



CSE 176 Introduction to Machine Learning

Lecture 13: RNN and Transformer

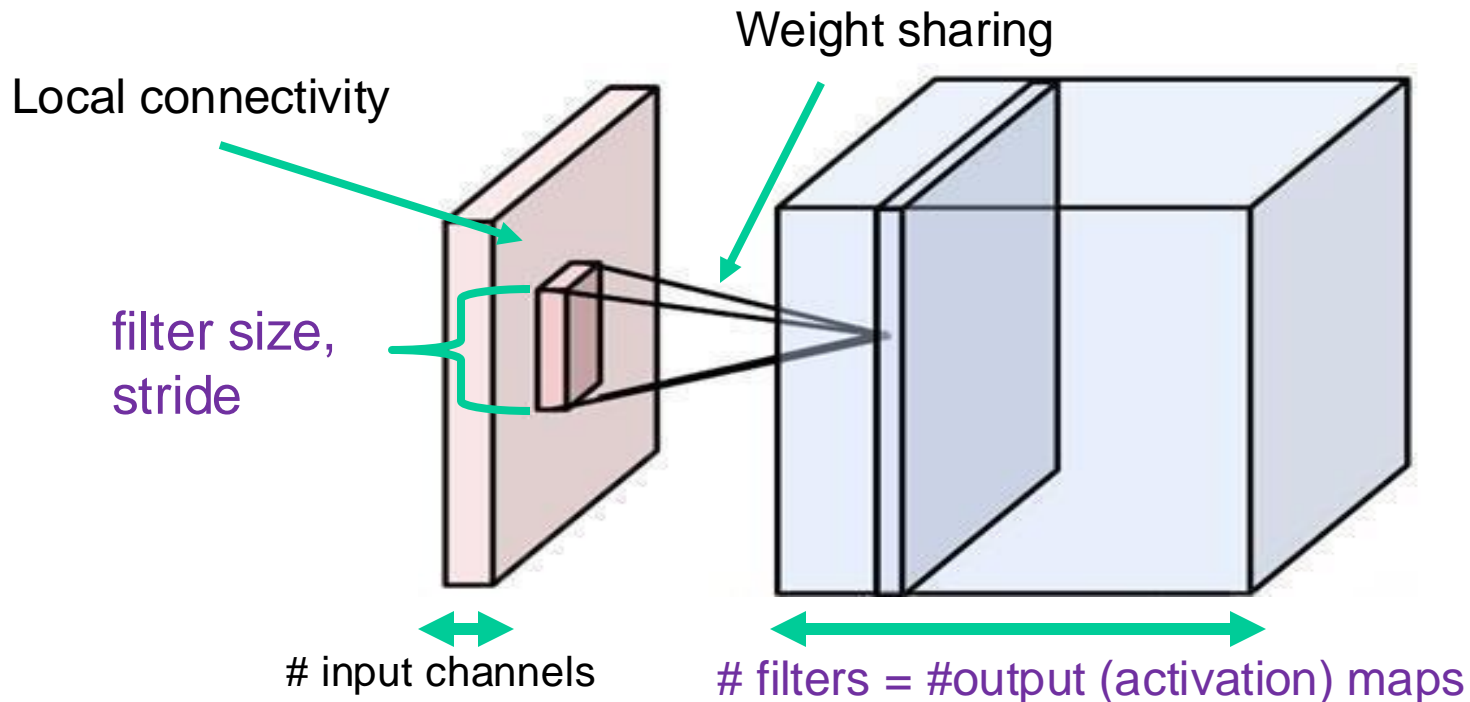
From last lecture: Convolutional Layer

Local connectivity

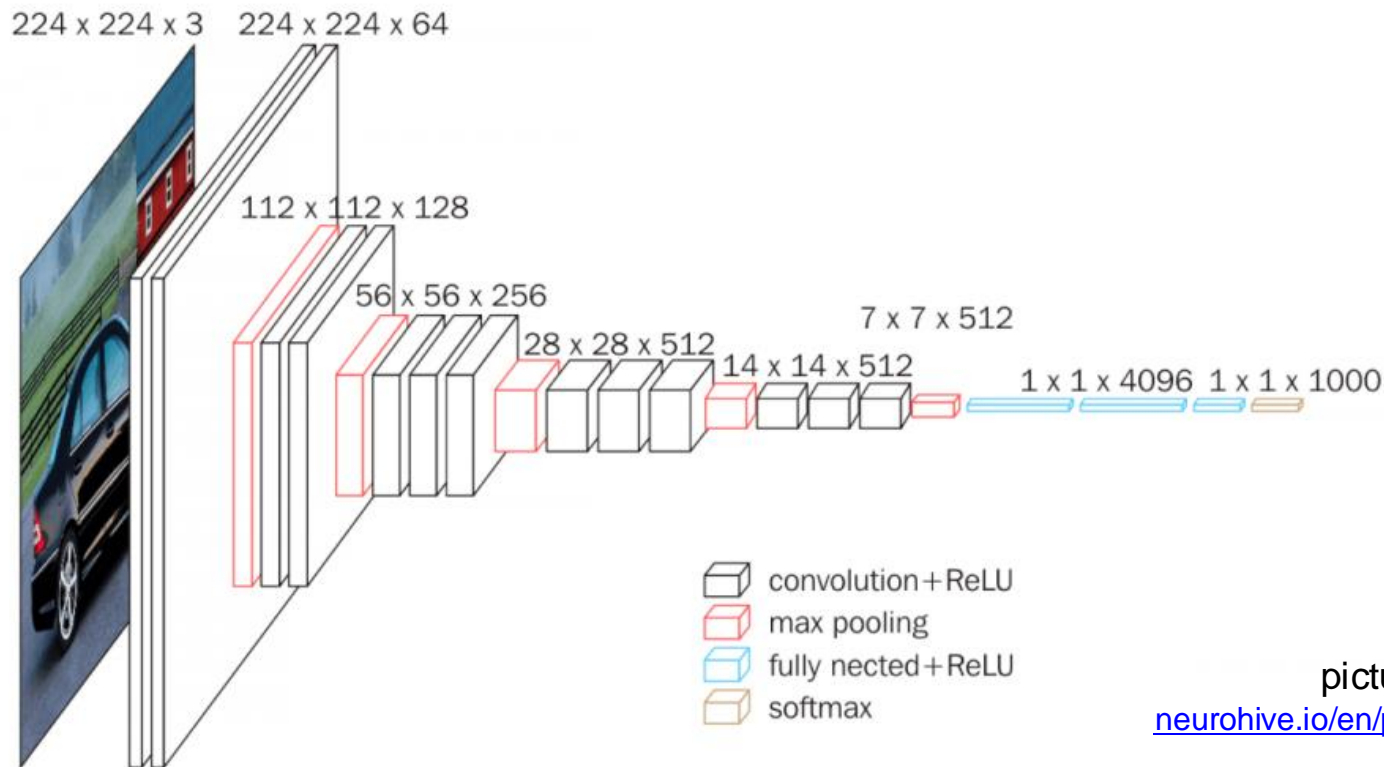
Weight sharing

Handling multiple input/output channels

Transforms 3D tensor into 3D tensor

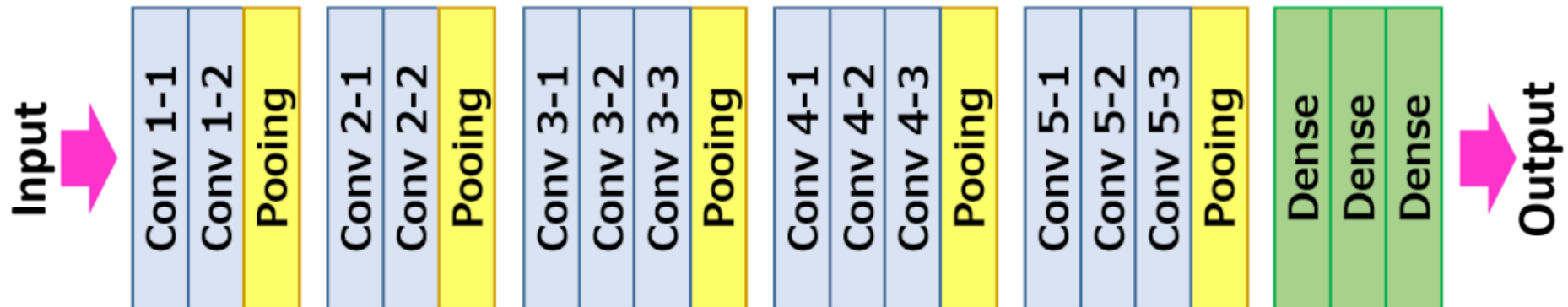


From last lecture: CNN e.g. VGG -16



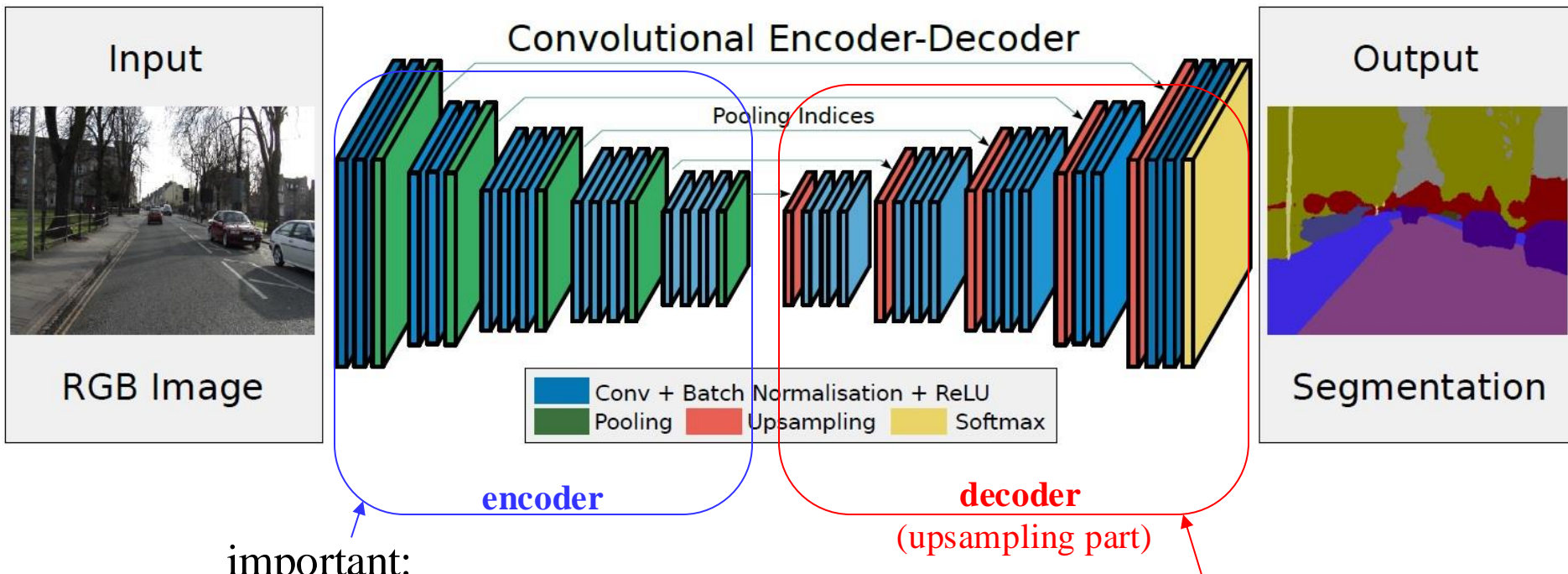
picture credits

neurohive.io/en/popular-networks/vgg16/



Recap: *Encoder/Decoder*

Segnet: A deep convolutional encoder-decoder architecture for image segmentation
Badrinarayanan, Kendall, Cipolla – TPAMI 2017



important:

encoder convolutional layers are typically pre-trained on *image net*

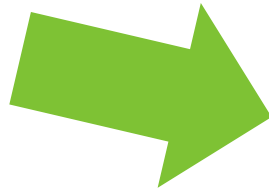
decoder upsamples encoder-generated features

Deconvolution: Example

Note: this result is equivalent to **Bilinear Interpolation**

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Input Image



Kernel		
0.25	0.5	0.25
0.5	1	0.5
0.25	0.5	0.25

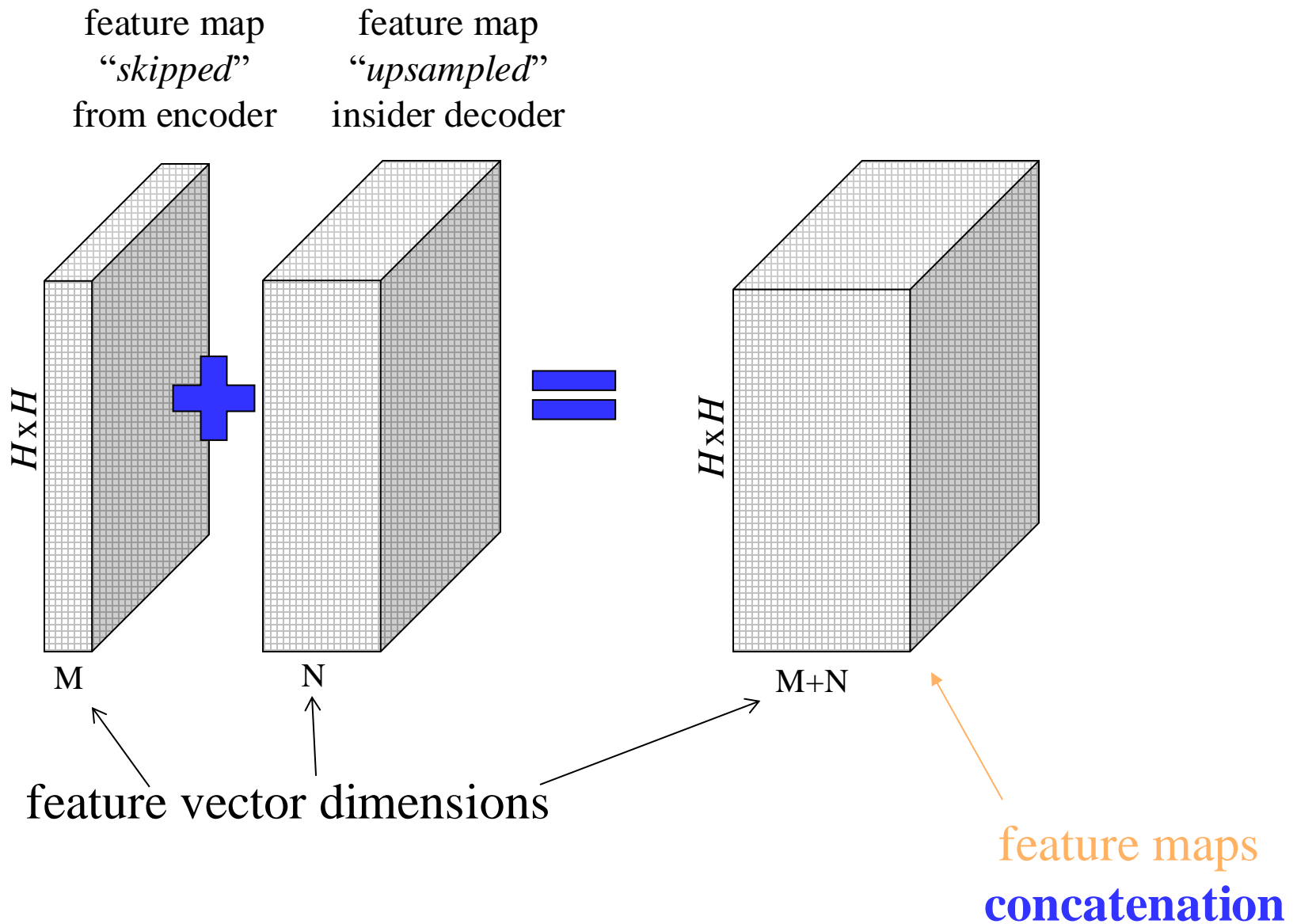
kernel=3x3
stride=2
padding=1

Output Image

0	0	0.25	0.5	0.75	1	1.25	1.5	0.75
0	0	0.5	1	1.5	2	2.5	3	1.5
4	2	2.5	3	3.5	4	4.5	5	2.5
2	4	4.5	5	5.5	6	6.5	7	3.5
3	6	6.5	7	7.5	8	8.5	9	4.5
4	8	8.5	9	9.5	10	10.5	11	5.5
5	10	10.5	11	11.5	12	12.5	13	6.5
6	12	12.5	13	13.5	14	14.5	15	7.5
3	6	6.25	6.5	6.75	7	7.25	7.5	3.75

