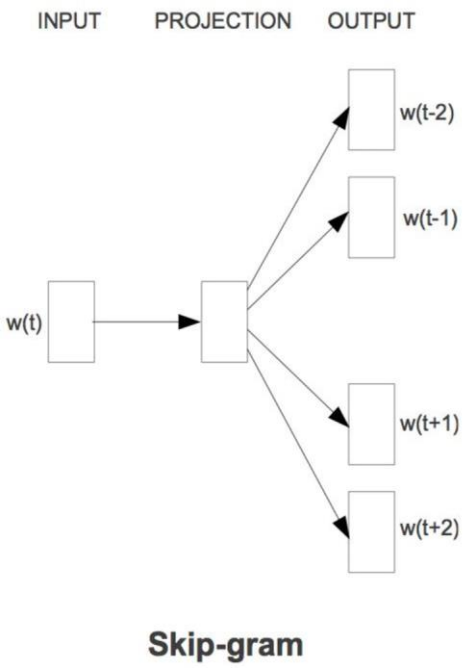
# Skip-Gram Objective

- Optimize the model by minimizing the (average) negative log likelihood

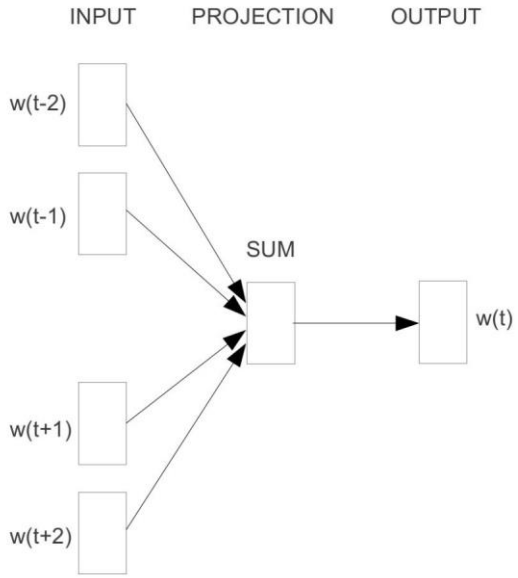
$$\mathcal{L}(\theta) = -\frac{1}{T}\sum_{1}^{T}\sum_{-m\le j\le m, j\neq 0} \log P\left(w_{t+j}\middle|w_t;\theta\right)$$

- Here $\theta$ is the model parameter set
- The model has $2d|V|$ parameters
- Each word has two vectors, why?
  - As word $a$ is not likely to appear in its context, thus $p(a|a)$ should be low.
  - This is not the case when each word has one vector, as $(u_a, u_a)$ is always 1.
- After trained, we usually use the center word vector $u_a$, or concatenate them.
- The model parameters can be optimized with SGD, etc.

# Continuous Bag of Words (CBOW)




Skip-gram




CBOW

$$L(\theta) = \prod_{t=1}^{T} P\left(w_t \mid \{w_{t+j}\}, -m \le j \le m, j \ne 0\right)$$

$$\bar{\mathbf{v}}_t = \frac{1}{2m} \sum_{-m \le j \le m, j \ne 0} \mathbf{v}_{t+j}$$

$$P(w_t \mid \{w_{t+j}\}) = \frac{\exp(\mathbf{u}_{w_t} \cdot \bar{\mathbf{v}}_t)}{\sum_{k \in V} \exp(\mathbf{u}_k \cdot \bar{\mathbf{v}}_t)}$$

[1301.3781] Efficient Estimation of Word Representations in Vector Space (arxiv.org)

15.1. Word Embedding (word2vec) — Dive into Deep Learning

# Available Dense Embeddings

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Word2vec (Mikolov et a. 2013)
  - https://code.google.com/archive/p/word2vec/
- GloVe (Pennington et al. 2014)
  - http://nlp.stanford.edu/projects/glove/
- Fasttext (Bojanowsi et al. 2017)
  - http://www.fasttext.cc/
- Tencent AI Lab Embedding
  - https://ai.tencent.com/ailab/nlp/en/embedding.html

# Word2Vec Coding Practice

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Python package gensim
- Tutorial for gensim

```
import gensim.downloader as api
word_vectors = api.load("glove-wiki-gigaword-50")
word_vectors.most_similar("glass")
# Check the "most similar words", using the default "cosine
similarity" measure.

result = word_vectors.most_similar(positive=['woman', 'king'],
negative=['man'])
most_similar_key, similarity = result[0]  # look at the first match
print(f"{most_similar_key}: {similarity:.4f}")
# >>> queen: 0.7699
```

# Data Preprocess

- Language identification
- Filtering
  - Topic
  - Content
  - Length
- Tokenization
  - Subword
  - Chinese vs. English

75 chinese 71414
76 government 71063
77 我们 69333
78 its 69235
79 or 68129
80 year 67948
81 要 67457
82 以 67367
83 taiwan 67206
84 于 65245
85 美国 65143
86 就 64294
87 个 64214
88 经济 64120
89 并 64022
90 should 63358
91 us 62842

# Data Preprocess

- Subword
  - Byte-pair-encoding (BPE)
  - WordPiece
  - SentencePiece

Neural Machine Translation of Rare Words with Subword Units - ACL Anthology

```
# echo "I saw a girl with a telescope."
_I _saw _a _girl _with _a _ te le s c o pe
```

- Note:
  - phone and __phone are two different tokens.

Tokenizers: How machines read (floydhub.com)

```
75 chinese 71414
76 government 71063
77 我们 69333
78 its 69235
79 or 68129
80 year 67948
81 要 67457
82 以 67367
83 taiwan 67206
84 于 65245
85 美国 65143
86 就 64294
87 个 64214
88 经济 64120
89 并 64022
90 should 63358
91 us 62842
```

```
_ဘာ: 1
_hyper 1
_نتفر 1
是不 1
_Aff 1
_Kenn 1
_авар 1
_dağ 1
_pinaka 1
_대학 1
_asegura 1
ꙮ 1
سوم 1
_이를 1
BP 1
_agat 1
```

