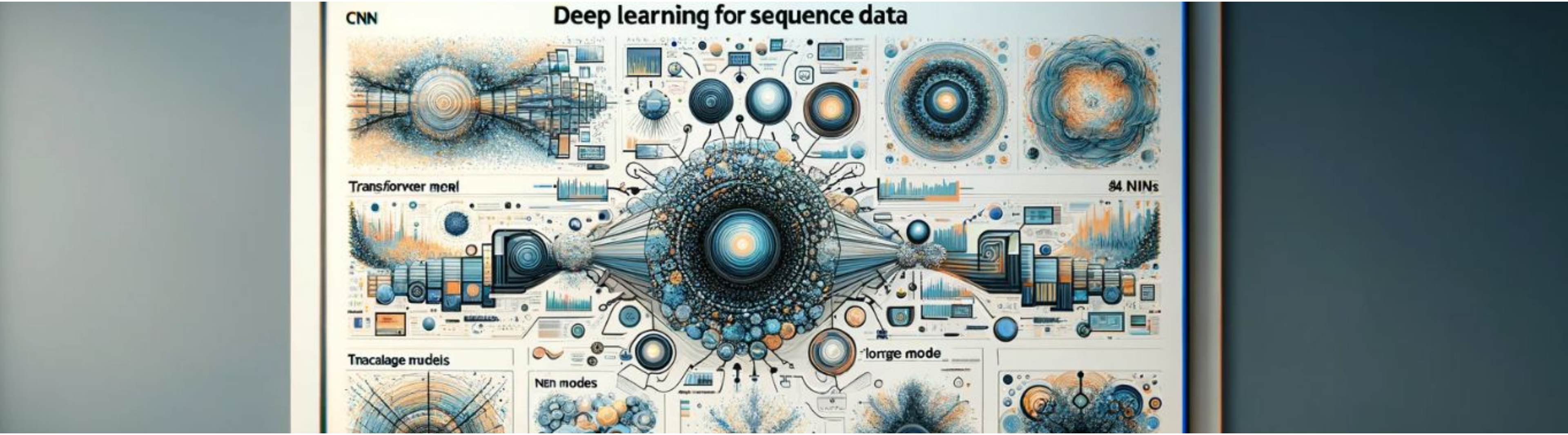
CSE7850/CX4803 Machine Learning in Computational Biology



Lecture 9: Deep Learning for Sequence Data

Yunan Luo

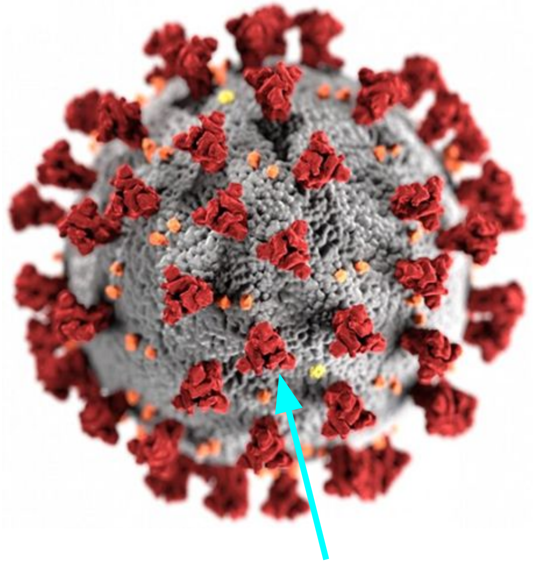
Week	Date	Topic	Contents
1	01/08	Introduction	Introduction & Logistics
1	01/10	Basics in computational biology	Molecular biology
2	01/15		No class (MLK day)
2	01/17		Sequence alignment I
3	01/22		Sequence alignment II
3	01/24	ML foundations	No Class (PyTorch video + exercise)
4	01/29		Regression & Gradient descent
4	01/31		Classification & Toolbox for Applied ML
5	02/05		Neural networks
5	02/07		Deep learning
6	02/12	Learning from sequence data	Deep learning for Protein/DNA sequences
6	02/14		Large language models (LLMs)
7	02/19	Learning from high-dim data	Clustering and dimensionality reduction
7	02/21		Generative AI
8	02/26	Learning from network data	Network basics & ML for graphs
8	02/28		Graph neural network
9	03/04	Learning from structure data	Protein structure prediction & generation (AlphaFold, diffusion models)

Today’s plan

- Biological sequences
- Deep learning for sequence data in biomedicine
  - Supervised learning
    - CNN, RNN, LSTM, Transformer
  - Self-supervised learning
    - Overview of language modeling (LLM)