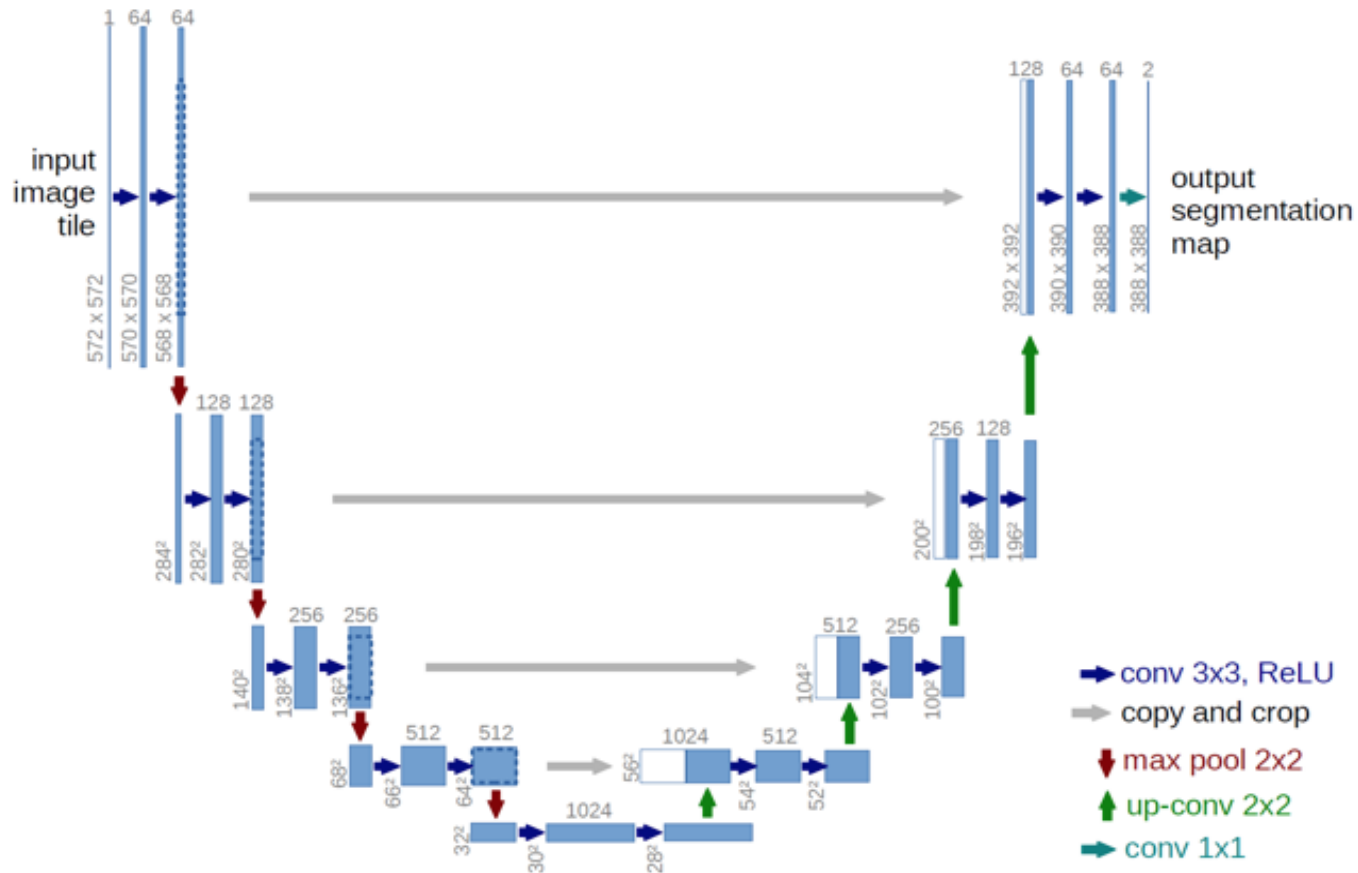
Bilinear Interpolation is a special case of deconvolution.

Skip connections: concatenation



U-net: expanding decoder with symmetry

and many skip connections



Let's use neural networks for
natural language processing!



Part I: NLP and Language Modelling

ChatGPT (Generative Pretrained Transformer)



You

What are top 10 things to do in Merced?



ChatGPT

Merced, California, offers a range of activities and attractions for visitors to enjoy. Here are the top 10 things to do in Merced:

1. **Merced National Wildlife Refuge:** Explore the refuge and observe diverse bird species, including migratory birds, waterfowl, and endangered species like the Aleutian Canada goose.
2. **Castle Air Museum:** Discover the history of aviation at this museum, which features a vast collection of aircraft, including military planes, helicopters, and memorabilia.
3. **Yosemite National Park:** While not directly in Merced, the park is nearby and offers breathtaking scenery, hiking trails, waterfalls, and opportunities for camping and wildlife viewing.
4. **Applegate Park and Zoo:** Enjoy a day outdoors at this park, which features playgrounds, picnic areas, a small zoo with animals like monkeys, birds, and reptiles, and a miniature train ride.
5. **Downtown Merced:** Explore the historic downtown area with its charming shops, cafes, restaurants, and galleries. Don't miss the Merced County Courthouse Museum to learn about the region's history.
6. **Merced County Fairgrounds:** Check out events and festivals held at the fairgrounds throughout the year, including the Merced County Fair, concerts, rodeos, and more.
7. **Lake Yosemite:** Spend a relaxing day by the lake, where you can enjoy fishing, boating, kayaking, picnicking, and walking along the scenic trails.

What is Natural Language Processing?

- ❑ Natural language processing is the set of methods for making human language accessible to computers.
(Jacob Eisenstein)
- ❑ Natural language processing is the field at the intersection of Computer science (Artificial intelligence) and linguistics.
(Christopher Manning)

NLP application: Machine translation

爷爷心疼孙女让娃睡懒觉 没想到引发了家庭矛盾

2020-08-25 08:06:03 来源: 钱江晚报

70岁的钟大伯(化名)陷入了“暑期焦虑”:这个暑假,他每周都要接送孙女上下培训班。高温、酷暑,每天回来,都像脚踩棉花般没力气。

除了身体上的不适,还有精神上的紧张。

觉得儿子儿媳给孩子报班太多,钟大伯还和他们产生了冲突:“大热天的,大人孩子都遭罪。”

这段时间,钟大伯因为容易激动发火,失眠,胃口差,血压一直不稳定,来到了浙江省人民医院精神卫生科就诊。

Google Translate

Grandpa feels sorry for his granddaughter and let the baby sleep in

2020-08-25 08:06:03 Source: Qianjiang Evening News

Uncle Zhong (a pseudonym), 70, fell into "summer anxiety": This summer, he would shuttle his granddaughter to and from training classes every week. With high temperatures and scorching heat, every day I come back, I feel as weak as stepping on cotton.

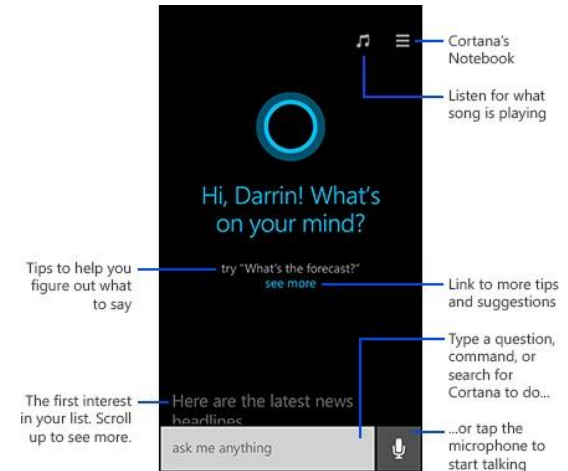
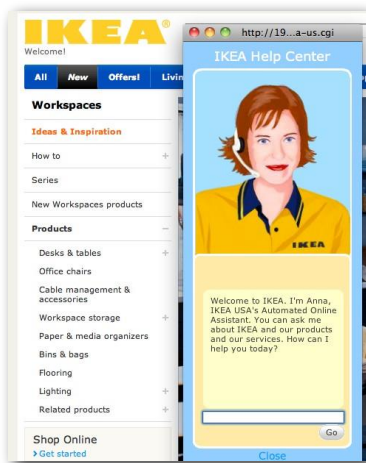
In addition to physical discomfort, there is also mental tension.

Feeling that his son and daughter-in-law were reporting too much for their children, Uncle Zhong also had a conflict with them: "It's a hot day, adults and children suffer."

During this period of time, Uncle Zhong came to the Mental Health Department of Zhejiang Provincial People's Hospital because he was prone to get angry, insomnia, poor appetite, and unstable blood pressure.

http://education.news.cn/2020-08/25/c_1210768533.htm

NLP application: Dialog systems, chatbots, assistants

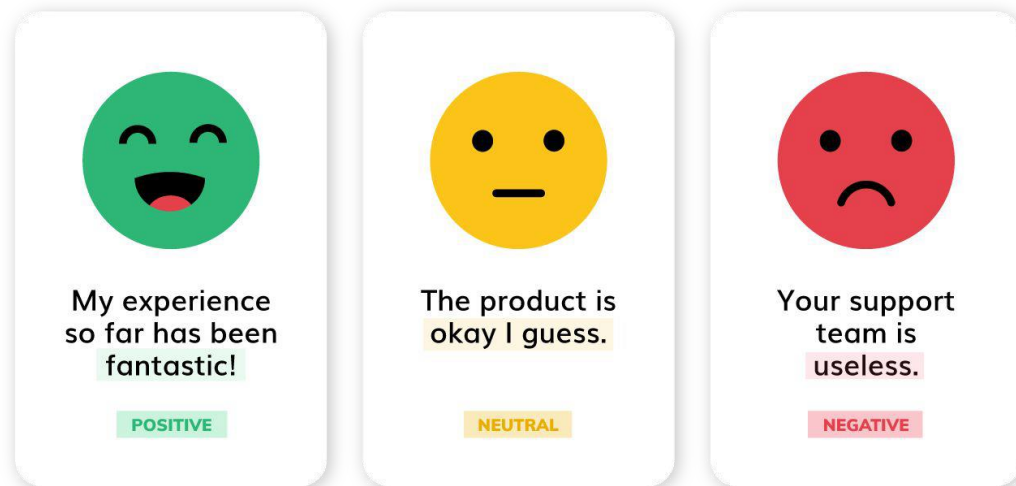


ChatGPT		
Examples	Capabilities	Limitations
"Explain quantum computing in simple terms" →	Remembers what user said earlier in the conversation	May occasionally generate incorrect information
"Got any creative ideas for a 10 year old's birthday?" →	Allows user to provide follow-up corrections	May occasionally produce harmful instructions or biased content
"How do I make an HTTP request in Javascript?" →	Trained to decline inappropriate requests	Limited knowledge of world and events after 2021



NLP application: Sentiment analysis

- ❑ Determine the meaning behind is positive, negative, or neutral



English Vocabulary

How large is the **vocabulary** of English (or any other language)?

Vocabulary size = the number of distinct word types

If you count words in text, you will find that...

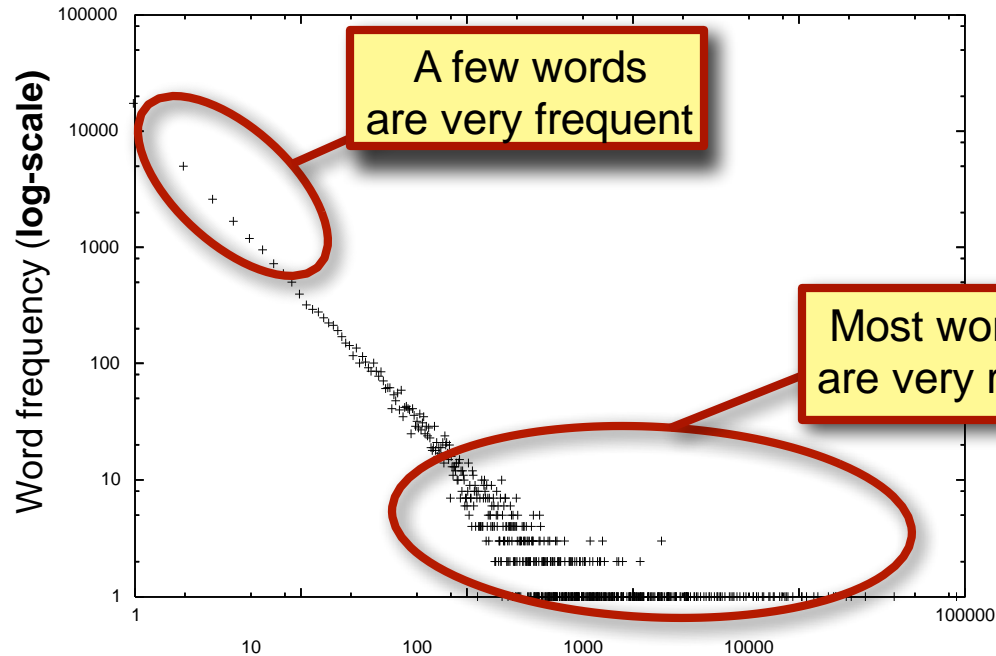
...a **few words** are **very frequent**
(the, be, to, of, and, a, in, that,...)

... **most words** are **very rare**.

... even if you've read a lot of text,
you will keep finding **words you haven't seen before**.

Word frequency: the number of occurrences of a word type in a text
(or in a collection of texts)

Long-tailed word distribution



English words, sorted by frequency (**log-scale**)

$w_1 = \textit{the}$, $w_2 = \textit{to}$, ..., $w_{5346} = \textit{computer}$, ...

Why do we need language models?

Many NLP tasks require **natural language output**:

- Machine translation**: return text in the target language
- Speech recognition**: return a transcript of what was spoken
- Natural language generation**: return natural language text
- Spell-checking**: return corrected spelling of input

Language models define **probability distributions over (natural language) strings or sentences**.

- We can use a language model to **generate strings**
- We can use a language model to **score/rank candidate strings** so that we can choose the best (i.e. most likely) one:
if $P_{LM}(A) > P_{LM}(B)$, return A, not B

Hmmm, but...

- ... what does it mean for a language model to “*define a probability distribution*”?
- ... *why* would we want to define probability distributions over languages?
- ... how can we construct a language model such that it *actually* defines a probability distribution?

You should be able to answer these questions after this lecture

Key concepts

N-gram language models

- Independence assumptions

- Getting from n-grams to a distribution over a language

- Relative frequency (maximum likelihood) estimation

- Smoothing

Now let's look at natural language

Text as a bag of words

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?'

$$P(\text{of}) = 3/66$$

$$P(\text{Alice}) = 2/66$$

$$P(\text{was}) = 2/66$$

$$P(\text{to}) = 2/66$$

$$P(\text{her}) = 2/66$$

$$P(\text{sister}) = 2/66$$

$$P(,) = 4/66$$

$$P(') = 4/66$$

Sampling with replacement

- A sampled sequence of words
 - beginning by, very Alice but was and? reading no tired of to into sitting sister the, bank, and thought of without her nothing: having conversations Alice once do or on she it get the book her had peeped was conversation it pictures or sister in, 'what is the use had twice of a book 'pictures or' to

$$P(\text{of}) = 3/66$$

$$P(\text{to}) = 2/66$$

$$P(,) = 4/66$$

$$P(\text{Alice}) = 2/66$$

$$P(\text{her}) = 2/66$$

$$P(') = 4/66$$

$$P(\text{was}) = 2/66$$

$$P(\text{sister}) = 2/66$$

In this model, $P(\text{English sentence}) = P(\text{word salad})$

Language modeling with N-grams

A **language model** over a vocabulary V assigns probabilities to strings drawn from V^* .

How do we compute the **probability of a string** $w^{(1)} \dots w^{(i)}$?

Recall the **chain rule**:

$$P(w^{(1)} \dots w^{(i)}) = P(w^{(1)}) \cdot P(w^{(2)} | w^{(1)}) \cdot \dots \cdot P(w^{(i)} | w^{(i-1)}, \dots, w^{(1)})$$

An **n-gram** language model assumes each word $w^{(i)}$ depends only on the **last n-1 words** $w^{(i-1)}, \dots, w^{(i-(n+1))}$

$$P_{ngram}(w^{(1)} \dots w^{(i)}) = P(w^{(1)}) \cdot P(w^{(2)} | w^{(1)}) \cdot \dots \cdot P(w^{(i)} | w^{(i-1)}, \dots, w^{(i-(n+1))})$$

N-gram models

N-gram models *assume* each word (event) depends only on the previous $n-1$ words (events):

$$\text{Unigram model: } P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^N P(w^{(i)})$$

$$\text{Bigram model: } P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^N P(w^{(i)} | w^{(i-1)})$$

$$\text{Trigram model: } P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^N P(w^{(i)} | w^{(i-1)}, w^{(i-2)})$$

Independence assumptions where the n -th event in a sequence depends only on the last $n-1$ events are called **Markov assumptions (of order $n-1$)**.

How many parameters do n-gram models have?