# Data Preprocess

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

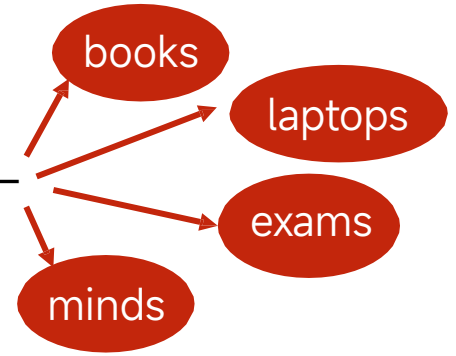
- SentencePiece toolkit

```
# Train a sentencepiece model on text corpus
spm_train --input=data/botchan.txt --model_prefix=myspm --vocab_size=1000

# Tokenize a sentence using the trained sentencepiece model
echo "I saw a girl with a telescope." | spm_encode --model=myspm.model
# >>> _I _saw _a _girl _with _a _ te le s c o pe .

# Tokenize a sentence into token ids using the trained sentencepiece model
echo "I saw a girl with a telescope." | spm_encode --model=myspm.model --output_format=id
# >>> 9 459 11 939 44 11 4 142 82 8 28 21 132 6

# Detokenize into text sentence from token ids
echo "9 459 11 939 44 11 4 142 82 8 28 21 132 6" | spm_decode --model=myspm.model --input_format=id
# >>> I saw a girl with a telescope.
```

# Language Modeling

- Language Modeling is the task of predicting what word comes next

The students opened their _____




- Given a sequence of words $x_1, x_2, \ldots, x_{i-1}$, compute the probability distribution of the next word $x_i$ $(x_i \in V = \{w_1, w_2, \ldots, w_{|V|}\})$:

$$p(x_t \mid x_{<t})$$

- N-gram Language Model
  - The probability of the next word depends only on a fixed size window (n-1) of previous words.

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1}) \approx \prod_{i=2}^{m} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$$

# Language Modeling

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Calculating the probability of a sentence

$$p(X) = \prod_{i=1}^{L} p(x_i | x_{<i})$$

- We can use LM to
  - Score a sentence
  - Generate a sentence

> **while** didn't choose end-of-sentence symbol:
>  **calculate** probability
>  **sample** a new word from the probability distribution

Statistical Language Modeling

# Evaluating Language Models

- The standard evaluation metric for Language Models is perplexity.

$$PPL = 2^{-l}$$

$$l = \frac{1}{T}\sum_{t=1}^{T} \log p_\theta(x_t|x_{<t})$$

- In some cases, it is also written as

$$\text{perplexity} = \prod_{t=1}^{T}\left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)}|\ \boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)})}\right)^{1/T}$$

Normalized by number of words

Inverse probability of corpus, according to Language Model

# Evaluating Language Models

- This is equal to the exponential of the cross-entropy loss $\mathcal{L}(\theta)$

$$\mathcal{L}(\theta) = CE(y_t, \hat{y_t}) = -\sum_{i=1}^{|V|} \hat{y}_t^i \log y_t^i$$

- $\hat{y}_t^i$ is a one-hot vector that stands for ground-truth token distribution
- PPL score is also equivalent to the exponentiation of the cross-entropy between the data and model predictions. [link]
- Lower perplexity is better

# Cross-Entropy Loss

- Useful when training a classification problem with multiple classes
  - The accuracy tells the model whether or not a particular prediction is correct
  - The cross-entropy loss gives information on how correct a particular prediction is
- Given a true distribution $t$ and a predicted distribution $p$, the cross entropy between them is given by the following equation

$$\mathcal{L} = -\sum_{i \in V} t(i) \log p(i)$$

- Where the true distribution $t = [0,0,..0,1,0,\dots,0]$, $p = [0.02,0.07,\dots,0.67,0.12,\dots,0.01]$
- Code for cross_entropy loss fairseq

# Cross-Entropy Loss

Source sequence:

Я видел котю на мате <eos>

"I" "saw" "cat" "on" "mat"

Target sequence:

I saw a cat on a mat <eos>

previous tokens

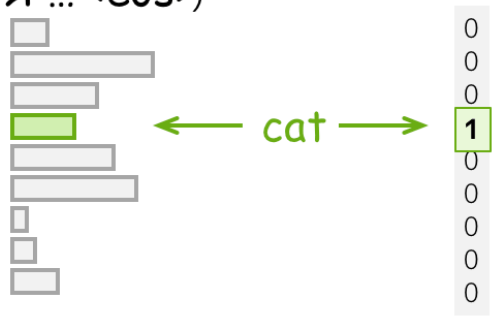
we want the model to predict this

← one training example

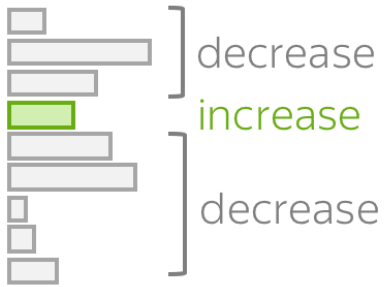
← one step for this example

Model prediction: p( ∗ |I saw a, Я ... <eos>)

Target



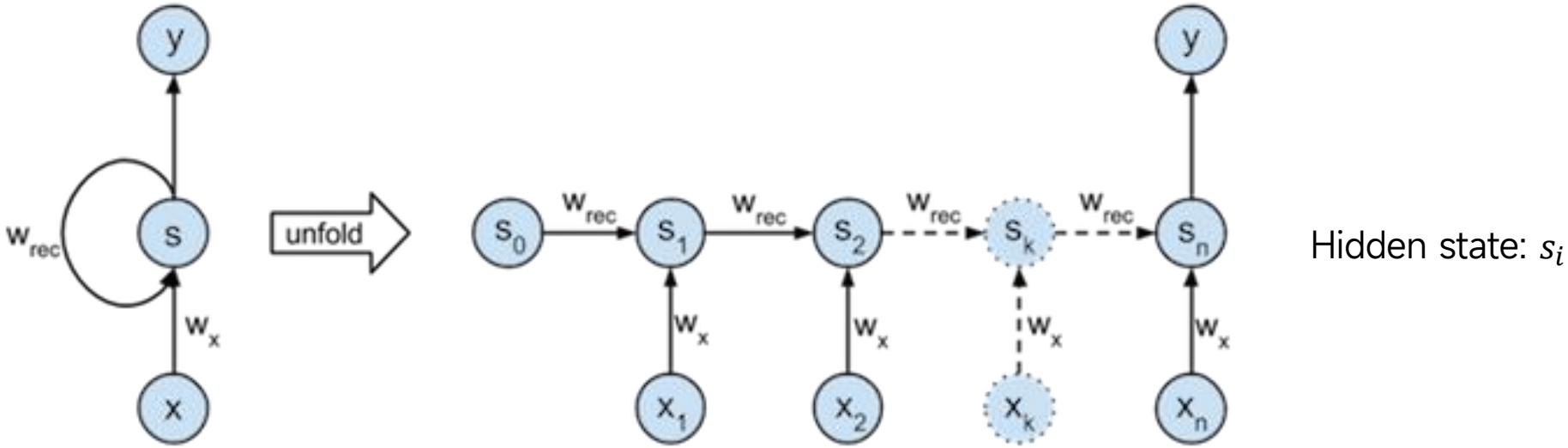

Loss = -log (p(cat)) → min



# Recurrent Neural Network (RNN)

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- A family of neural networks that can handle variable length inputs