Protein sequence

```
MFVFLVLLPLVSSQCVNLTTRTQLPPAYTNSFTRGVYYPDKVFRSSVLHSTQDLFLPFFS
NVTWFHAIHVSGTNGTKRFDNPVLPFNDGVYFASTEKSNIIRGWIFGTTLDSKTQSLLIV
NNATNVVIKVCEFQFCNDPFLGVYYHKNNKSWMESEFRVYSSANNCTFEYVSQPFLMDLE
GKQGNFKNLREFVFKNIDGYFKIYSKHTPINLVRDLPQGFSALEPLVDLPIGINITRFQT
LLALHRSYLTPGDSSSGWTAGAAAYYVGYLQPRTFLLKYNENGTITDAVDCALDPLSETK
CTLKSFTVEKGIYQTSNFRVQPTESIVRFPNITNLCPFGEVFNATRFASVYAWNRKRISN
CVADYSVLYNSASFSTFKCYGVSPTKLNDLCFTNVYADSFVIRGDEVRQIAPGQTGKIAD
YNYKLPDDFTGCVIAWNSNNLDSKVGGNYNYLYRLFRKSNLKPFERDISTEIYQAGSTPC
NGVEGFNCYFPLQSYGFQPTNGVGYQPYRVVVLSFELLHAPATVCGPKKSTNLVKNKCVN
FNFNGLTGTGVLTESNKKFLPFQQFGRDIADTTDAVRDPQTLEILDITPCSFGGVSVITP
GTNTSNQVAVLYQDVNCTEVPVAIHADQLTPTWRVYSTGSNVFQTRAGCLIGAEHVNNSY
ECDIPIGAGICASYQTQTNSPRRARSVASQSIIAYTMSLGAENSVAYSNNSIAIPTNFTI
SVTTEILPVSMTKTSVDCTMYICGDSTECSNLLLQYGSFCTQLNRALTGIAVEQDKNTQE
VFAQVKQIYKTPPIKDFGGFNFSQILPDPSKPSKRSFIEDLLFNKVTLADAGFIKQYGDC
LGDIAARDLICAQKFNGLTVLPPLLTDEMIAQYTSALLAGTITSGWTFGAGAALQIPFAM
QMAYRFNGIGVTQNVLYENQKLIANQFNSAIGKIQDSLSSTASALGKLQDVVNQNAQALN
TLVKQLSSNFGAISSVLNDILSRLDKVEAEVQIDRLITGRLQSLQTYVTQQLIRAAEIRA
SANLAATKMSECVLGQSKRVDFCGKGYHLMSFPQSAPHGVVFLHVTYVPAQEKNFTTAPA
ICHDGKAHFPREGVFVSNGTHWFVTQRNFYEPQIITTDNTFVSGNCDVVIGIVNNTVYDP
LQPELDSFKEELDKYFKNHTSPDVDLGDISGINASVVNIQKEIDRLNEVAKNLNESLIDL
QELGKYEQYIKWPWYIWLGFIAGLIAIVMVTIMLCCMTSCCSCLKGCCSCGSCCKFDEDD
```



SARS-CoV-2 spike protein

Summary: biological sequences

- DNA = nucleotide sequence
    - Alphabet size = 4 (A,C,G,T)
- RNA (single stranded)
    - Alphabet size = 4 (A,C,G,U)
- Protein sequence
    - Alphabet size = 20

Deep learning for sequence data

Problem formulation

Regression/Classification

Sequence → ML model → Label

Example: protein stability prediction

Input	Output
…DNGVDGEWTYDDATKTFTVTE	1.0
…DNGCDGEWTYDDATKTFTVTE	-0.2
…DNGVWGEWTYDDATKTFTVTE	3.9
…DNGVWGEWTYDDATKTFTFTE	5.4
…DNGVMGEWTYDDATKTFTDTE	-0.1