Given a vocabulary V of $|V|$ word types: so, for $|V| = 10^4$:

Unigram model: $|V|$ parameters

10^4 parameters

(one distribution $P(w^{(i)})$ with $|V|$ outcomes
[each $w \in V$ is one outcome])

Bigram model: $|V|^2$ parameters

10^8 parameters

Trigram model: $|V|^3$ parameters

10^{12} parameters

A bigram model for Alice

• Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?'

$$P(w^{(i)} = \text{of} \mid w^{(i-1)} = \text{tired}) = 1$$

$$P(w^{(i)} = \text{of} \mid w^{(i-1)} = \text{use}) = 1$$

$$P(w^{(i)} = \text{sister} \mid w^{(i-1)} = \text{her}) = 1$$

$$P(w^{(i)} = \text{beginning} \mid w^{(i-1)} = \text{was}) = 1/2$$

$$P(w^{(i)} = \text{reading} \mid w^{(i-1)} = \text{was}) = 1/2$$

$$P(w^{(i)} = \text{bank} \mid w^{(i-1)} = \text{the}) = 1/3$$

$$P(w^{(i)} = \text{book} \mid w^{(i-1)} = \text{the}) = 1/3$$

$$P(w^{(i)} = \text{use} \mid w^{(i-1)} = \text{the}) = 1/3$$

Using a bigram model for Alice

English

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?'

Word Salad

beginning by, very Alice but was and?
reading no tired of to into sitting
sister the, bank, and thought of without
her nothing: having conversations Alice
once do or on she it get the book her had
peeped was conversation it pictures or
sister in, 'what is the use had twice of
a book 'pictures or' to

Now, $P(\text{English}) \gg P(\text{word salad})$

$$P(w^{(i)} = \text{of} \mid w^{(i-1)} = \text{tired}) = 1$$

$$P(w^{(i)} = \text{of} \mid w^{(i-1)} = \text{use}) = 1$$

$$P(w^{(i)} = \text{sister} \mid w^{(i-1)} = \text{her}) = 1$$

$$P(w^{(i)} = \text{beginning} \mid w^{(i-1)} = \text{was}) = 1/2$$

$$P(w^{(i)} = \text{reading} \mid w^{(i-1)} = \text{was}) = 1/2$$

$$P(w^{(i)} = \text{bank} \mid w^{(i-1)} = \text{the}) = 1/3$$

$$P(w^{(i)} = \text{book} \mid w^{(i-1)} = \text{the}) = 1/3$$

$$P(w^{(i)} = \text{use} \mid w^{(i-1)} = \text{the}) = 1/3$$

From n-gram probabilities to language models with EOS

Think of a language model as a **stochastic process**:

- At each time step, randomly pick one more word.
- **Stop** generating more words when the word you pick is a special **end-of-sentence (EOS)** token.

To be able to pick the EOS token, we have to **modify our training data** so that **each sentence ends in EOS**.

This means our vocabulary is now $V^{\text{EOS}} = V \cup \{\text{EOS}\}$

We then get an **actual language model**,
i.e. a distribution over **strings of any length**

Technically, this is only true because $P(\text{EOS} \mid \dots)$ will be high enough that we are always guaranteed to stop after having generated a finite number of words

A leaky or inconsistent language model would have $P(L) < 1$. That could happen if EOS had a very small probability (but doesn't really happen in practice).

Learning (estimating) a language model

Where do we get the **parameters of our model** (its actual probabilities) from?

$$P(w^{(i)} = \text{'the'} / w^{(i-1)} = \text{'on'}) = ???$$

We need (a large amount of) text as **training data** to estimate the parameters of a language model.

The most basic parameter estimation technique:
relative frequency estimation (frequency = counts)

$$P(w^{(i)} = \text{'the'} / w^{(i-1)} = \text{'on'}) = C(\text{'on the'}) / C(\text{'on'})$$

Also called **Maximum Likelihood Estimation (MLE)**

$C(\text{'on the'})$ [or $f(\text{'on the'})$ for frequency]:

How often does 'on the' appear in the training data?

$$\text{NB: } C(\text{'on'}) = \sum_{w \in V} C(\text{'on'} w)$$

How do we use language models?

Independently of any application, we could use a language model as a *random sentence generator* (we sample sentences according to their language model probability)

We can use a language model as a *sentence ranker*.

We prefer output sentences S_{Out} that have a higher language model probability. We can use a language model $P(S_{\text{Out}})$ to *score and rank these different candidate output sentences*, e.g. as follows:

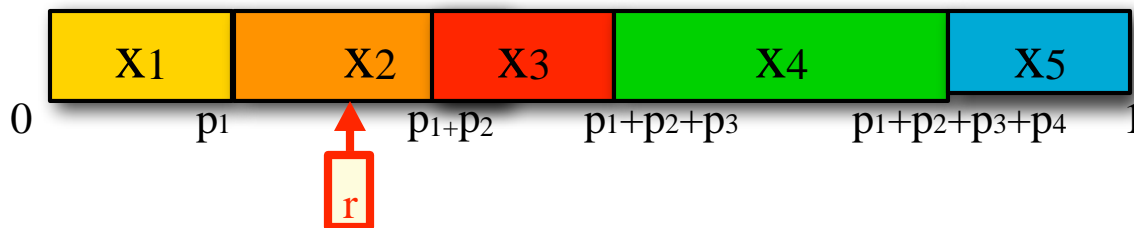
$$\operatorname{argmax}_{S_{\text{Out}}} P(S_{\text{Out}} \mid \text{Input}) = \operatorname{argmax}_{S_{\text{Out}}} P(\text{Input} \mid S_{\text{Out}})P(S_{\text{Out}})$$

Generating from a distribution

How do you generate text from an n -gram model?

That is, how do you sample from a distribution $P(X | Y=y)$?

- Assume X has N possible outcomes (values): $\{x_1, \dots, x_N\}$ and $P(X=x_i | Y=y) = p_i$
- Divide the interval $[0,1]$ into N smaller intervals according to the probabilities of the outcomes
- Generate a random number r between 0 and 1.
- Return the x_i whose interval the number is in.



Generating the Wall Street Journal

unigram: Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives

bigram: Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her

trigram: They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Generating Shakespeare

Unigram	<ul style="list-style-type: none"> • To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have • Every enter now severally so, let • Hill he late speaks; or! a more to leg less first you enter • Are where exeunt and sighs have rise excellency took of.. Sleep knave we. near; vile like
Bigram	<ul style="list-style-type: none"> • What means, sir. I confess she? then all sorts, he is trim, captain. • Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. • What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman? • Enter Menenius, if it so many good direction found'st thou art a strong upon command of fear not a liberal largess given away, Falstaff! Exeunt
Trigram	<ul style="list-style-type: none"> • Sweet prince, Falstaff shall die. Harry of Monmouth's grave. • This shall forbid it should be branded, if renown made it empty. • Indeed the duke; and had a very good friend. • Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.
Quadrigram	<ul style="list-style-type: none"> • King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; • Will you not tell me who I am? • It cannot be but so. • Indeed the short and the long. Marry, 'tis a noble Lepidus.

What have we covered so far?

We have covered a broad overview of some basic techniques in NLP:

- N-gram language models

Let's create a (much better) neural language model!

Our first neural net for NLP:

A neural n -gram model

Given a fixed-size vocabulary V , an n -gram model predicts the probability of the n -th word following the preceding $n-1$ words:

$$P(w^{(i)} | w^{(i-1)}, w^{(i-2)}, \dots, w^{i-(n-1)})$$

How can we model this with a neural net?

- **Input layer:** concatenate $n-1$ word vectors
- **Output layer:** a softmax over $|V|$ units

An n-gram model $P(w \mid w_1 \dots w_k)$ as a feedforward net (**naively**)

Assumptions:

The **vocabulary** V contains V types (incl. UNK, BOS, EOS)

We want to condition each word on k preceding words

Our (naive) model:

— [Naive]

Each **input word** $w_i \in V$ is a **V -dimensional one-hot vector** $v(w)$

→ The **input layer** $\mathbf{x} = [v(w_1), \dots, v(w_k)]$ has **$V \times k$ elements**

— We assume **one hidden layer** \mathbf{h}

— The **output layer** is a softmax over V elements

$$P(w \mid w_1 \dots w_k) = \text{softmax}(\mathbf{h}\mathbf{W}^2 + \mathbf{b}^2)$$

An n-gram model $P(w \mid w_1 \dots w_k)$ as a feedforward net (**better**)

Assumptions:

The **vocabulary** V contains V types (incl. UNK, BOS, EOS)

We want to condition each word on k preceding words

Our (**better**) model:

— **[Better]**

Each **input word** $w_i \in V$ is an **n -dimensional dense embedding vector** $v(w)$ (**with $n \ll V$**)

→ The **input layer** $\mathbf{x} = [v(w_1), \dots, v(w_k)]$ has **$n \times k$ elements**

— We assume **one hidden layer** \mathbf{h}

— The **output layer** is a softmax over V elements

$$P(w \mid w_1 \dots w_k) = \text{softmax}(\mathbf{h}\mathbf{W}^2 + \mathbf{b}^2)$$

Our neural n-gram models

Architecture:

Input Layer: $\mathbf{x} = [v(w_1) \dots v(w_k)]$

Hidden Layer: $\mathbf{h} = g(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1)$

Output Layer: $P(w \mid w_1 \dots w_k) = \text{softmax}(\mathbf{h}\mathbf{W}^2 + \mathbf{b}^2)$

How many parameters do we need? [# of weights and biases]:

Hidden layer with one-hot inputs: $\mathbf{W}^1 \in \mathbb{R}^{(k \cdot V) \times \text{dim}(\mathbf{h})}$ $\mathbf{b}^1 \in \mathbb{R}^{\text{dim}(\mathbf{h})}$

Hidden layer with dense inputs: $\mathbf{W}^1 \in \mathbb{R}^{(k \cdot n) \times \text{dim}(\mathbf{h})}$ $\mathbf{b}^1 \in \mathbb{R}^{\text{dim}(\mathbf{h})}$

Output layer (any inputs): $\mathbf{W}^2 \in \mathbb{R}^{\text{dim}(\mathbf{h}) \times V}$ $\mathbf{b}^2 \in \mathbb{R}^V$

With $V = 10\text{K}$, $n = 300$ (word2vec), $\text{dim}(\mathbf{h}) = 300$

$k = 2$ (trigram): $\mathbf{W}^1 \in \mathbb{R}^{20,000 \times 300}$ or $\mathbf{W}^1 \in \mathbb{R}^{600 \times 300}$ and $\mathbf{b}^1 \in \mathbb{R}^{300}$

$k = 5$ (six-gram): $\mathbf{W}^1 \in \mathbb{R}^{50,000 \times 300}$ or $\mathbf{W}^1 \in \mathbb{R}^{1500 \times 300}$ and $\mathbf{b}^1 \in \mathbb{R}^{300}$

$\mathbf{W}^2 \in \mathbb{R}^{300 \times 10,000}$ $\mathbf{b}^2 \in \mathbb{R}^{10,000}$

Six-gram model with one-hot inputs: 27,000,460,000 parameters,
with dense inputs: 3,460,000 parameters

Traditional six-gram model: $10^{4 \times 6} = 10^{24}$ parameters

1D CNNs for text

Text is a (variable-length) **sequence** of words (word vectors)

[#channels = dimensionality of word vectors]

We can use a **1D CNN** to slide a window of n tokens across:

— Filter size $n = 3$, stride = 1, no padding

The quick brown fox jumps over the lazy dog

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

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

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

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

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

— Filter size $n = 2$, stride = 2, no padding:

The quick brown fox jumps over the lazy dog

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

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

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

What we have learned last time

- ❑ Language Modeling
- ❑ N-gram is a simple language model
- ❑ Neural N-gram Language Model
- ❑ Feedforward network and CNN for NLP

Recap: Language models

Many NLP tasks require **natural language output**:

- Machine translation
- Speech recognition
- Natural language generation
- Spell-checking

Language models define **probability distributions over (natural language) strings or sentences**.

- We can use a language model to **generate strings**
- We can use a language model to **score/rank candidate strings** so that we can choose the best (i.e. most likely) one:
if $P_{LM}(A) > P_{LM}(B)$, return A, not B

Recap: Language modeling with N-grams

A **language model** over a vocabulary V assigns probabilities to strings drawn from V^* .

How do we compute the **probability of a string** $w^{(1)} \dots w^{(i)}$?

Recall the **chain rule**:

$$P(w^{(1)} \dots w^{(i)}) = P(w^{(1)}) \cdot P(w^{(2)} | w^{(1)}) \cdot \dots \cdot P(w^{(i)} | w^{(i-1)}, \dots, w^{(1)})$$

An **n-gram** language model assumes each word $w^{(i)}$ depends only on the **last n-1 words** $w^{(i-1)}, \dots, w^{(i-(n+1))}$

$$P_{ngram}(w^{(1)} \dots w^{(i)}) = P(w^{(1)}) \cdot P(w^{(2)} | w^{(1)}) \cdot \dots \cdot P(w^{(i)} | w^{(i-1)}, \dots, w^{(i-(n+1))})$$

Recap: N-gram models

N-gram models *assume* each word (event) depends only on the previous $n-1$ words (events):

$$\text{Unigram model: } P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^N P(w^{(i)})$$

$$\text{Bigram model: } P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^N P(w^{(i)} | w^{(i-1)})$$

$$\text{Trigram model: } P(w^{(1)} \dots w^{(N)}) = \prod_{i=1}^N P(w^{(i)} | w^{(i-1)}, w^{(i-2)})$$

Independence assumptions where the n -th event in a sequence depends only on the last $n-1$ events are called **Markov assumptions (of order $n-1$)**.

1D CNNs for text

Text is a (variable-length) **sequence** of words (word vectors)

[#channels = dimensionality of word vectors]

We can use a **1D CNN** to slide a window of n tokens across:

— Filter size $n = 3$, stride = 1, no padding

The quick brown fox jumps over the lazy dog

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

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

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

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

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

— Filter size $n = 2$, stride = 2, no padding:

The quick brown fox jumps over the lazy dog

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

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

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

Example question

1. (1 point) Calculate the probability of the sentence *i want to eat lunch*, given the probabilities for a bi-gram language model in Fig. 1. Assume $P(i|<s>)=0.19$ with start-symbol $<s>$ and $P(</s>|lunch)=0.40$ with end-symbol $</s>$.

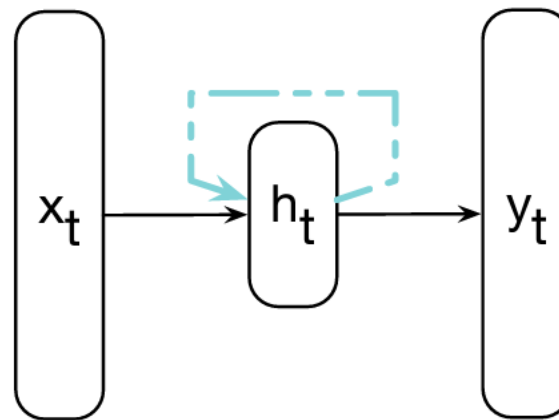
	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Figure 1: Bigram probabilities for eight words learned from a corpus. Zero probabilities are in gray. The rows are previous words and the columns are next words.

What's wrong with MLP or CNN for NLP?

Recurrent Neural Network

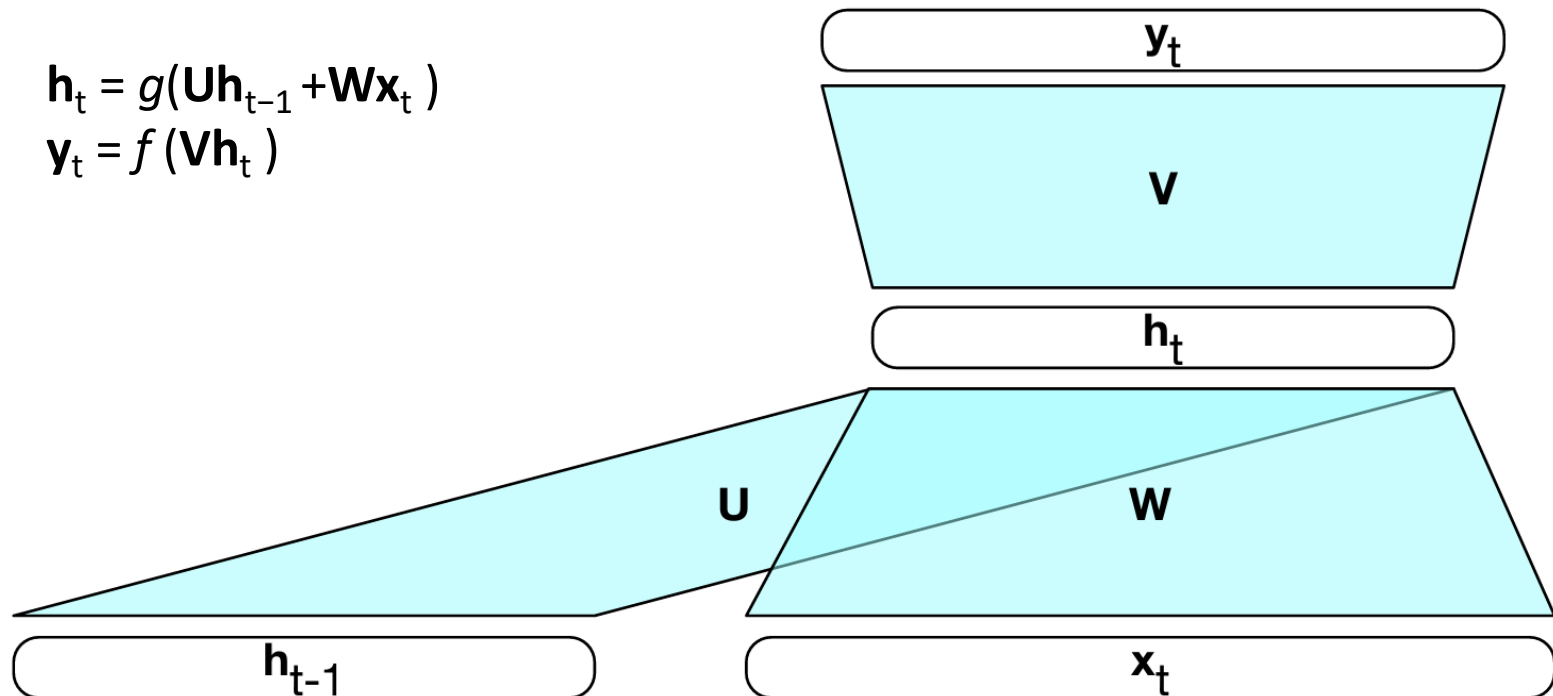
- ❑ Temporal nature in language processing
- ❑ RNN deals with sequential input data stream like language.