Hidden state: $s_i$

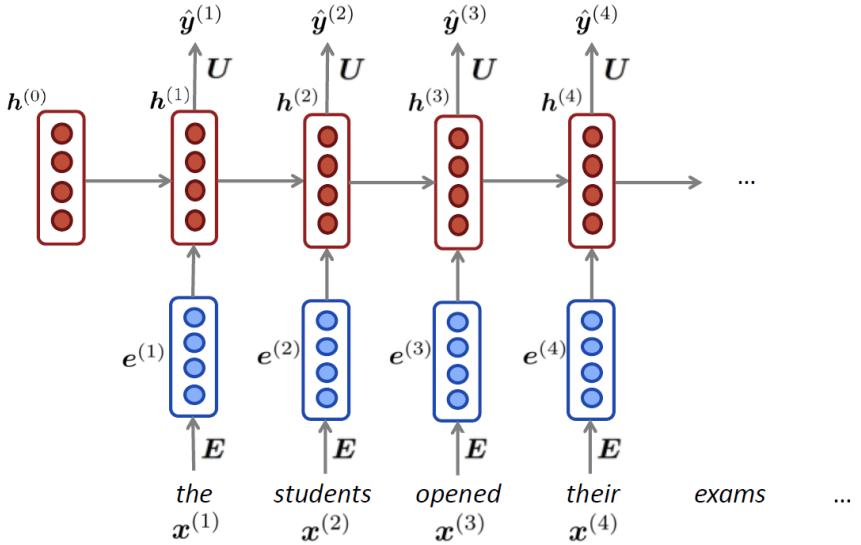
A function: $\boldsymbol{y} = RNN(\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n) \in \mathbb{R}^h$, where $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n \in \mathbb{R}^d$

Core idea: apply the same weights repeatedly at different positions!

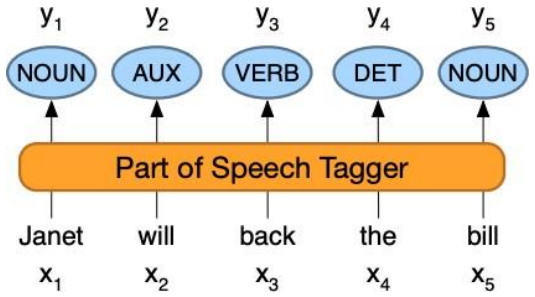
# Recurrent Neural Network (RNN)

- Effective approach for language modeling, sequence tagging, text classification

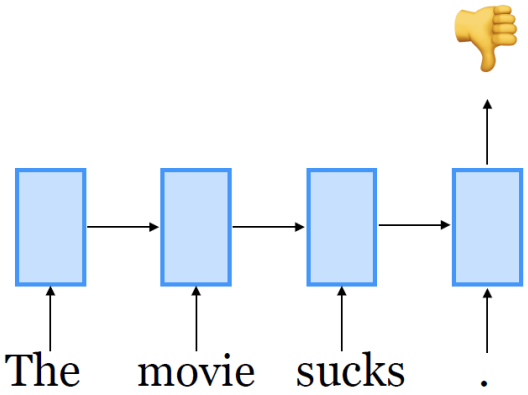
Language modeling




Sequence tagging




Text classification

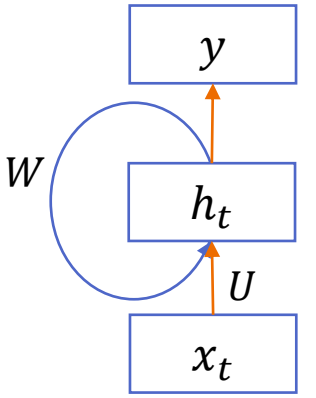


# Vanilla RNN

- Define the RNN network as a function $y = \mathrm{RNN}(x_1, x_2, \ldots, x_n)$, where $y \in R^h, x_i \in R^h$
- Define the hidden state $h_t$ as