Sequence encoding

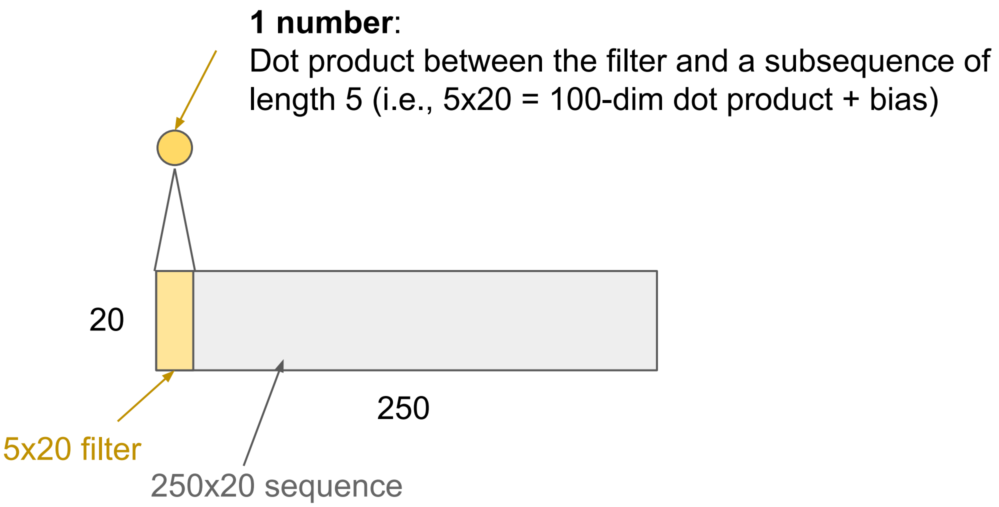
- one-hot encoding

	C	G	A	T	A	A	C	C	G	A	T	A	T
A	0	0	1	0	1	1	0	0	0	1	0	1	0
C	1	0	0	0	0	0	1	1	0	0	0	0	0
G	0	1	0	0	0	0	0	0	1	0	0	0	0
T	0	0	0	1	0	0	0	0	0	0	1	0	1

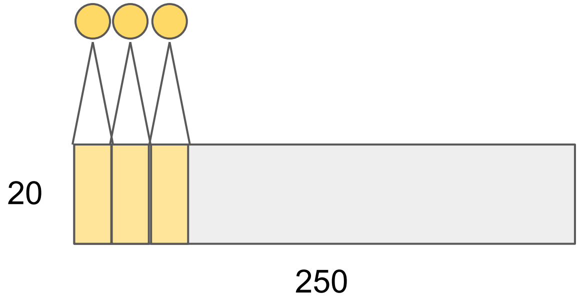
- contextual embedding (language models)
    - Rives, Alexander, et al. "Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences." *Proceedings of the National Academy of Sciences* 118.15 (2021).

Model 1:
Convolutional Neural Network (CNN)