A simple RNN

A Simple Recurrent Neural Network

□ RNN illustrated as a feed-forward network

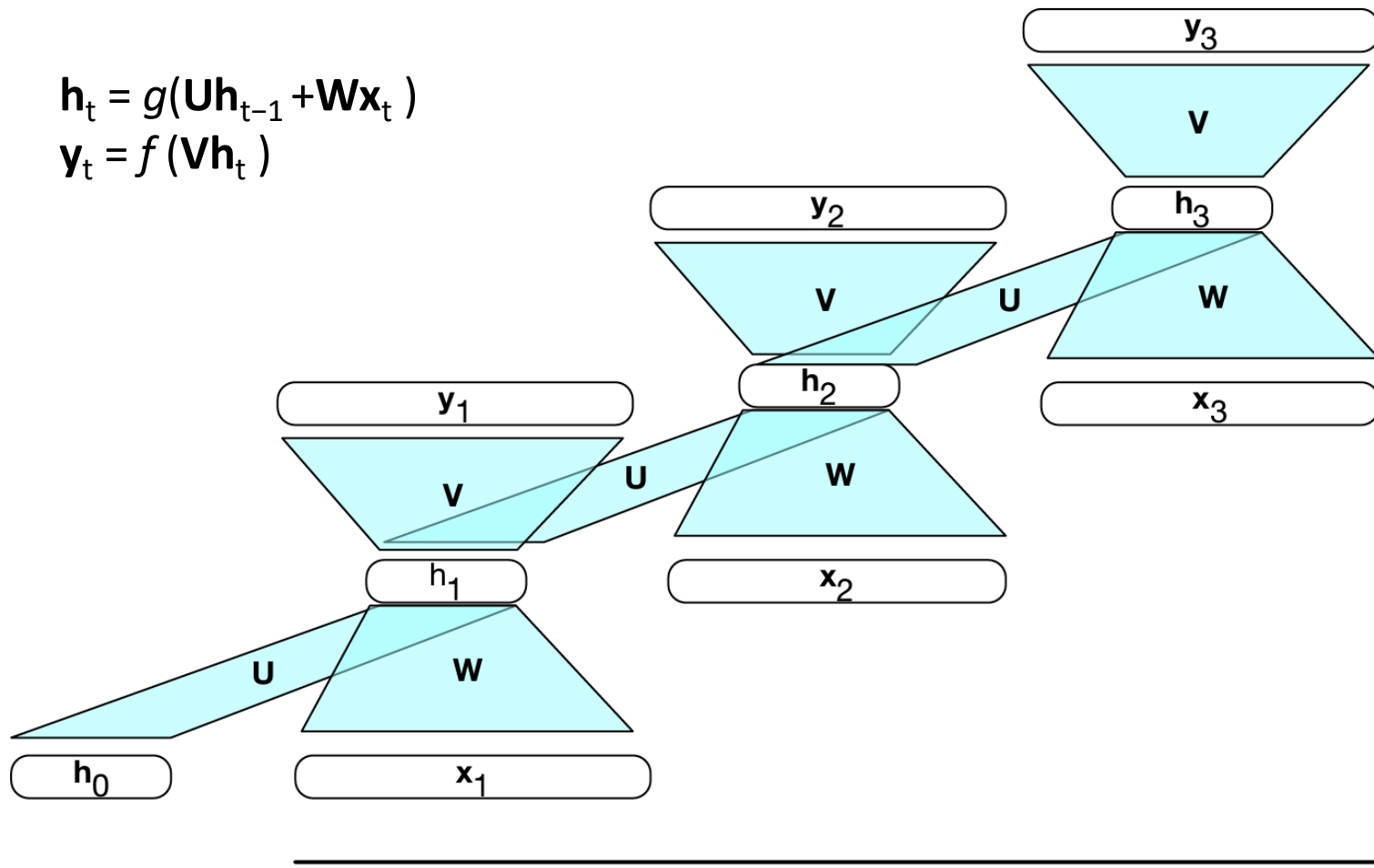


A Simple Recurrent Neural Network

□ RNN unrolled in time

$$h_t = g(Uh_{t-1} + Wx_t)$$

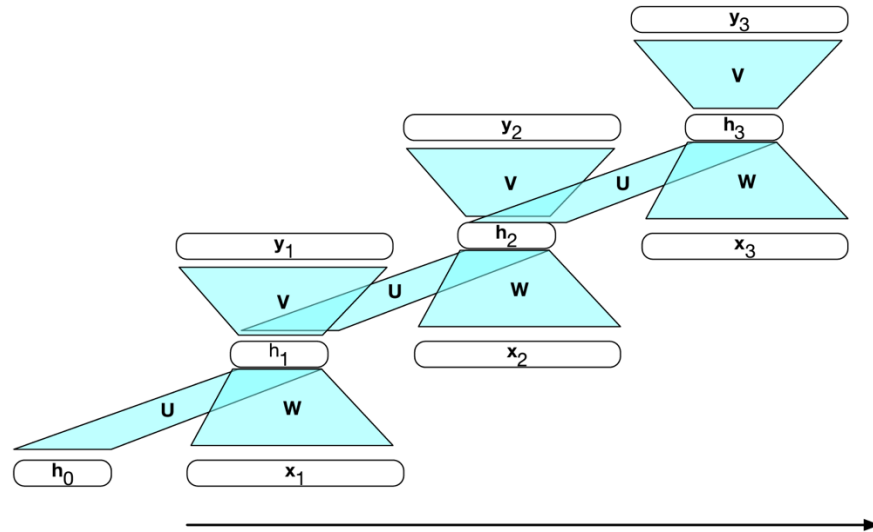
$$y_t = f(Vh_t)$$



How to optimize Recurrent Neural Network?

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t)$$

$$\mathbf{y}_t = f(\mathbf{V}\mathbf{h}_t)$$



□ Backpropagation through time

$$\frac{\partial L_3}{\partial \mathbf{W}} = \frac{\partial L_3}{\partial \mathbf{h}_3} \frac{\partial \mathbf{h}_3}{\partial \mathbf{W}} + \frac{\partial L_3}{\partial \mathbf{h}_3} \frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_2}{\partial \mathbf{W}} + \frac{\partial L_3}{\partial \mathbf{h}_3} \frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2} \frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1} \frac{\partial \mathbf{h}_1}{\partial \mathbf{W}}$$

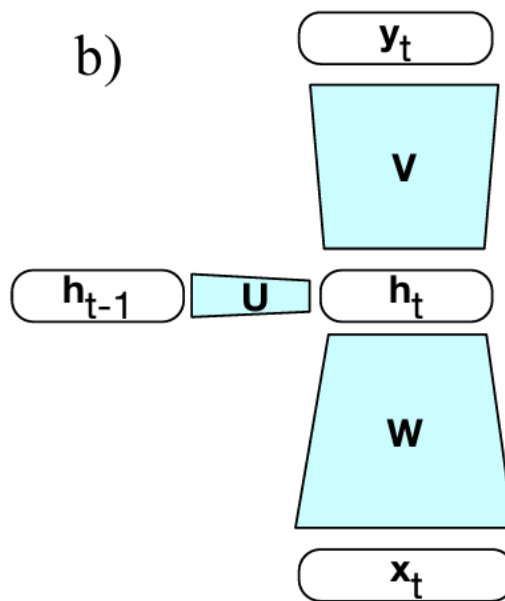
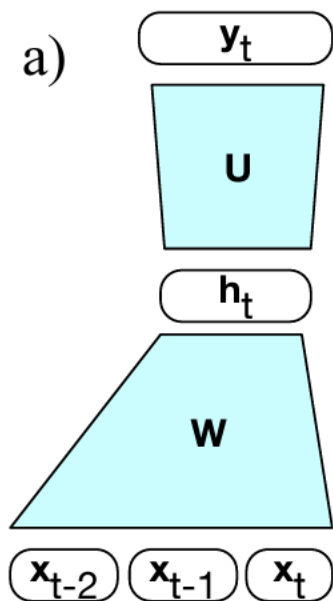
$$\boxed{\frac{\partial L}{\partial \mathbf{W}} = -\frac{1}{n} \sum_{t=1}^n \sum_{k=1}^t \frac{\partial L_t}{\partial \mathbf{h}_t} \left(\prod_{j=k+1}^t \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \right) \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}}}$$

RNNs as Language Models

- Language models predict the next word in a sequence given some preceding context.