$$h_t = f(h_{t-1}, x_t), h_t \in R^h$$

- Where $h_0$ is the initial state and can be set as 0




- For vanilla RNN,

$$h_t = g(Wh_{t-1} + Ux_t + b), h_t \in R^h$$

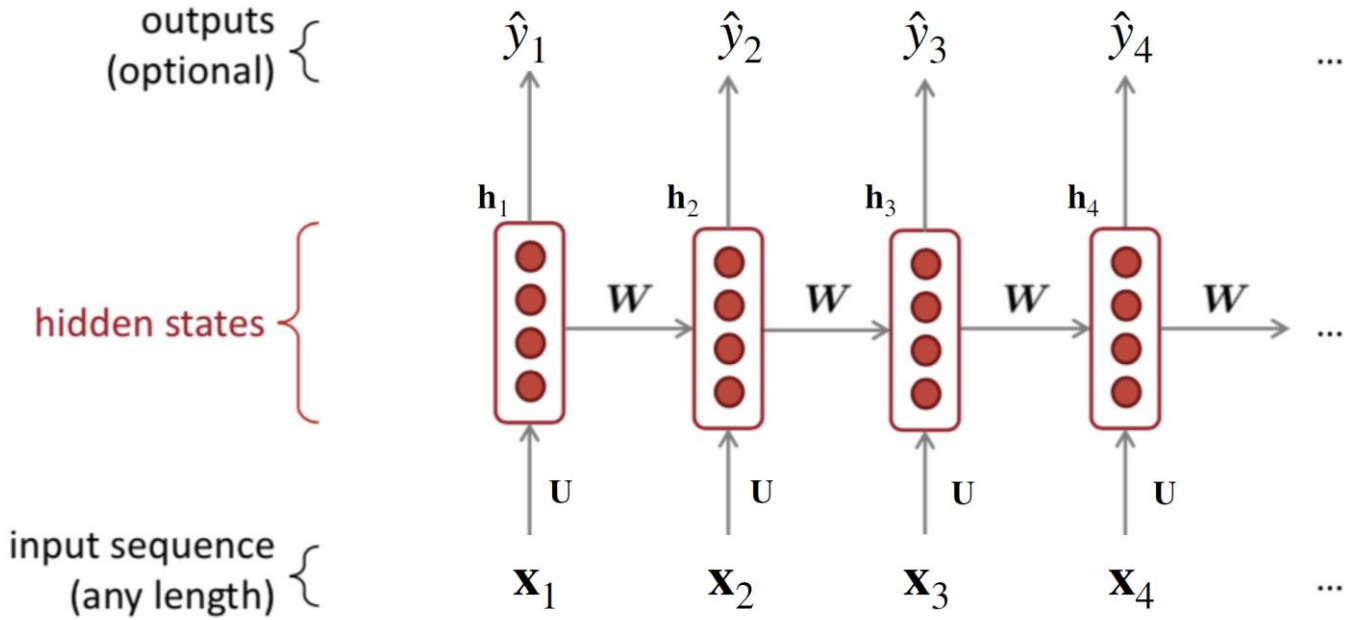
$$y = W_o h_t + b_o$$

- Where, $g(x)$ is a non-linear function, e.g., tanh, ReLU,
- $W \in R^{h \times h}, U \in R^{h \times h}, b \in R^h$

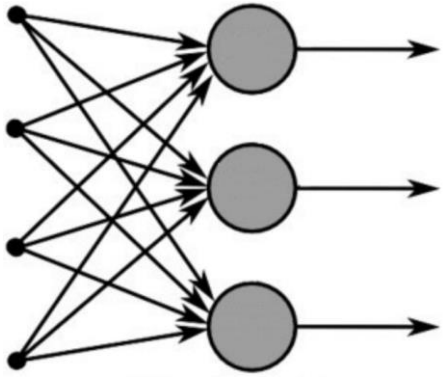
# Vanilla RNN

- Key idea: apply the same weights W, U, b repeatedly

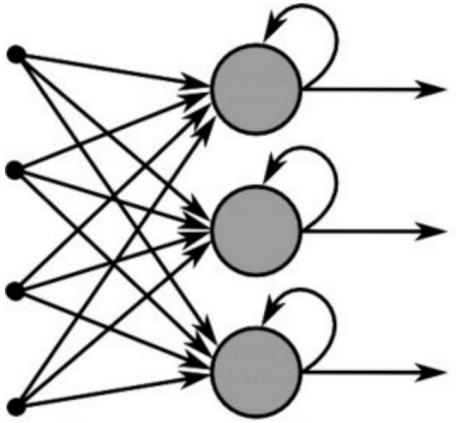
$$h_t = g(Wh_{t-1} + Ux_t + b)$$



# RNN vs Feedforward NN



Feed-Forward Neural Network



Recurrent Neural Network

$$h_1 = g(U_1 x + b_1)$$

$$h_2 = g(U_2 h_1 + b_2)$$

$$h_t = g(W h_{t-1} + U x_t + b)$$

# Vanilla RNN: Pros and Cons

- Advantages:
  - Can process any length input
  - Computation for step $t$ can (in theory) use information from many steps back
  - Model size doesn't increase for longer input context
- Disadvantages:
  - Recurrent computation is slow (can't parallelize)
  - In practice, difficult to access information from many steps back (Optimization issue)

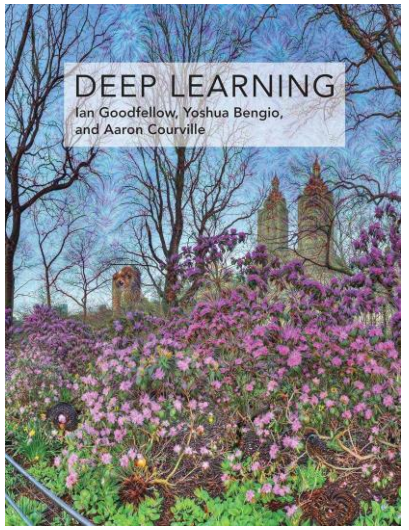
# Problems of Vanilla RNN

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Gradient exploding problem
  - Gradients become too large
  - model will become difficult to converge (unstable)
  - One solution: gradient clipping, take a step in the same direction but a smaller step
- Gradient vanishing problem
  - Gradients become too small
  - Difficult to know which direction the parameters should move to
  - Model can't capture long-term dependencies
  - Model may capture a wrong recent dependency

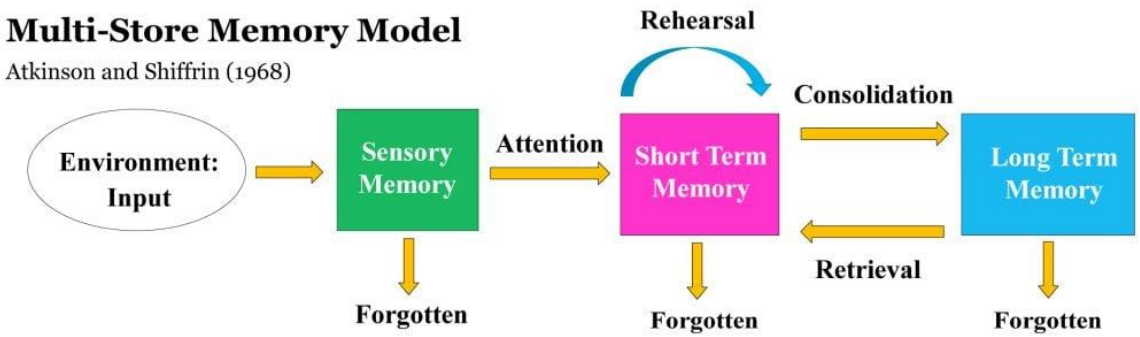
More reading about optimization problem in the Deep Learning book