$P(\text{fish} | \text{Thanks for all the})$

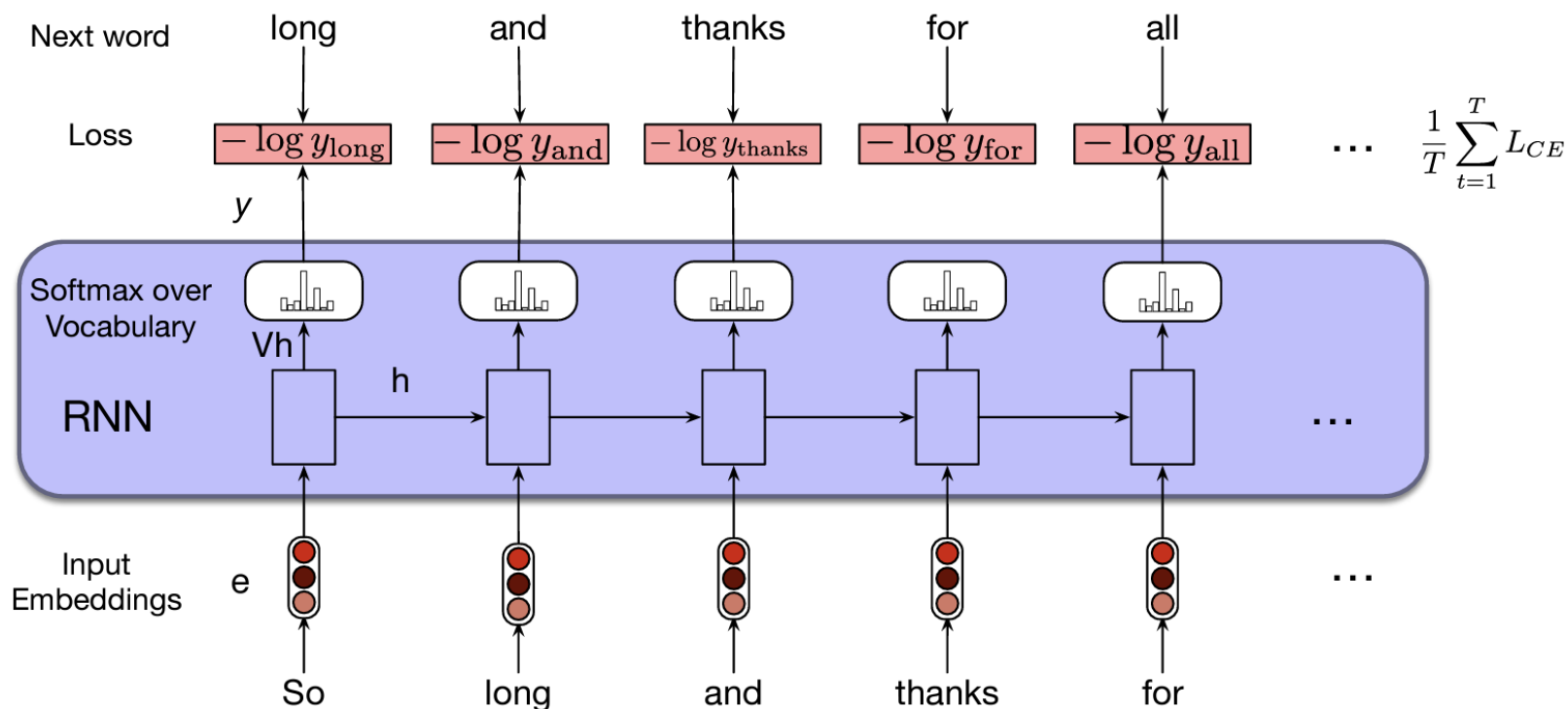
- RNN Language Model



$$h_t = g(Uh_{t-1} + Wx_t)$$
$$y_t = \text{softmax}(Vh_t)$$

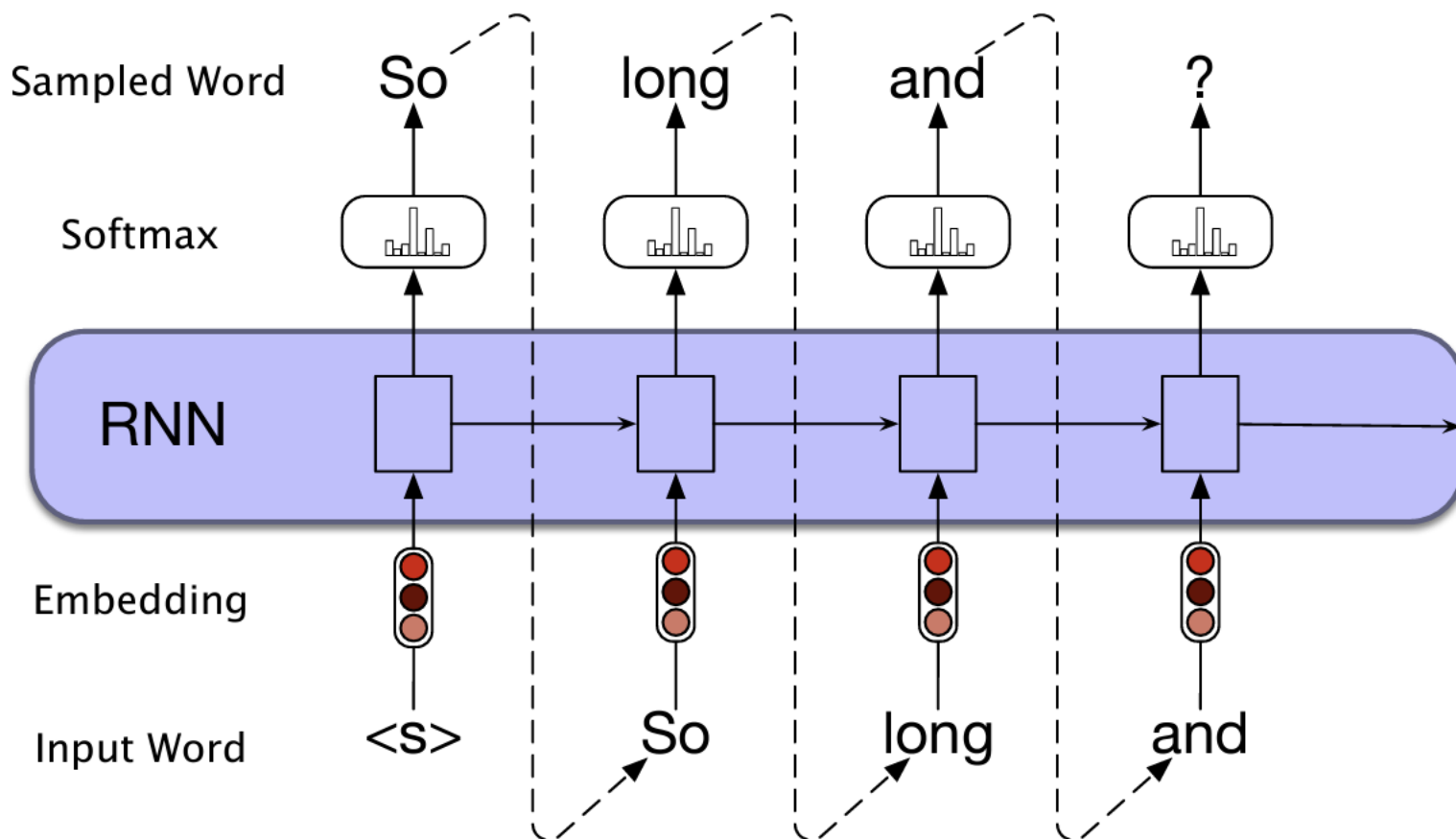
Training an RNN Language Model

Maximum likelihood estimation



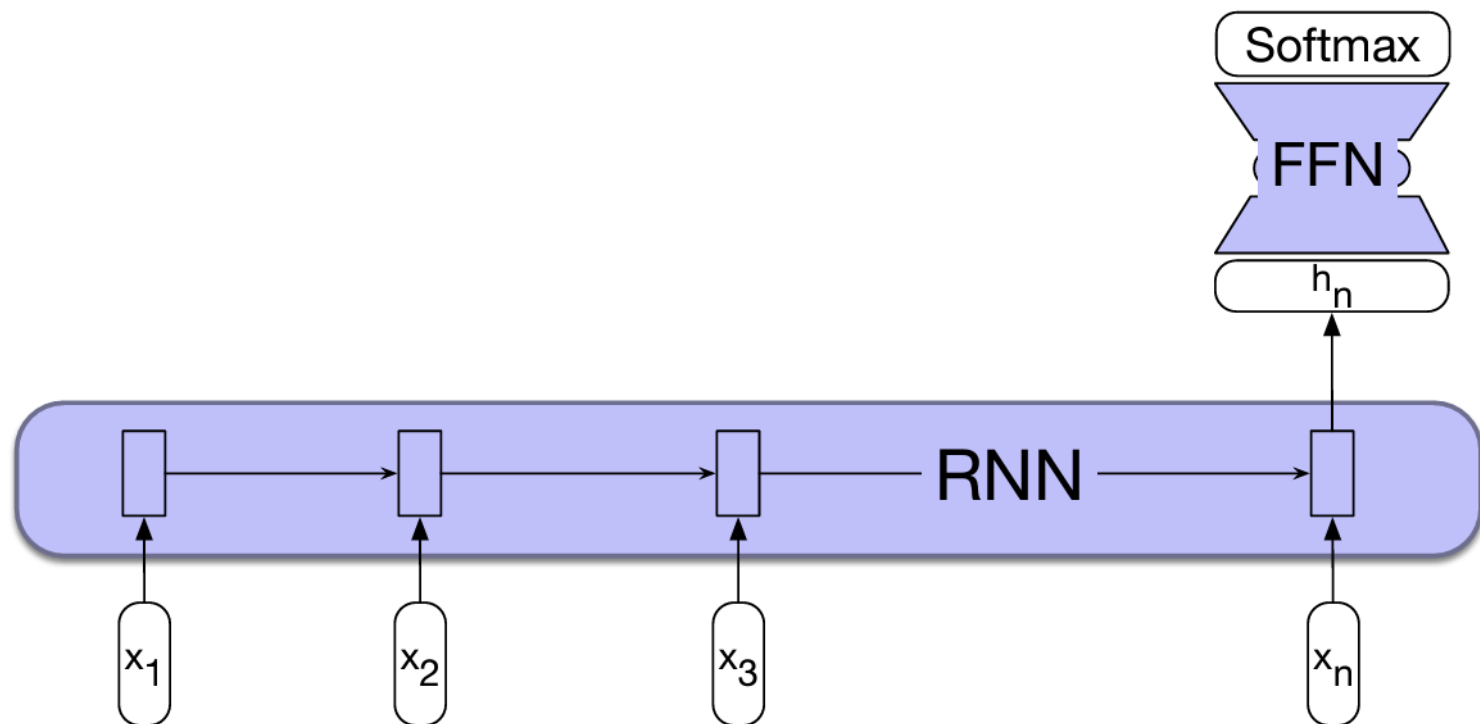
Generation with RNN Language Model

□ Autoregressive (casual) generation



RNNs for other NLP tasks

- ❑ RNN for sequence classification
- ❑ Commonly called text classification, like sentiment analysis or spam detection



Vanishing/exploding gradients

- Consider the gradient of L_t at step t , with respect to the hidden state \mathbf{h}_k at some previous step k ($k < t$):

$$\frac{\partial L_t}{\partial \mathbf{h}_k} = \frac{\partial L_t}{\partial \mathbf{h}_t} \left(\prod_{t \geq j > k} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \right)$$

- Recurrent multiplication
- Gradients too small (vanishing gradient) or too large (exploding gradient)

Exploding gradients

- ❑ What is the problem?
- ❑ We take a very large step in SGD
- ❑ Solution: Gradient clipping