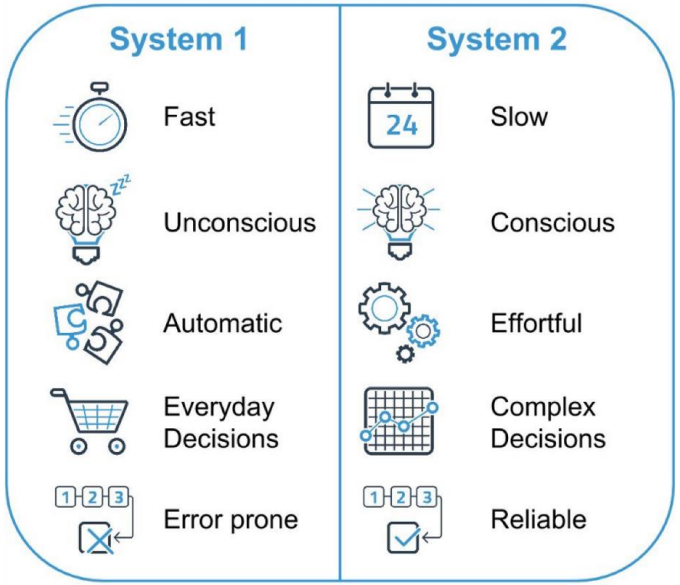
cs224n-2018-lecture9-vanishing_gradient



# RNN Variants: LSTM

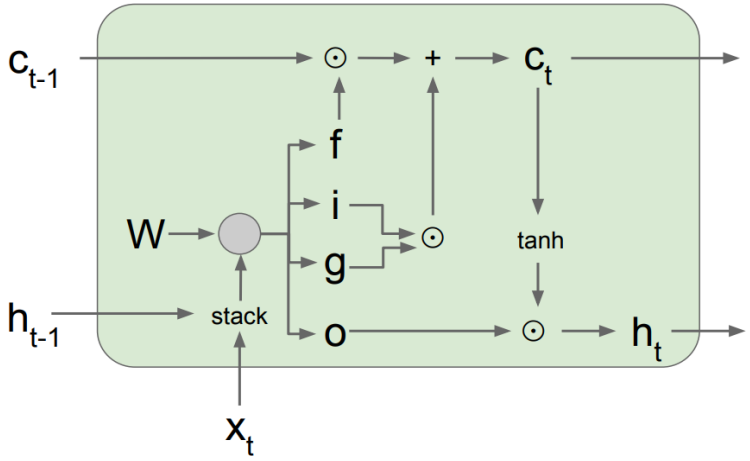
- Long Short-Term Memory RNN (LSTM)
  - Proposed by Hochreiter and Schmidhuber in 1997 as a solution to the vanishing gradients problem
- Short-term memory
  - Recall specific information about anything for a brief period
  - Only last for about 30 seconds ~ minutes
- Long-term memory
  - Last for minutes to years




System 2 deep learning: The next step toward artificial general intelligence -Benjio

# RNN Variants: LSTM




- Hidden state as $h_t$ and cell state as $c_t$
- $c_t$ stores long-term information
  - Changes very little step to step
- $h_t$ stores short-term information
  - Changes all the time

Input gate **(how much to write)**:

$$\mathbf{i}_t = \sigma(\mathbf{W}^i\mathbf{h}_{t-1} + \mathbf{U}^i\mathbf{x}_t + \mathbf{b}^i) \in \mathbb{R}^h$$

Forget gate **(how much to erase)**:

$$\mathbf{f}_t = \sigma(\mathbf{W}^f\mathbf{h}_{t-1} + \mathbf{U}^f\mathbf{x}_t + \mathbf{b}^f) \in \mathbb{R}^h$$

Output gate **(how much to reveal)**:

$$\mathbf{o}_t = \sigma(\mathbf{W}^o\mathbf{h}_{t-1} + \mathbf{U}^o\mathbf{x}_t + \mathbf{b}^{(o)}) \in \mathbb{R}^h$$

New memory cell **(what to write)**:

$$\mathbf{g}_t = \tanh(\mathbf{W}^g\mathbf{h}_{t-1} + \mathbf{U}^g\mathbf{x}_t + \mathbf{b}^g) \in \mathbb{R}^h$$

Final memory cell: $c_t = f_t \odot c_{t-1} + i_t \odot g_t$

Final hidden cell: $h_t = o_t \odot \tanh(c_t)$

Output $y = MLP(h_t)$

Understanding LSTM Networks -- colah's blog

LSTM — PyTorch 2.0 documentation

# RNN Variants: GRU

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Gated Recurrent Unit (GRU)
  - Introduced by Kyunghyun Cho et al. in 2014
- Simplified 3 gates to 2 gates: reset gate and update gate, without an explicit cell state

**Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation**

**Kyunghyun Cho**
**Bart van Merriënboer** **Caglar Gulcehre** Université de Montréal
firstname.lastname@umontreal.ca
**Dzmitry Bahdanau** Jacobs University, Germany
d.bahdanau@jacobs-university.de
**Fethi Bougares** **Holger Schwenk** Université du Maine, France
firstname.lastname@lium.univ-lemans.fr
**Yoshua Bengio** Université de Montréal, CIFAR Senior Fellow
find.me@on.the.web

GRU — PyTorch 2.0 documentation

# Gated Recurrent Unit (GRU)

- Reset gate:

$$\mathbf{r}_t = \sigma(\mathbf{W}^r \mathbf{h}_{t-1} + \mathbf{U}^r \mathbf{x}_t + \mathbf{b}^r)$$

- Update gate:

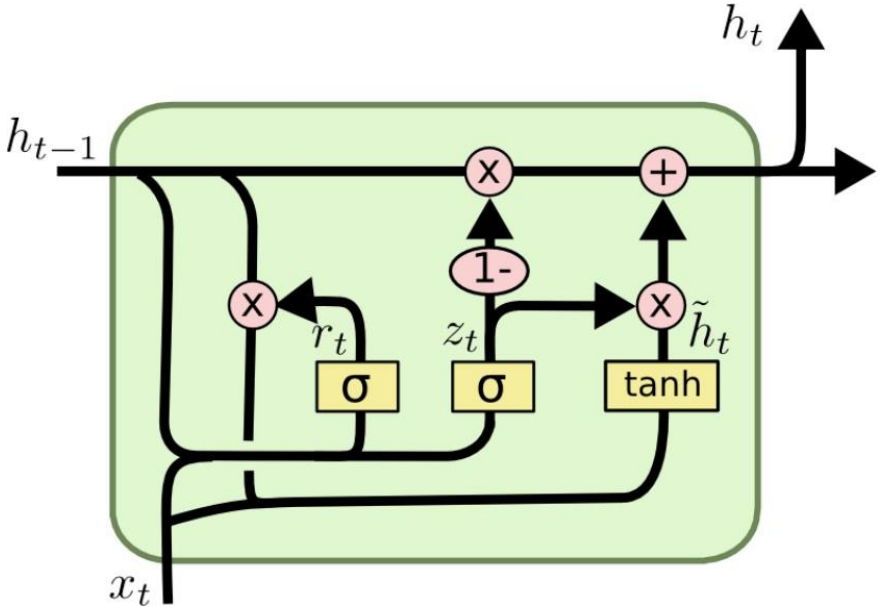
$$\mathbf{z}_t = \sigma(\mathbf{W}^z \mathbf{h}_{t-1} + \mathbf{U}^z \mathbf{x}_t + \mathbf{b}^z)$$

- New hidden state:

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{U}\mathbf{x}_t + \mathbf{b})$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t$$

merge input and forget gate!



# RNN Language Model

- Language modeling

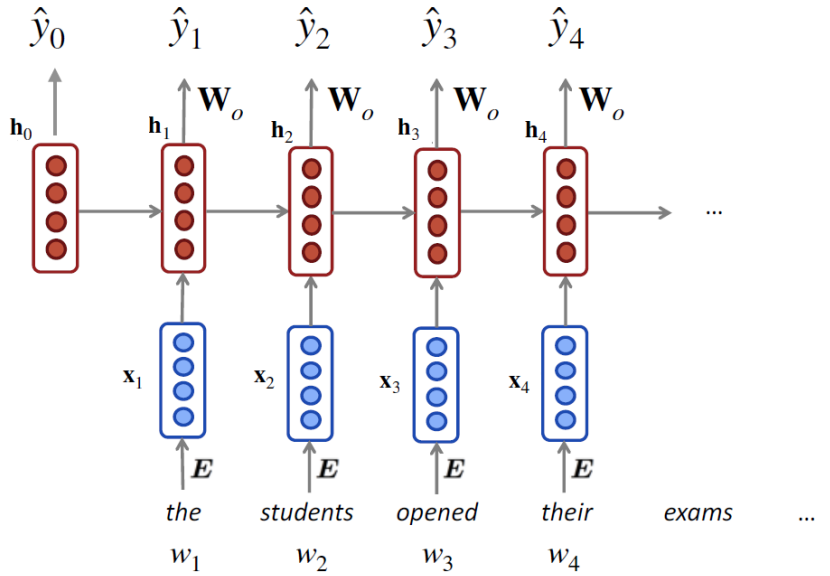
$$P(w_1, w_2, \ldots, w_n) = P(w_1) \times P(w_2 \mid w_1) \times P(w_3 \mid w_1, w_2) \times \ldots \times P(w_n \mid w_1, w_2, \ldots, w_{n-1})$$

$$= P(w_1 \mid \mathbf{h}_0) \times P(w_2 \mid \mathbf{h}_1) \times P(w_3 \mid \mathbf{h}_2) \times \ldots \times P(w_n \mid \mathbf{h}_{n-1})$$

- We select the output token by

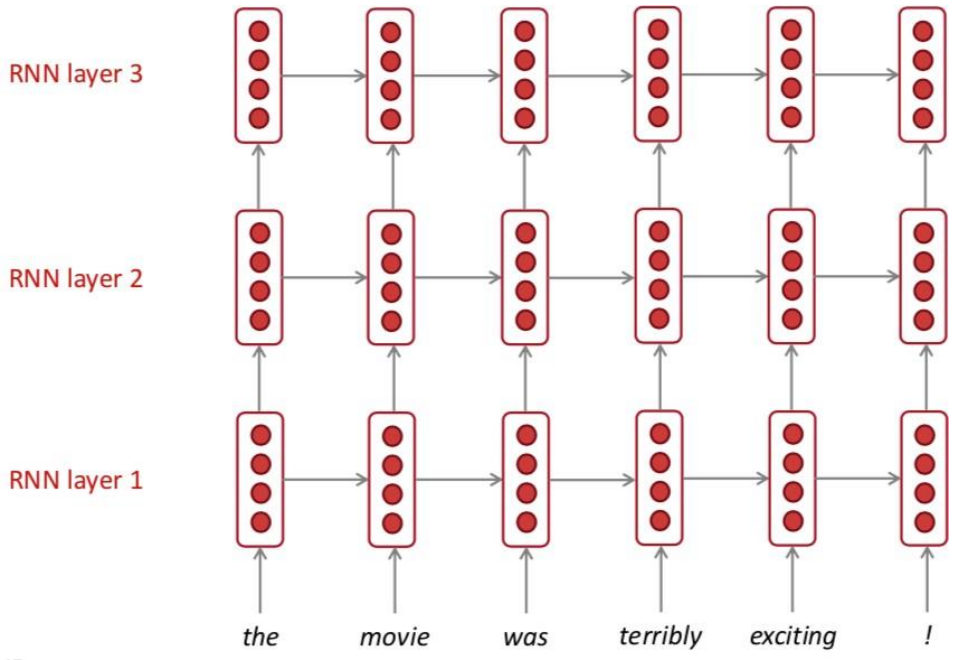
$$\hat{y} = \text{softmax}(W_o h_t + b_o) \in R^{|V|}$$

- Weight tying
  - Input word embedding $E$
  - Output embedding $W_o$
  - Set $E = W_o \in R^{|V| \times d}$



# Multi-layer RNNs

- In practice, using 2 to 4 layers is common (usually better than 1 layer)
- Transformer networks can be up to 24 layers with lots of skip-connections



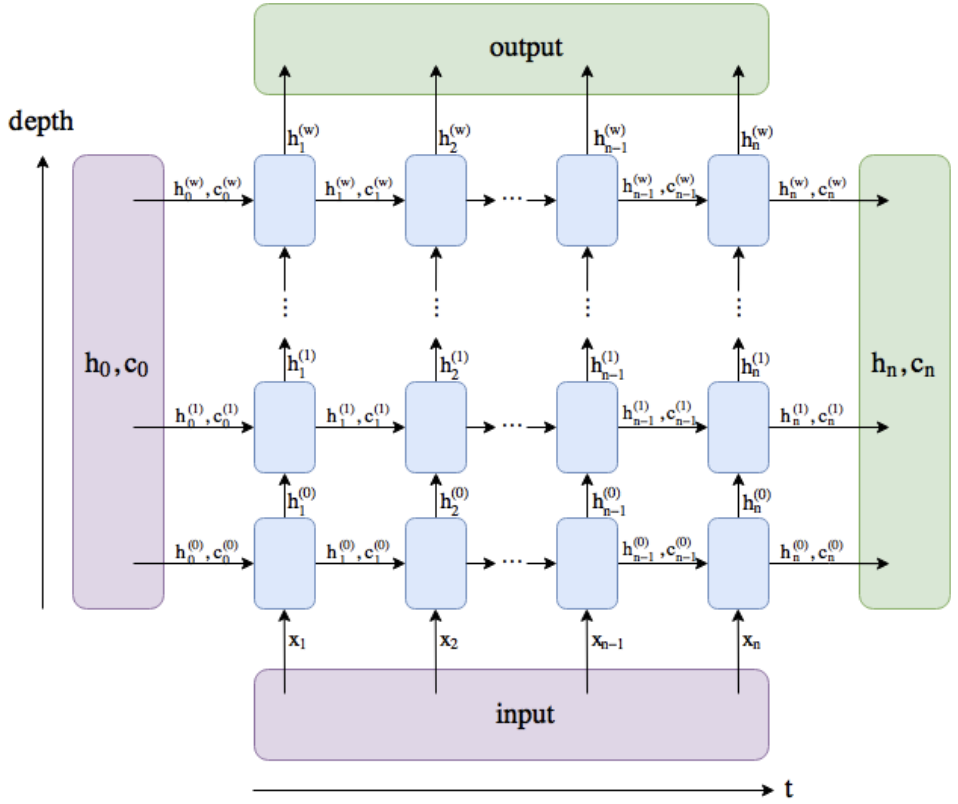

The hidden states from RNN layer are the inputs to RNN layer

# Multi-layer RNNs

- Pytorch LSTM documentation

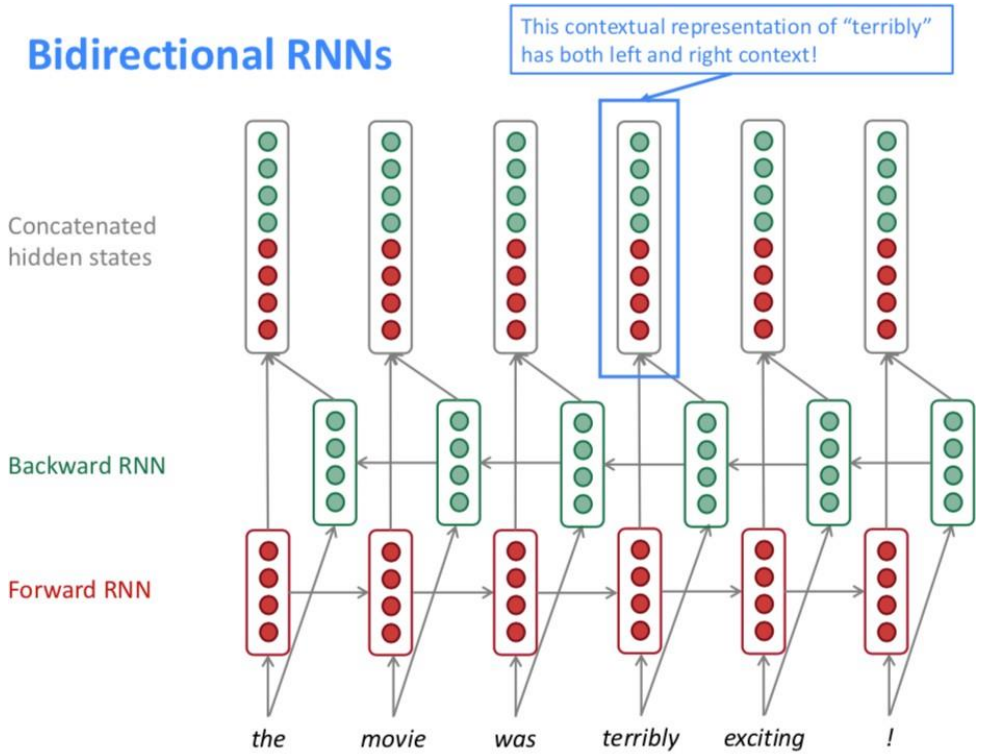
Examples:

```
>>> rnn = nn.LSTM(10, 20, 2)
>>> input = torch.randn(5, 3, 10)
>>> h0 = torch.randn(2, 3, 20)
>>> c0 = torch.randn(2, 3, 20)
>>> output, (hn, cn) = rnn(input, (h0, c0))
```



# Bidirectional RNN

- Expect the hidden state contains information from both side
- For RNN language modeling



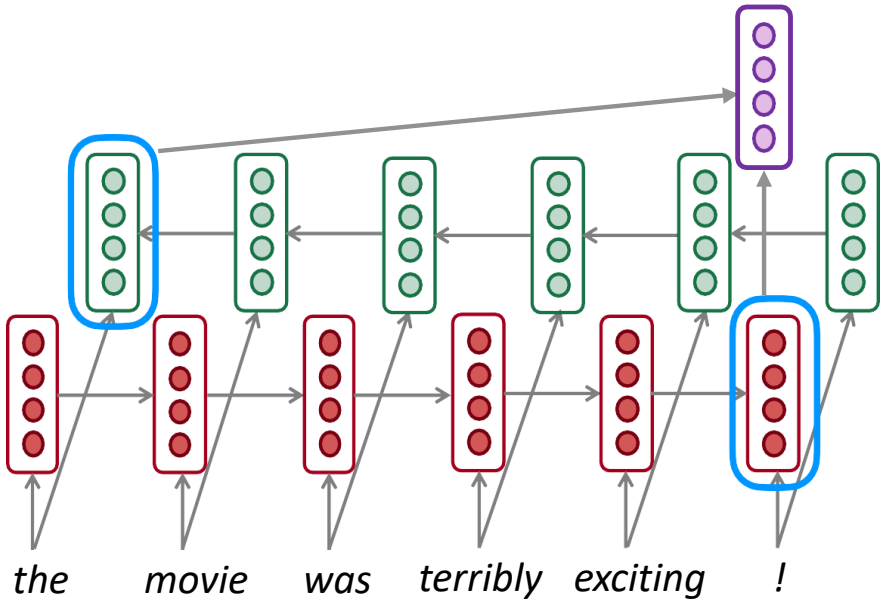

$$\overrightarrow{\mathbf{h}}_t = f_1(\overrightarrow{\mathbf{h}}_{t-1}, \mathbf{x}_t), t = 1,2,\ldots n$$