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
```

Vanishing gradients

- ❑ What is the problem?
- ❑ Parameters barely get updated (no learning)
- ❑ Solution:
 - ❑ LSTMs: Long short-term memory networks

Local vs distant information

- ❑ Hidden states tend to contain local information

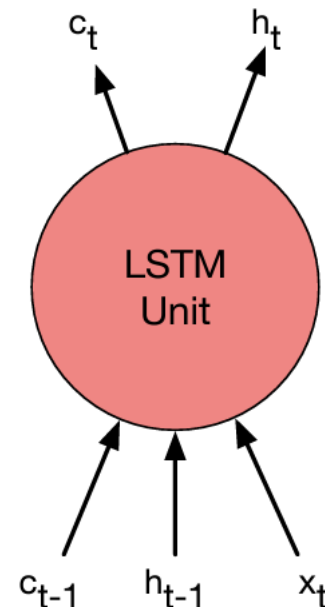
- ❑ But distant information is critical

“The flights the airline was canceling were full”

- ❑ Should predict “were” given distant information (flights)

Long Short-term Memory (LSTM)

- ❑ A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the vanishing gradients problem
- ❑ Basic idea: turning multiplication into addition
- ❑ Use “gates” to control how much information to add/erase
- ❑ At each timestep, there is a hidden state h_t (local information) and also a cell state C_t (distant information)



Long Short-term Memory (LSTM)

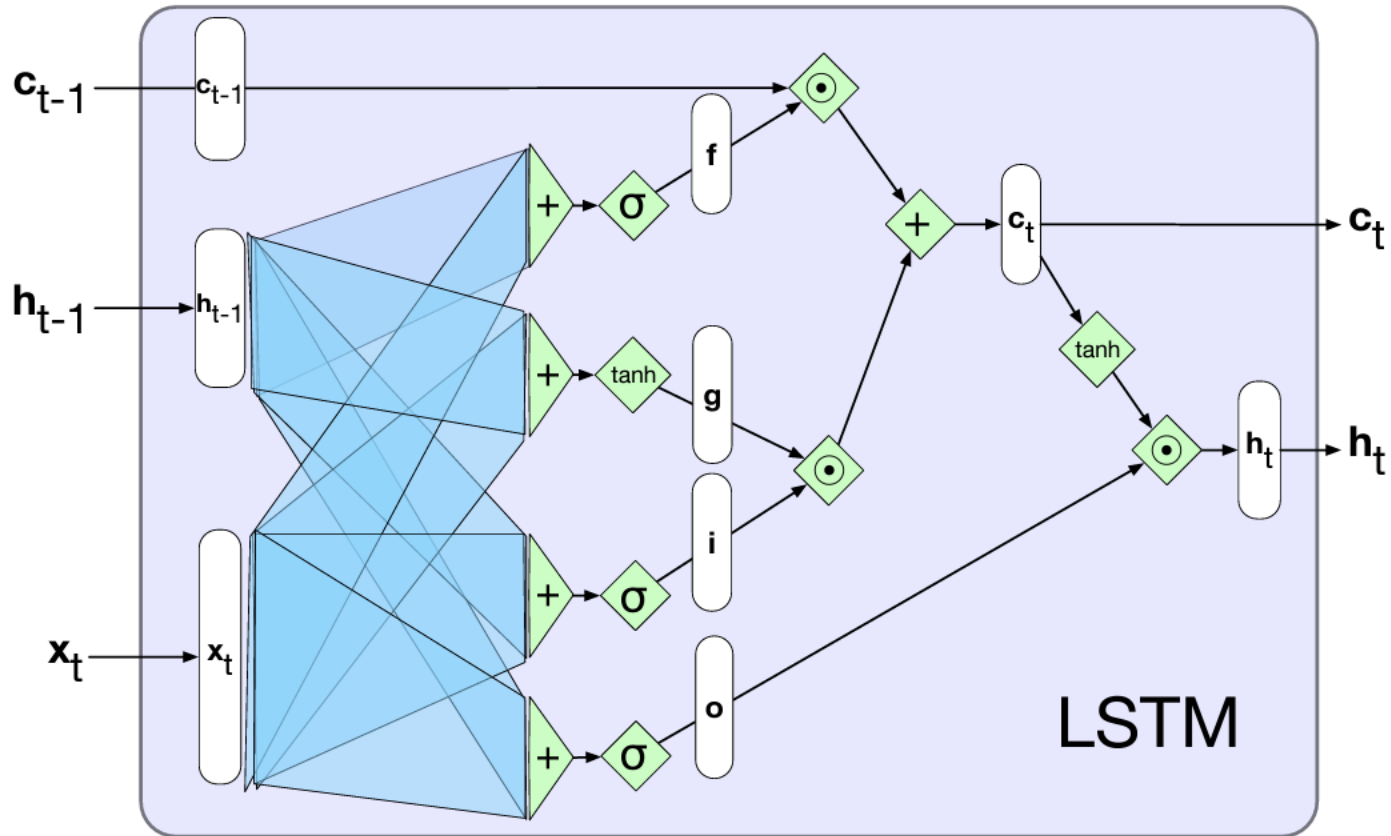
- ❑ Gate: feedforward layer, followed by a sigmoid activation function, followed by a pointwise multiplication with the layer being gated
- ❑ For example, output gate (What to output for hidden state)

$$\mathbf{o}_t = \sigma(\mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{W}_o \mathbf{x}_t)$$

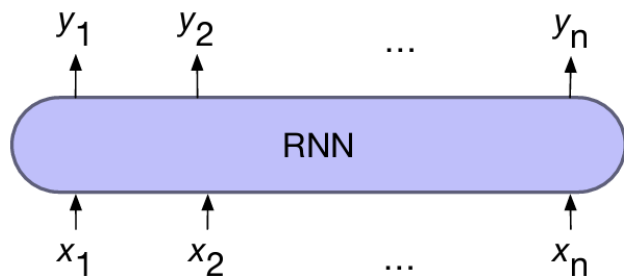
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

- ❑ Other gates
 - ❑ Forget gate
 - ❑ Add gate
 - ❑ Input gate

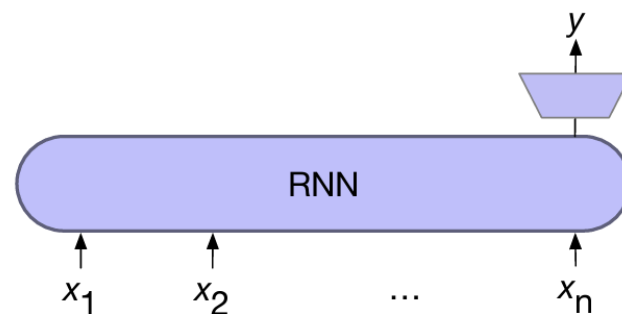
Long Short-term Memory (LSTM)



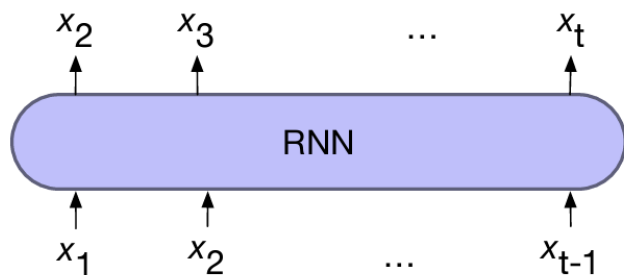
Summary: Common RNN NLP architectures



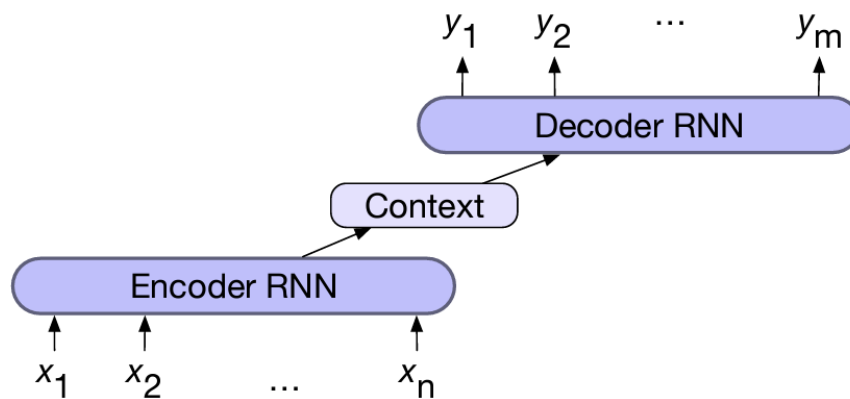
a) sequence labeling



b) sequence classification



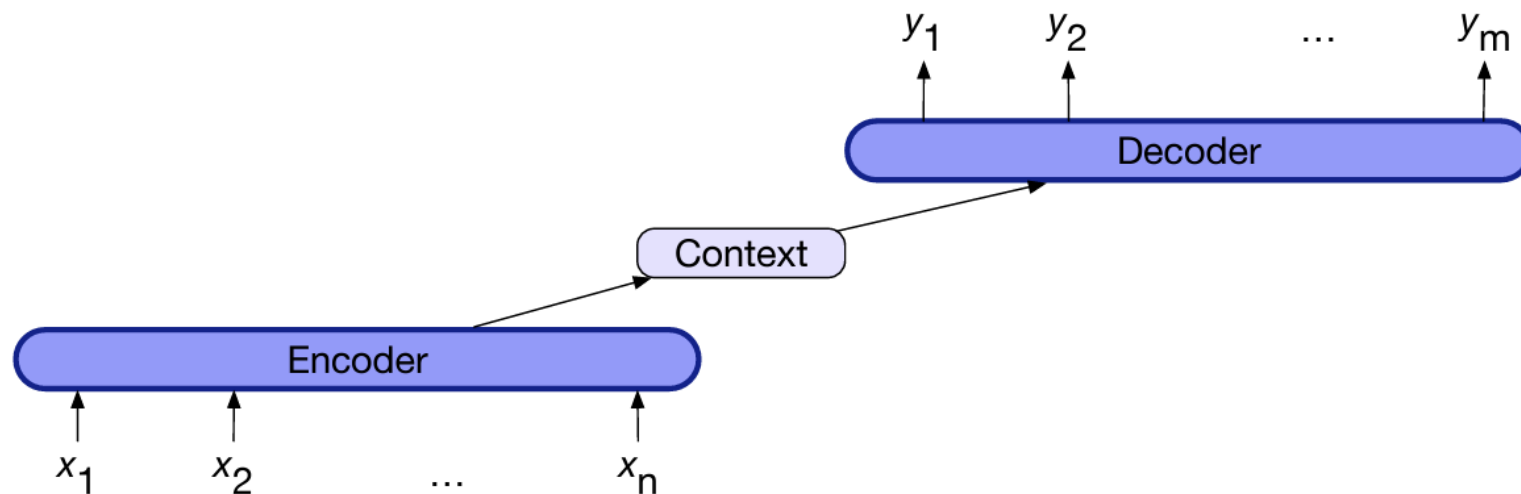
c) language modeling



d) encoder-decoder

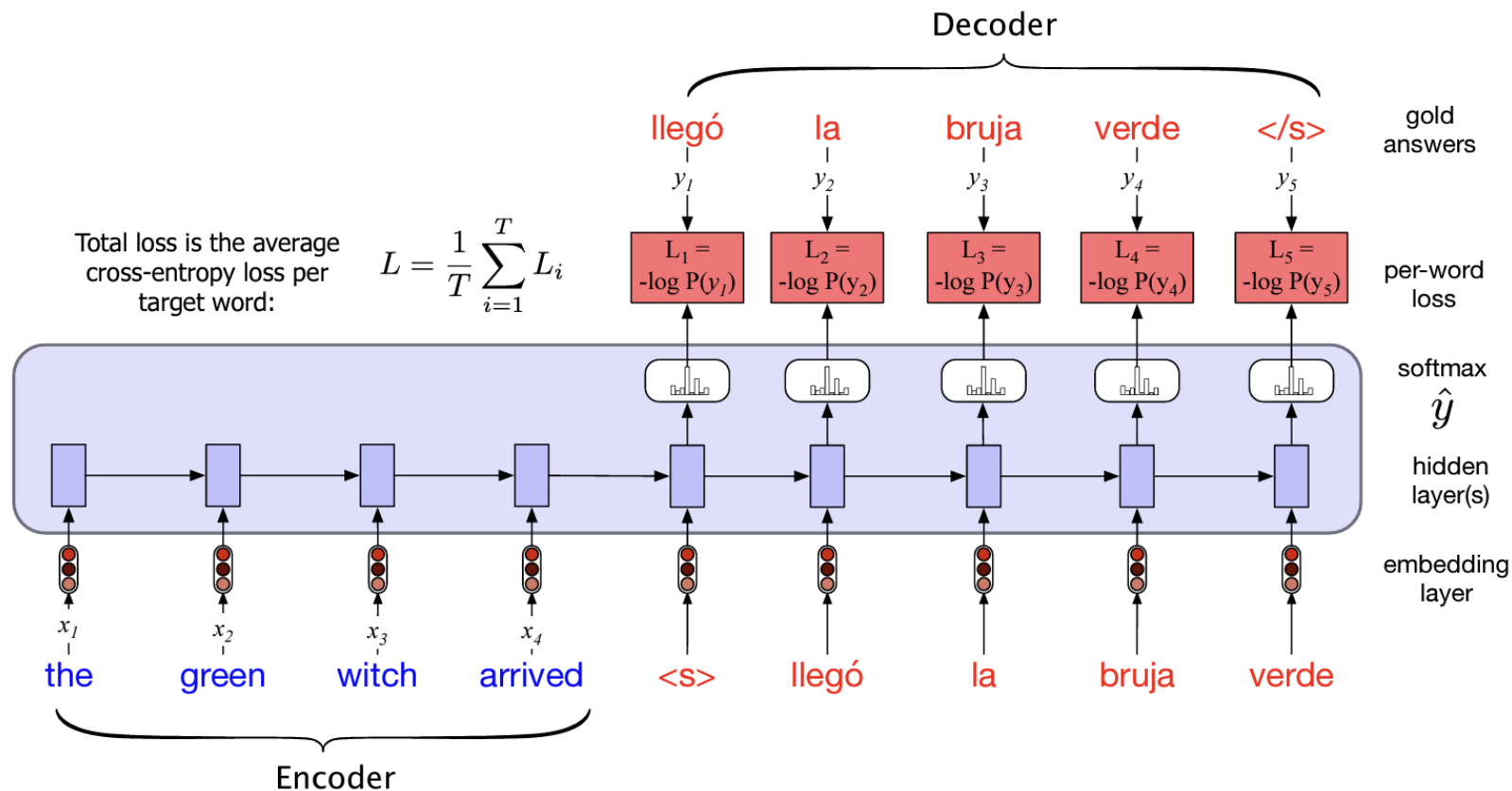
Encoder Decoder Architecture

- ❑ Arbitrary length output given an input sequence
- ❑ A.K.A. sequence-to-sequence network
- ❑ Context vector conveys the essence of the input to the decoder



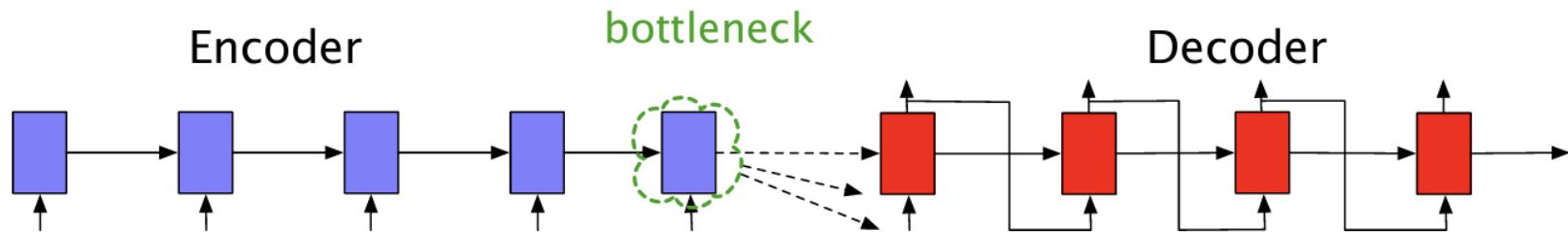
Encoder Decoder Architecture

□ Training an encoder-decoder for machine translation



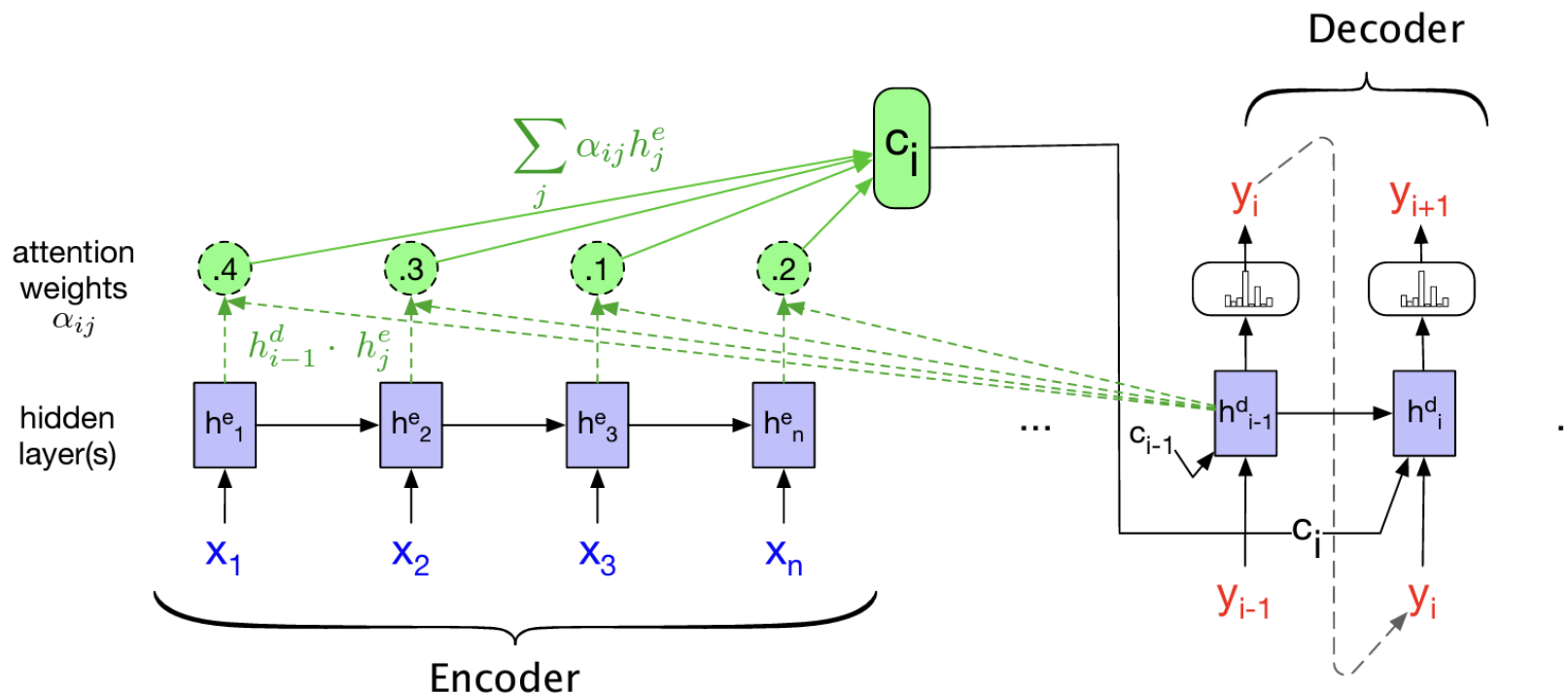
Problem of Encoder-decoder architecture

- ❑ Context vector encodes EVERYTHING about input sequence
- ❑ Context vector acts as a bottleneck



Attention Mechanism

- ❑ Each output in decoder accesses all the hidden states from the encoder, not just the last state
- ❑ Each output attends to all input



Transformer: The intuition

- ❑ Context matters for natural language understanding

- ❑ For example:

- ❑ The **chicken** crossed the road because **it** wanted to get to the other side

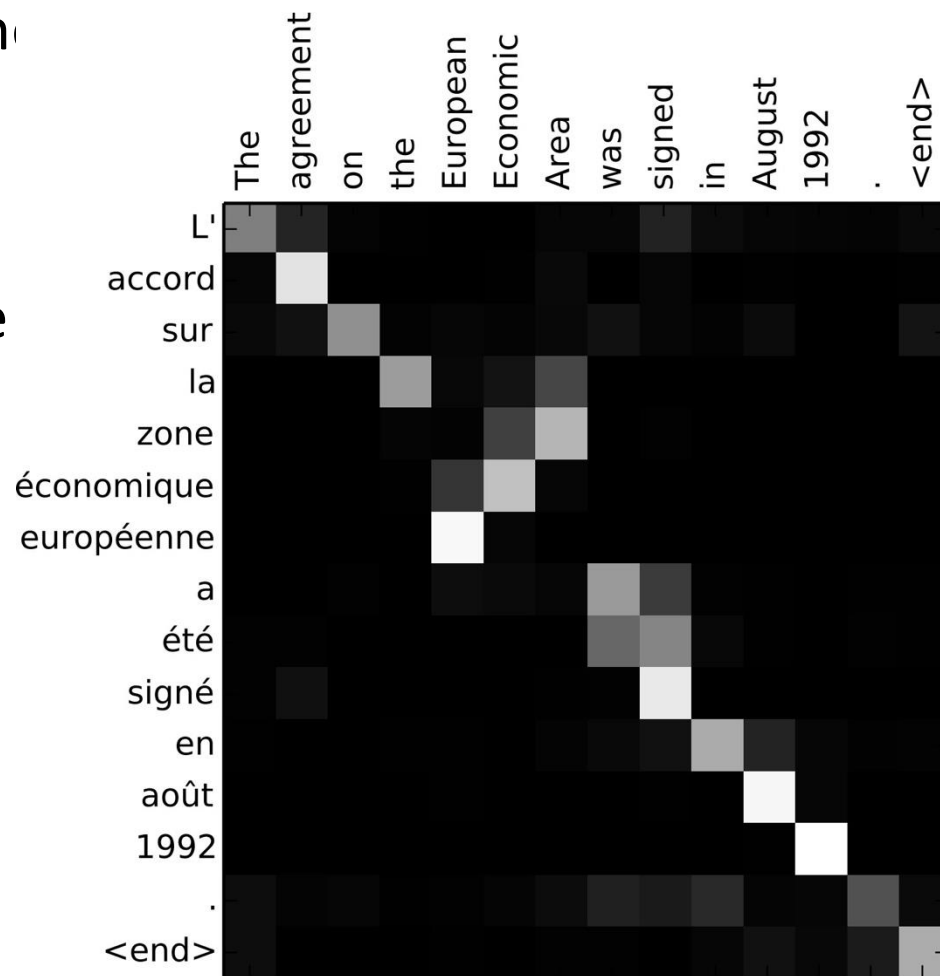
- ❑ I walked along the **pond**, and noticed that one of the trees along the **bank** had fallen into the **water** after the storm.

Attention weights between words

❑ Example: English to French translation

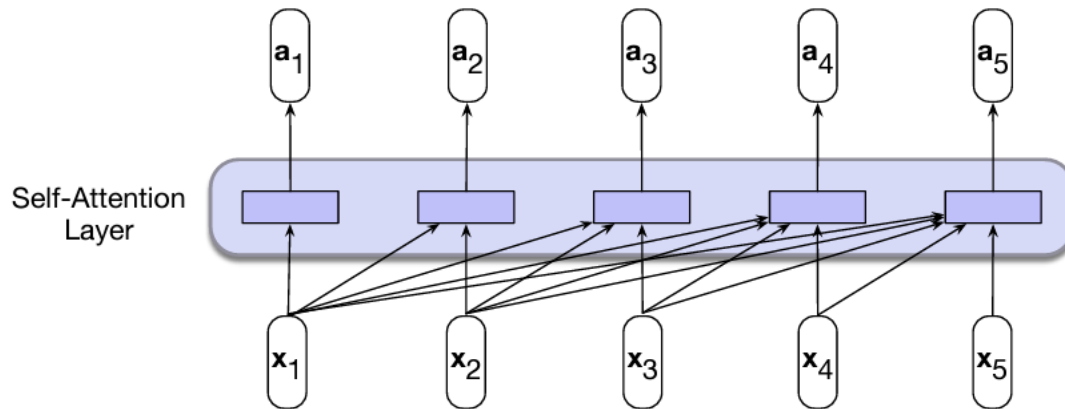