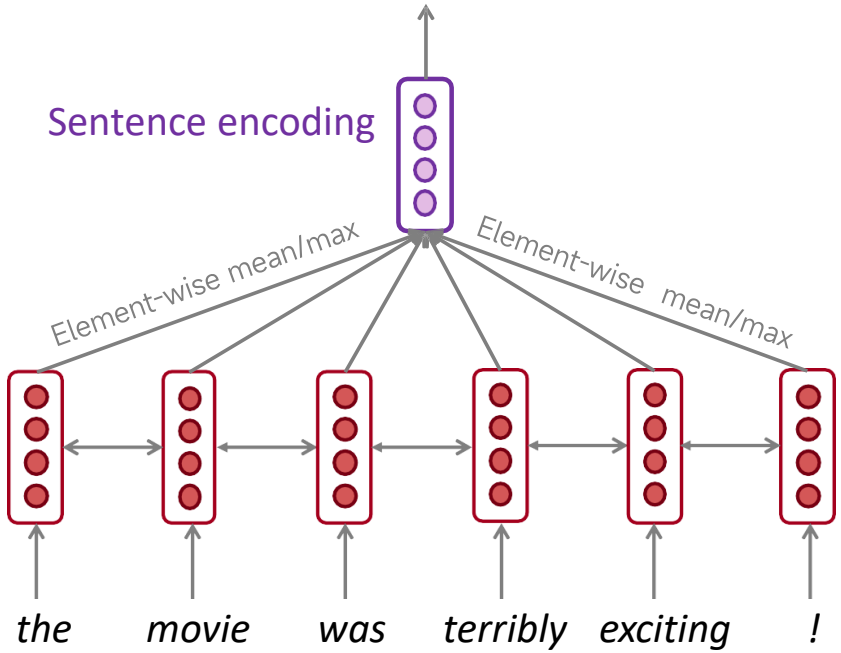
$$\overleftarrow{\mathbf{h}}_t = f_2(\overleftarrow{\mathbf{h}}_{t+1}, \mathbf{x}_t), t = n, n-1,\ldots 1$$

$$\mathbf{h}_t = [\overleftarrow{\mathbf{h}}_t, \overrightarrow{\mathbf{h}}_t] \in \mathbb{R}^{2h}$$

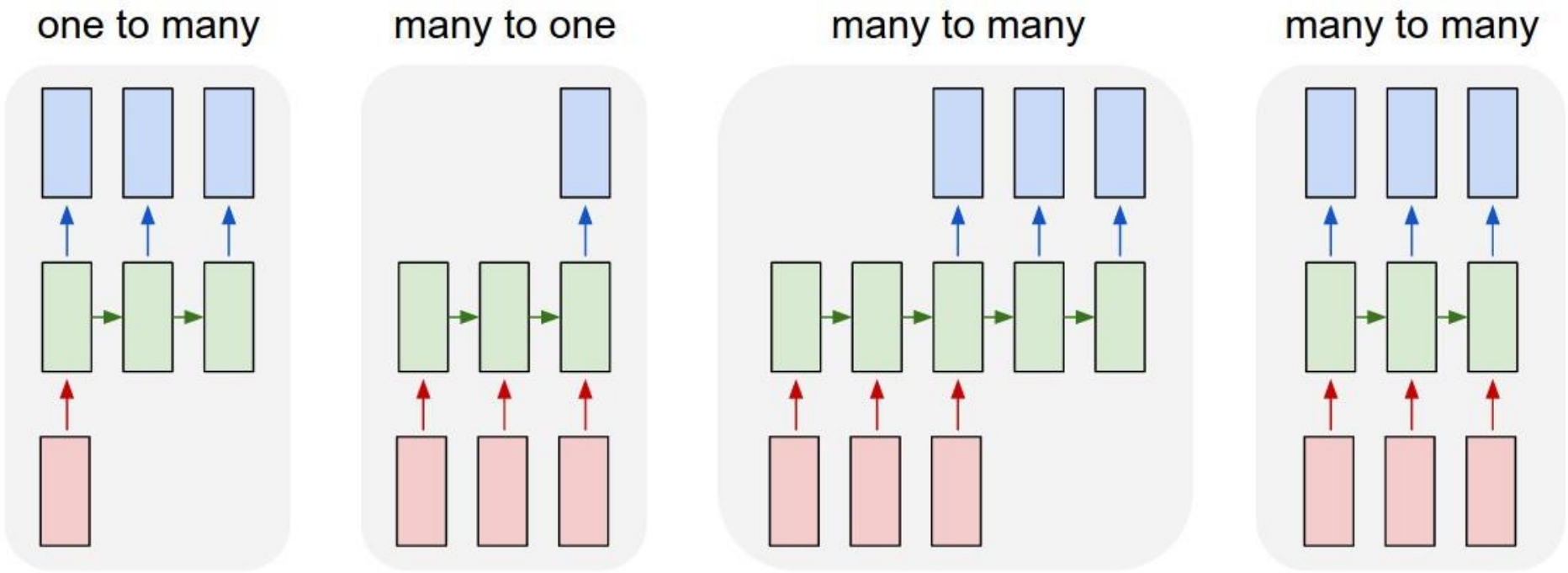
# Bidirectional RNN

- For text classification tasks:
  - Concatenate the last hidden vectors in two directions
  - Or take the mean/max over all the hidden vectors



# Different Types of Sequence Modeling Tasks




[Translation with a Sequence to Sequence Network and Attention — PyTorch Tutorials]

# Further Reading

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Learning Word Embedding | Lil'Log
- On word embeddings - Part 1
- A Recipe for Training Neural Networks
- 深度学习科研，如何高效进行代码和实验管理？ - 知乎

# Thank you