❑ Input: “The agreement on the European Economic Area was signed in August 1992.”

❑ Output: “L'accord sur la zone économique européenne a été signé en août 1992.”



Casual or backward-looking self-attention

- Attends to all the inputs up to, and including, the current one



Self-attention

□ Version 1:

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

$$\begin{aligned} \alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \leq i \\ &= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^i \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \quad \forall j \leq i \end{aligned}$$

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

Query, Key, and Value

- ❑ Query: the current focus of attention when being compared to all of the other preceding inputs.
- ❑ Key: a preceding input being compared to the current focus
- ❑ Value: used to compute the output for the current focus

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q; \quad \mathbf{k}_i = \mathbf{x}_i \mathbf{W}^K; \quad \mathbf{v}_i = \mathbf{x}_i \mathbf{W}^V$$

- ❑ Version 2:

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{q}_i \cdot \mathbf{k}_j$$

$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

Self-attention

□ Final Version

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q; \mathbf{k}_i = \mathbf{x}_i \mathbf{W}^K; \mathbf{v}_i = \mathbf{x}_i \mathbf{W}^V$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

$$\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \leq i$$

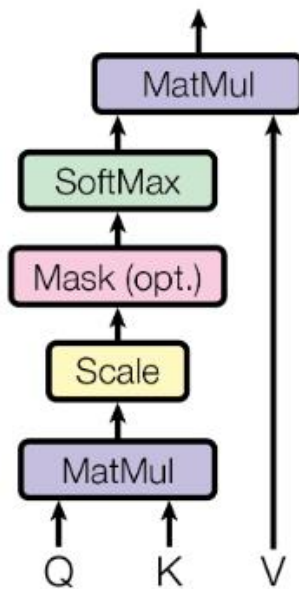
$$\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$$

Attention Operation

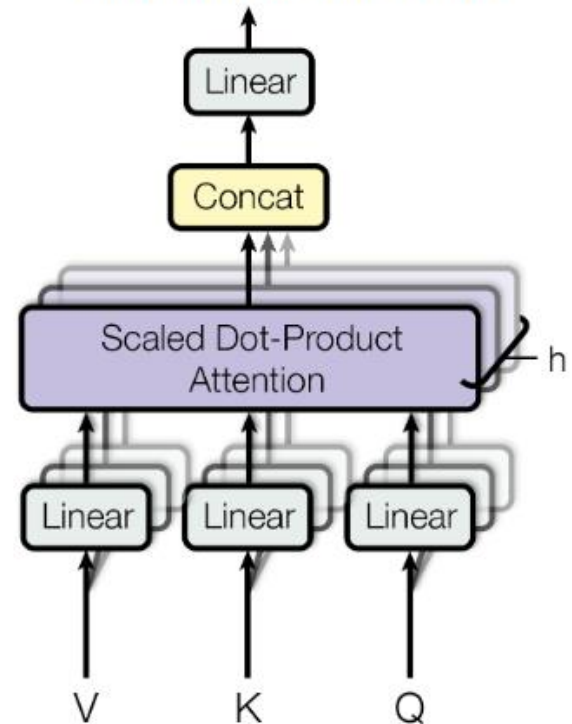
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Multi-head attention

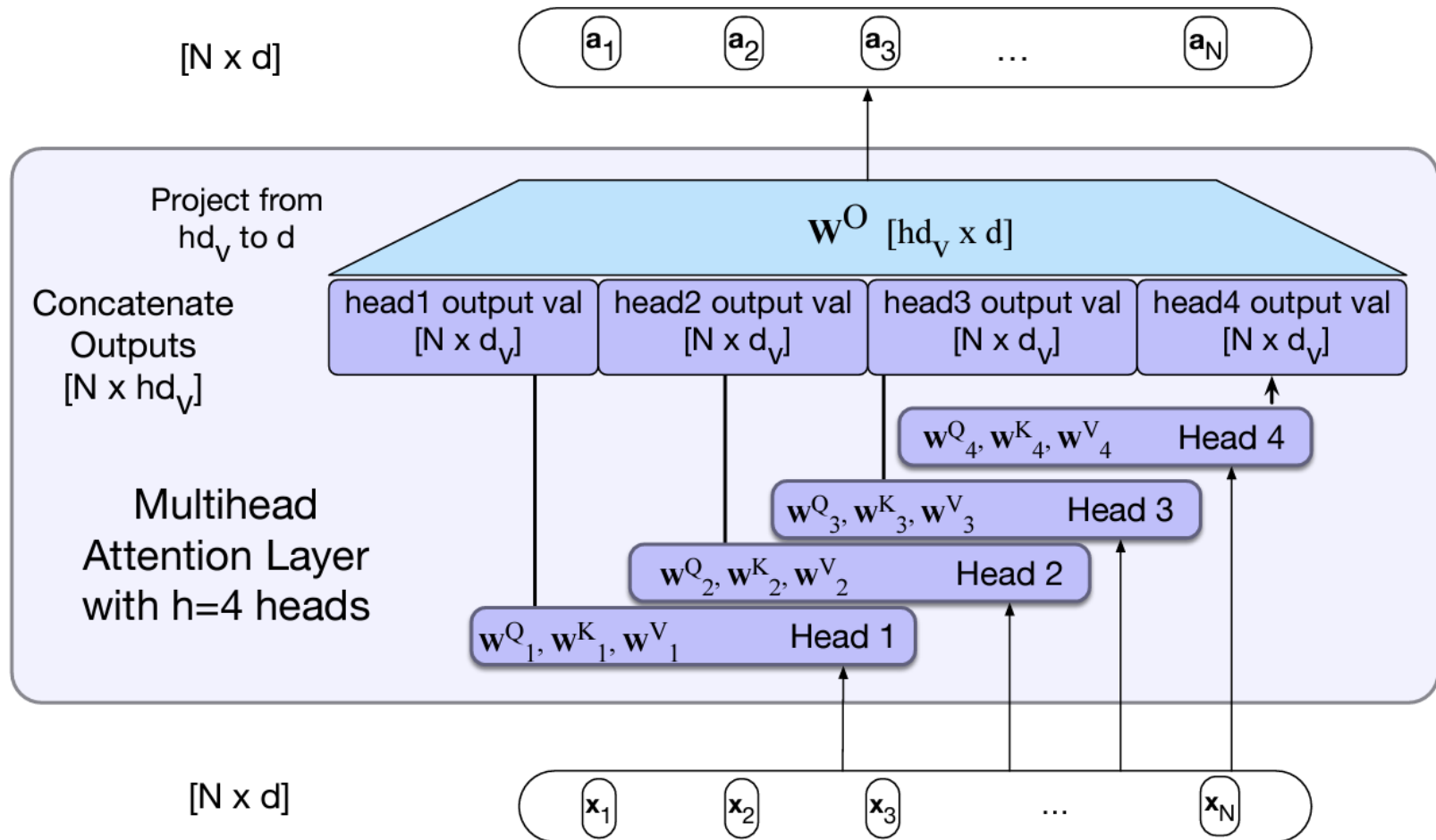
Scaled Dot-Product Attention



Multi-Head Attention



Multi-head attention



Self attention v.s. Cross attention

☐ Self Attention

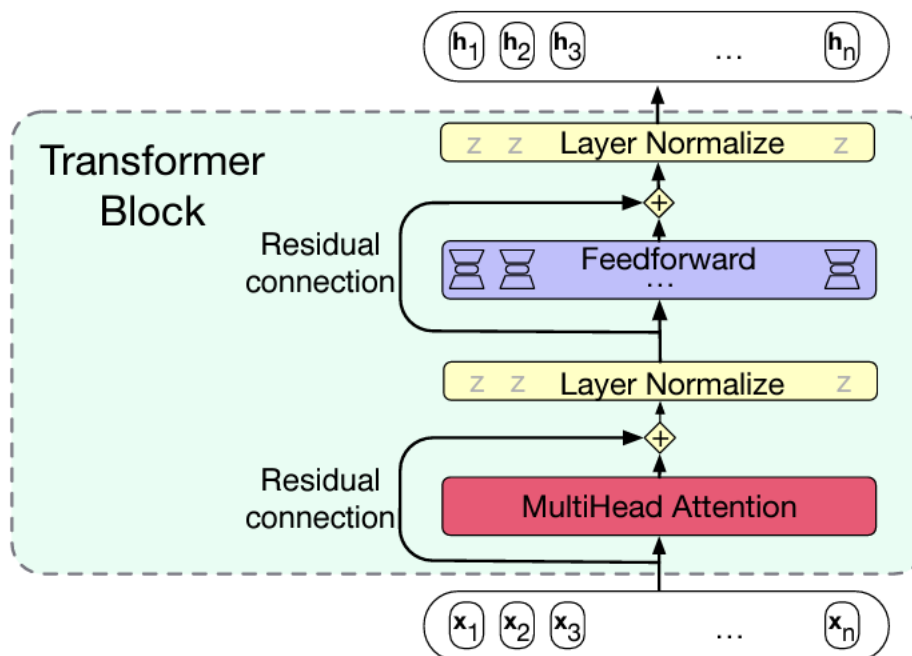
- ☐ Key, Value, and Query from the same set of tokens

☐ Cross Attention

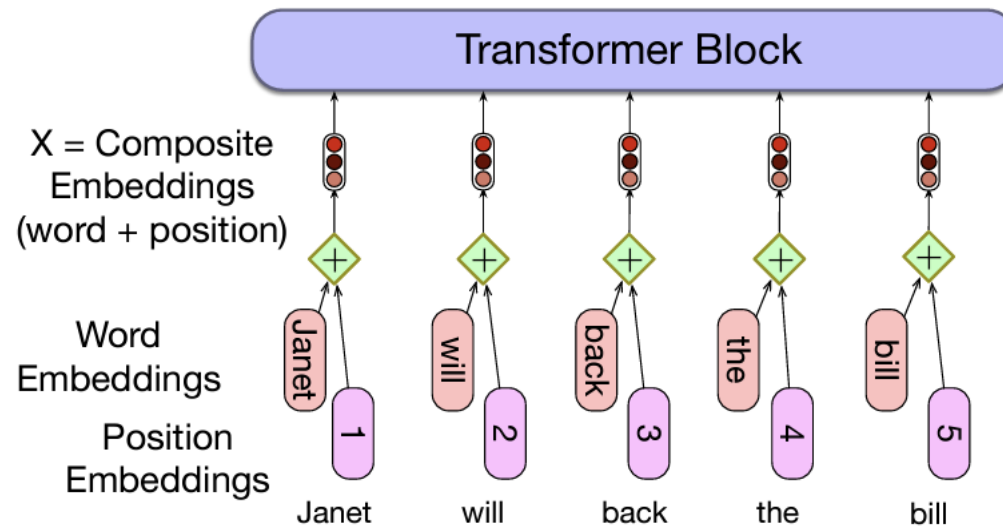
- ☐ Key, and Value from one set of tokens
- ☐ Query from another set of tokens
- ☐ E.g. words in one language pay attention to words in another.

From Attention to Transformer Block

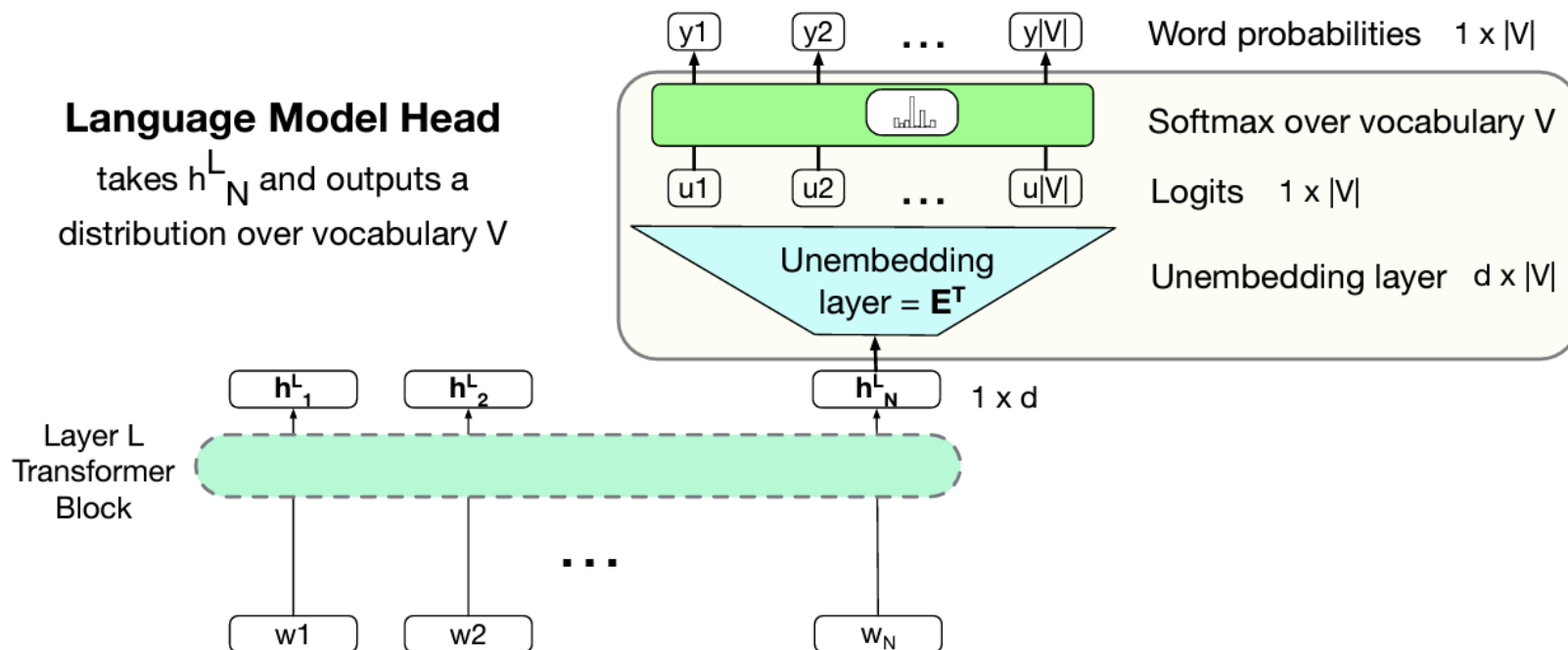
- ❑ A transformer block has
 - ❑ **Self Attention**: information exchange between tokens
 - ❑ **Feed forward network**: Information transform within tokens
 - ❑ E.g. a multi-layer perceptron with 1 hidden layer
 - ❑ **Normalization** (Layer normalization)
 - ❑ **Residual connection**



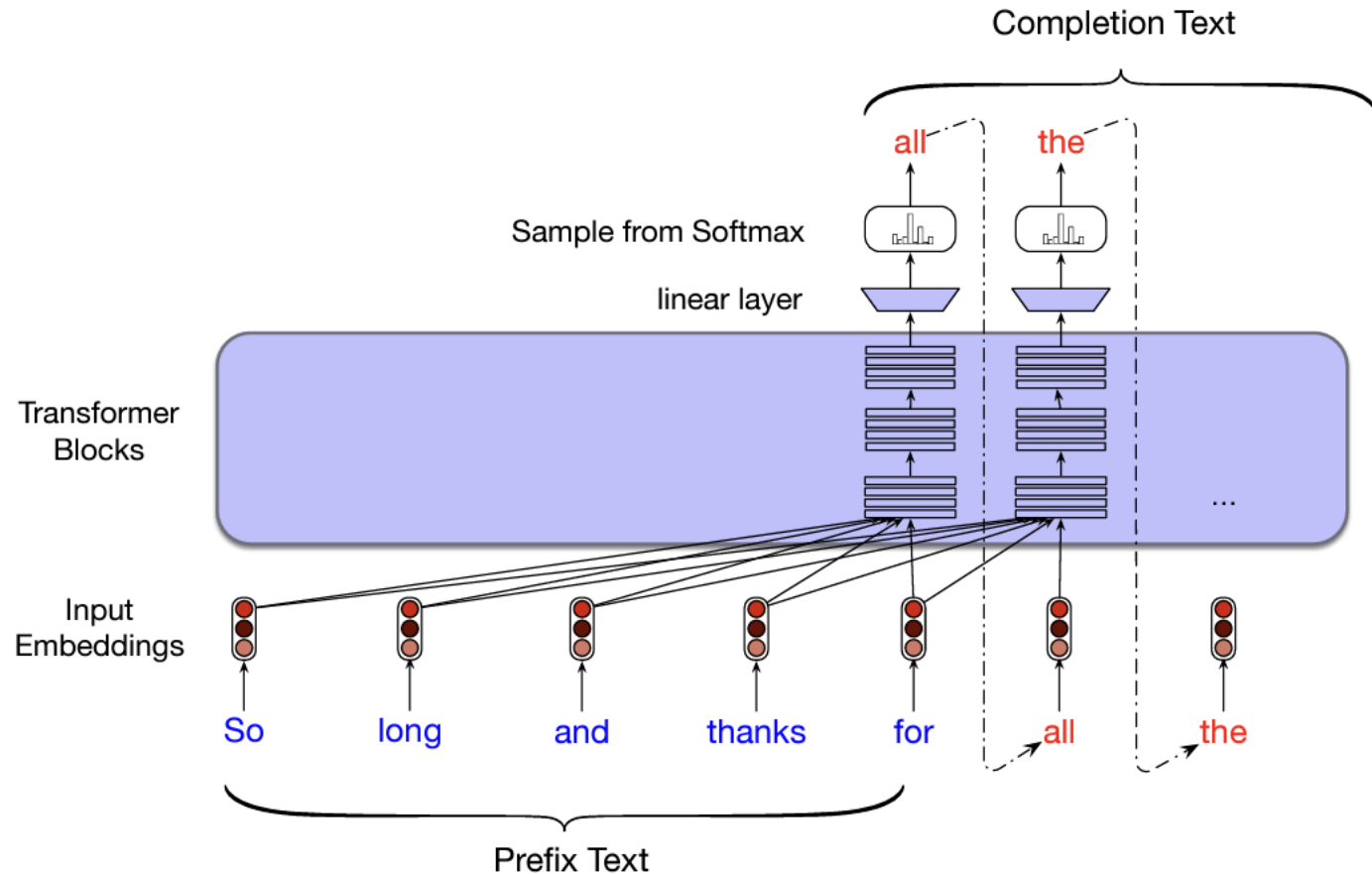
Embedding for token and position



Language Model Head



Transformer-based Large Language Model



Preprint. Under review.

AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

Alexey Dosovitskiy^{*,†}, Lucas Beyer^{*}, Alexander Kolesnikov^{*}, Dirk Weissenborn^{*},
Xiaohua Zhai^{*}, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer,
Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby^{*,†}

^{*}equal technical contribution, [†]equal advising

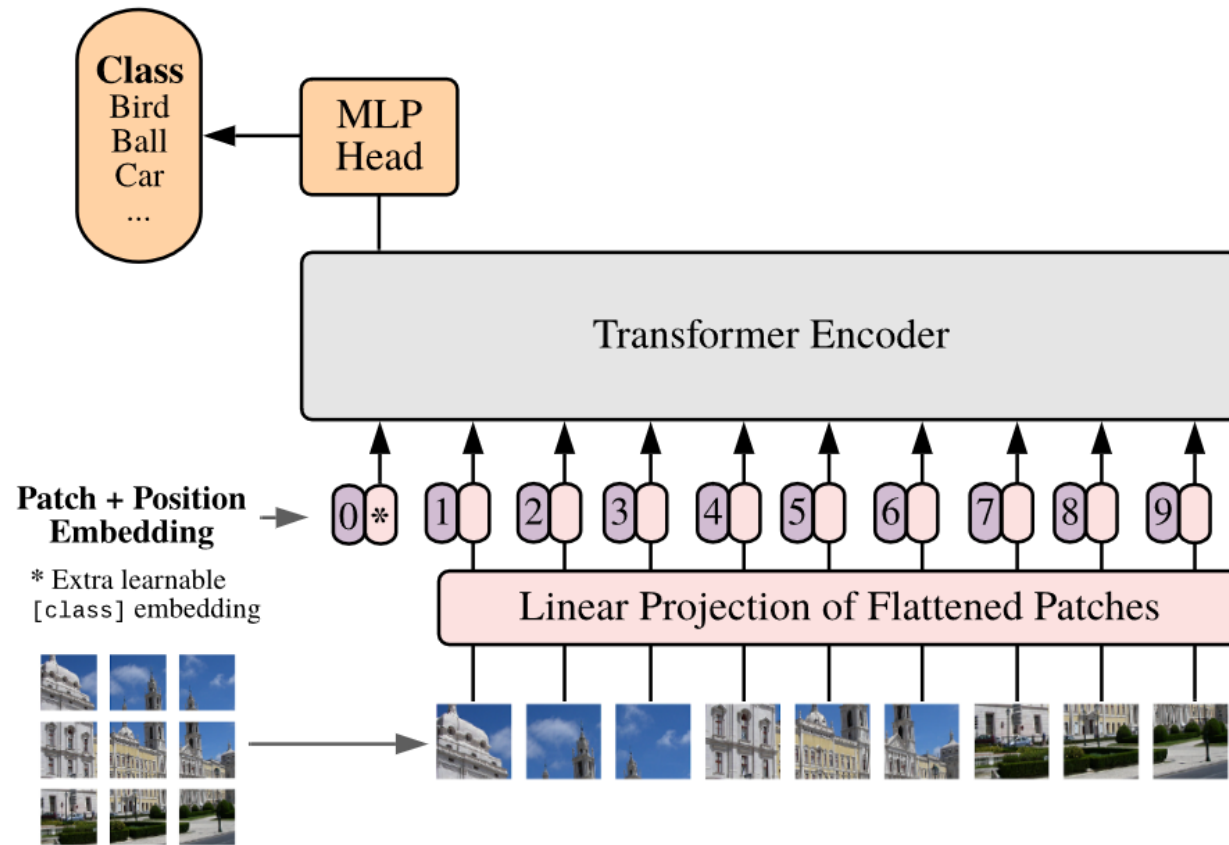
Google Research, Brain Team

{adosovitskiy, neilhoulby}@google.com

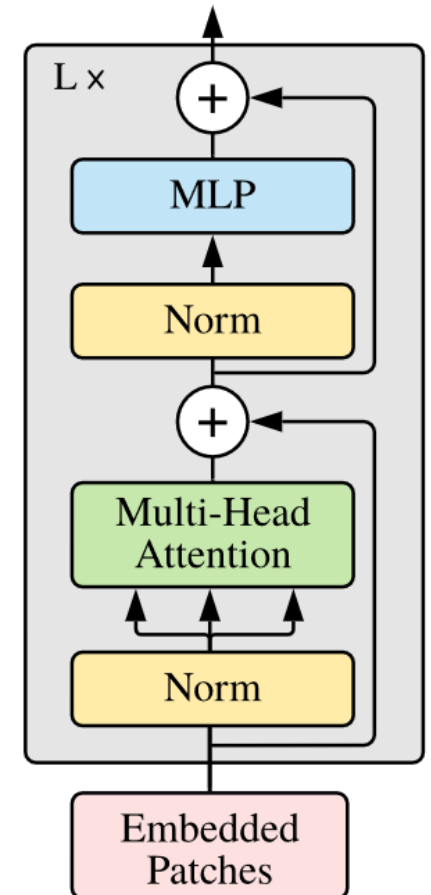
ABSTRACT

While the Transformer architecture has become the de-facto standard for natural language processing tasks, its applications to computer vision remain limited. In vision, attention is either applied in conjunction with convolutional networks, or used to replace certain components of convolutional networks while keeping their overall structure in place. We show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks. When pre-trained on large amounts of data and transferred to multiple mid-sized or small image recognition benchmarks (ImageNet, CIFAR-100, VTAB, etc.), Vision Transformer (ViT) attains excellent results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train.

Vision Transformer (ViT)



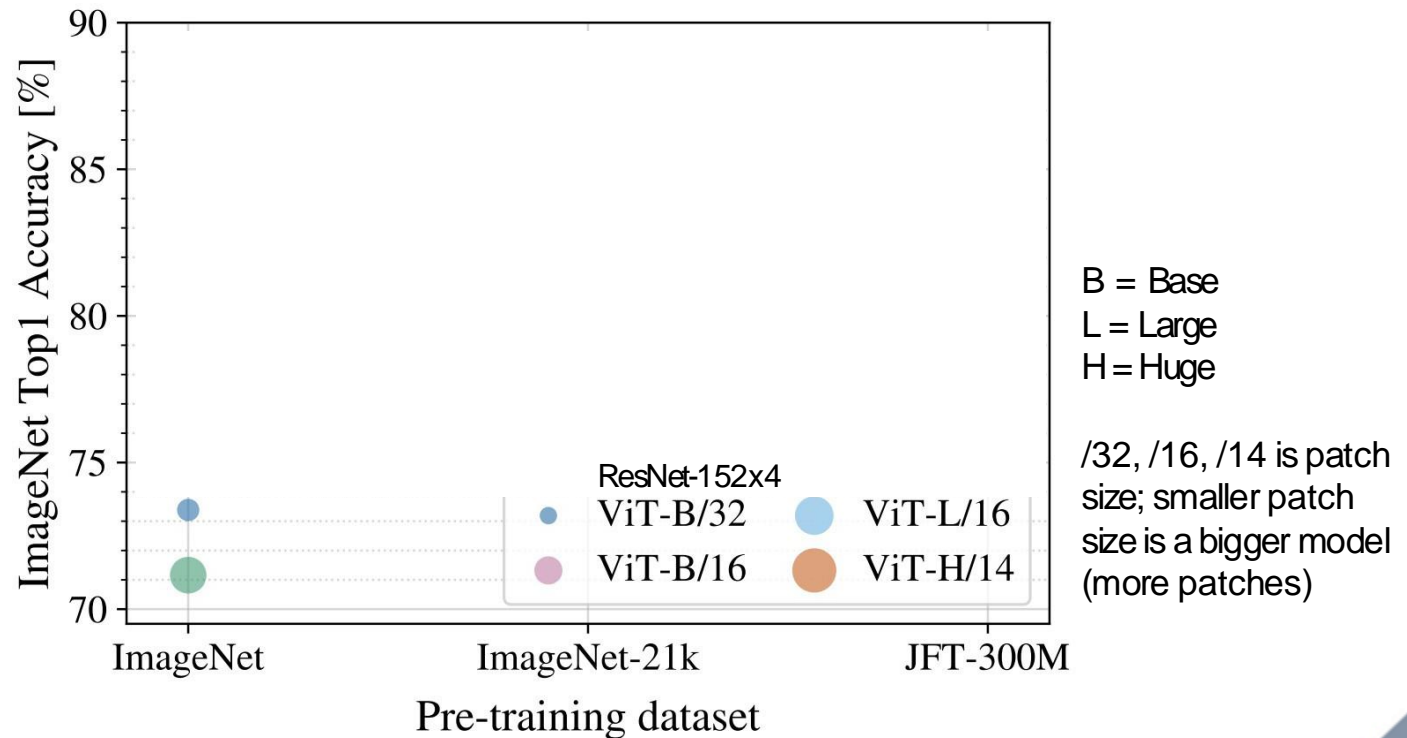
Transformer Encoder



Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 1: Details of Vision Transformer model variants.

Vision Transformer (ViT) vs ResNets



B = Base
L = Large
H = Huge

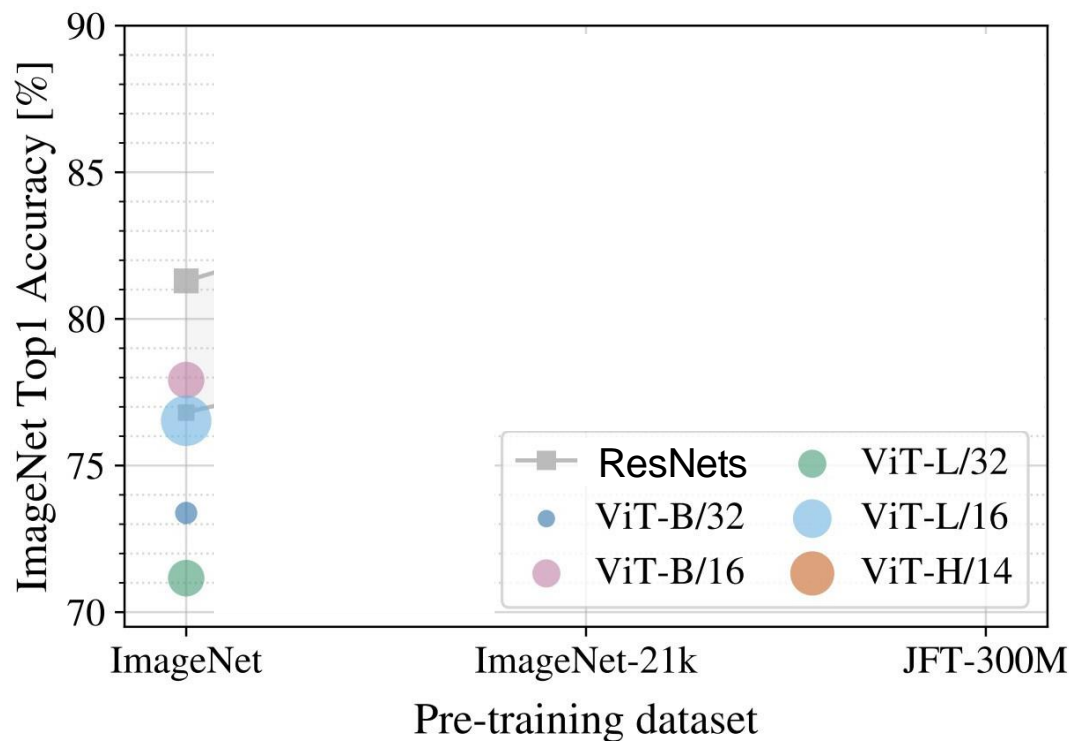
/32, /16, /14 is patch size; smaller patch size is a bigger model (more patches)

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT) vs ResNets

Recall: ImageNet dataset has 1k categories, 1.2M images

When trained on ImageNet, ViT models perform worse than ResNets



B = Base
L = Large
H = Huge

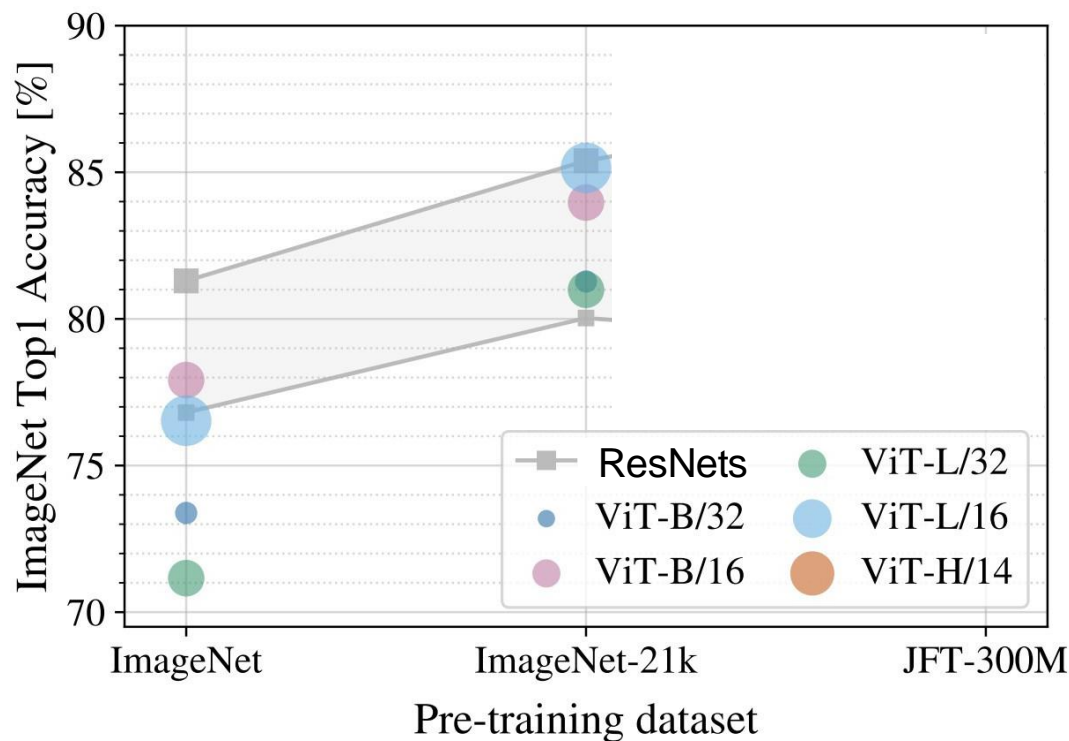
/32, /16, /14 is patch size; smaller patch size is a bigger model (more patches)

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT) vs ResNets

ImageNet-21k has 14M images with 21k categories

If you pretrain on ImageNet-21k and fine-tune on ImageNet, ViT does better: big ViTs match big ResNets



B = Base
L = Large
H = Huge

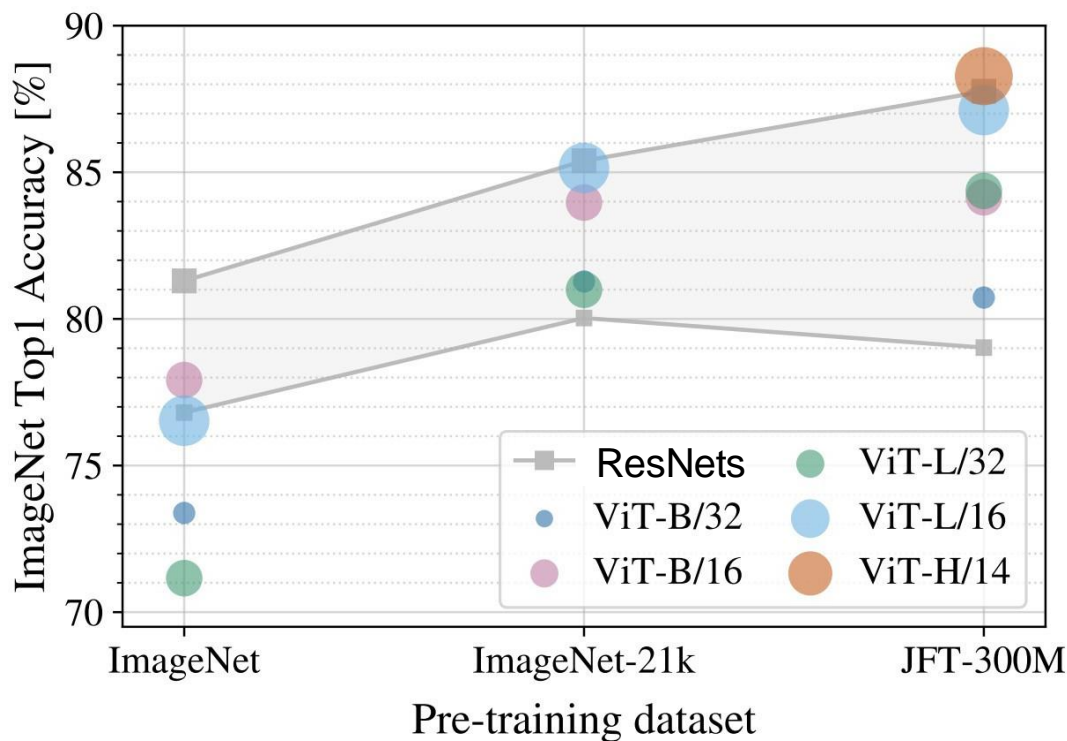
/32, /16, /14 is patch size; smaller patch size is a bigger model (more patches)

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT) vs ResNets

JFT-300M is an internal Google dataset with 300M labeled images

If you pretrain on JFT and finetune on ImageNet, large ViTs outperform large ResNets



B = Base
L = Large
H = Huge

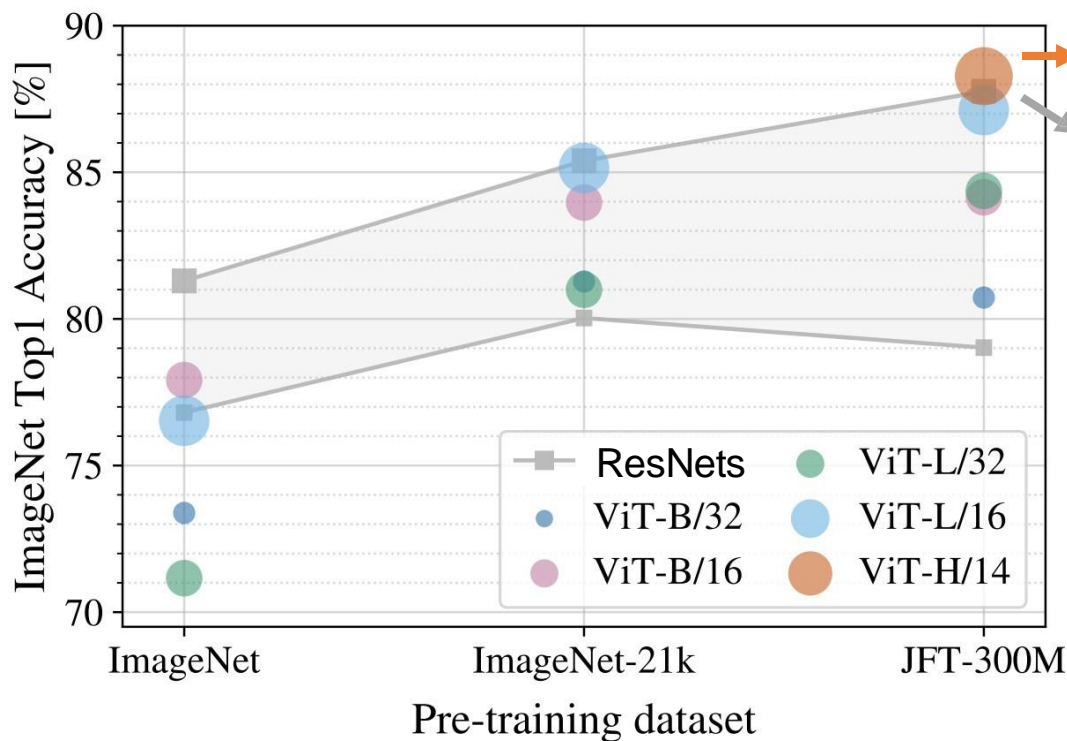
/32, /16, /14 is patch size; smaller patch size is a bigger model (more patches)

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

Vision Transformer (ViT) vs ResNets

JFT-300M is an internal Google dataset with 300M labeled images

If you pretrain on JFT and finetune on ImageNet, large ViTs outperform large ResNets



ViT: 2.5k TPU-v3 core days of training

ResNet 9.9k TPU-v3 core days of training

ViTs make more efficient use of GPU / TPU hardware (matrix multiply is more hardware-friendly than conv)

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021