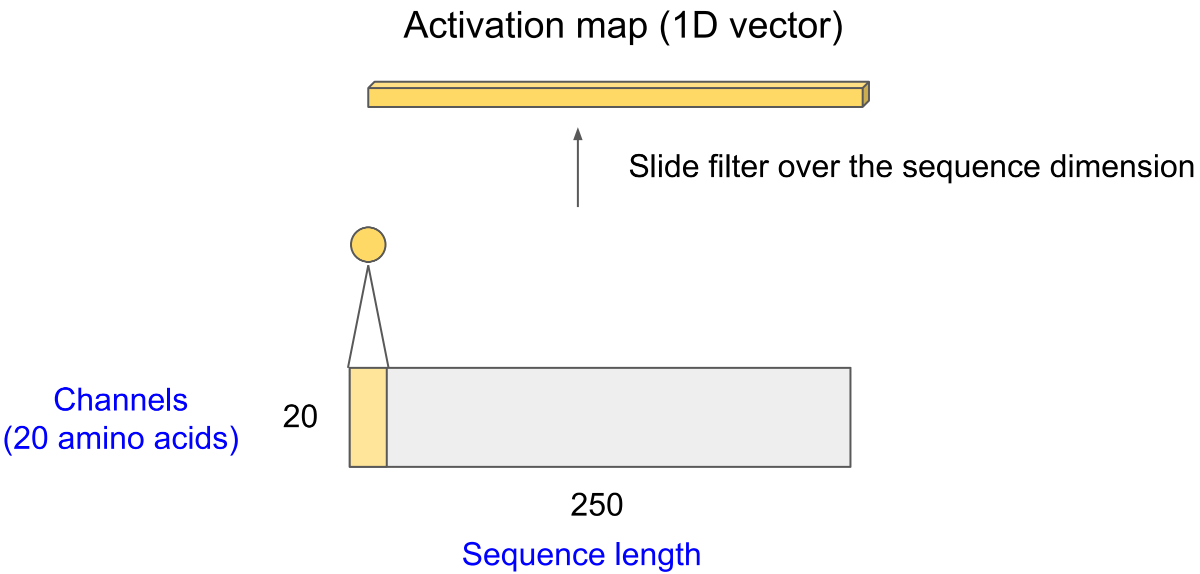
1D convolution



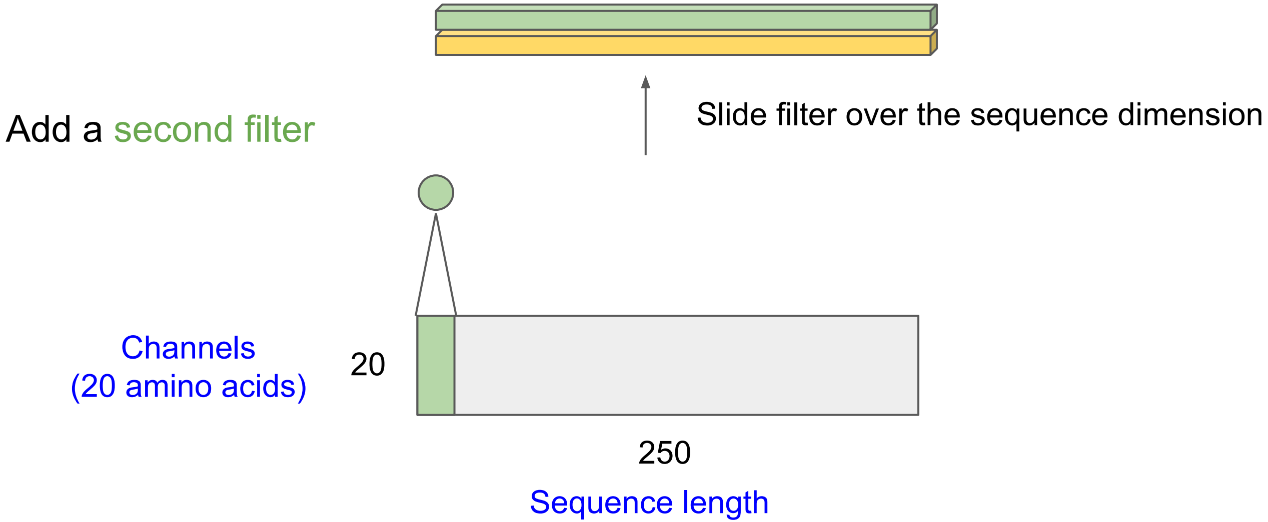
1D convolution



1D convolution



1D convolution

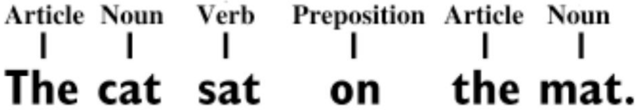


Model 2:
Recurrent Neural Network (RNN)

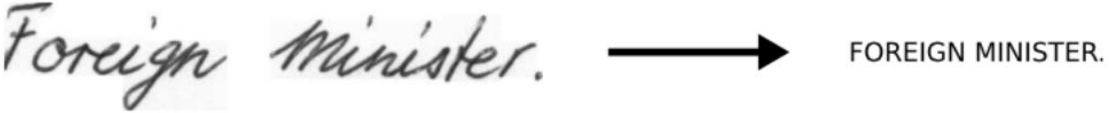
Model #2: Recurrent Neural Network (RNN)

Example: sequence labeling problems

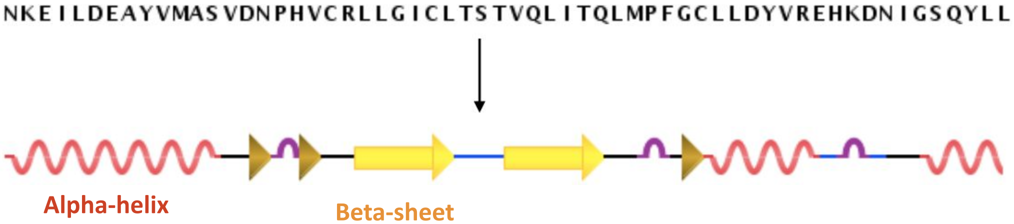
- Part of speech

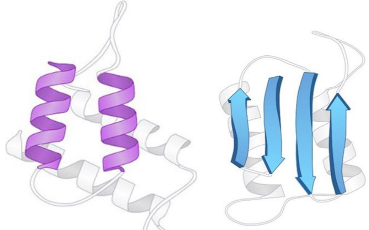


- Handwriting recognition



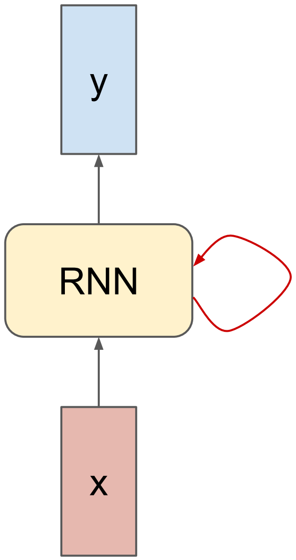
- Protein secondary structure prediction





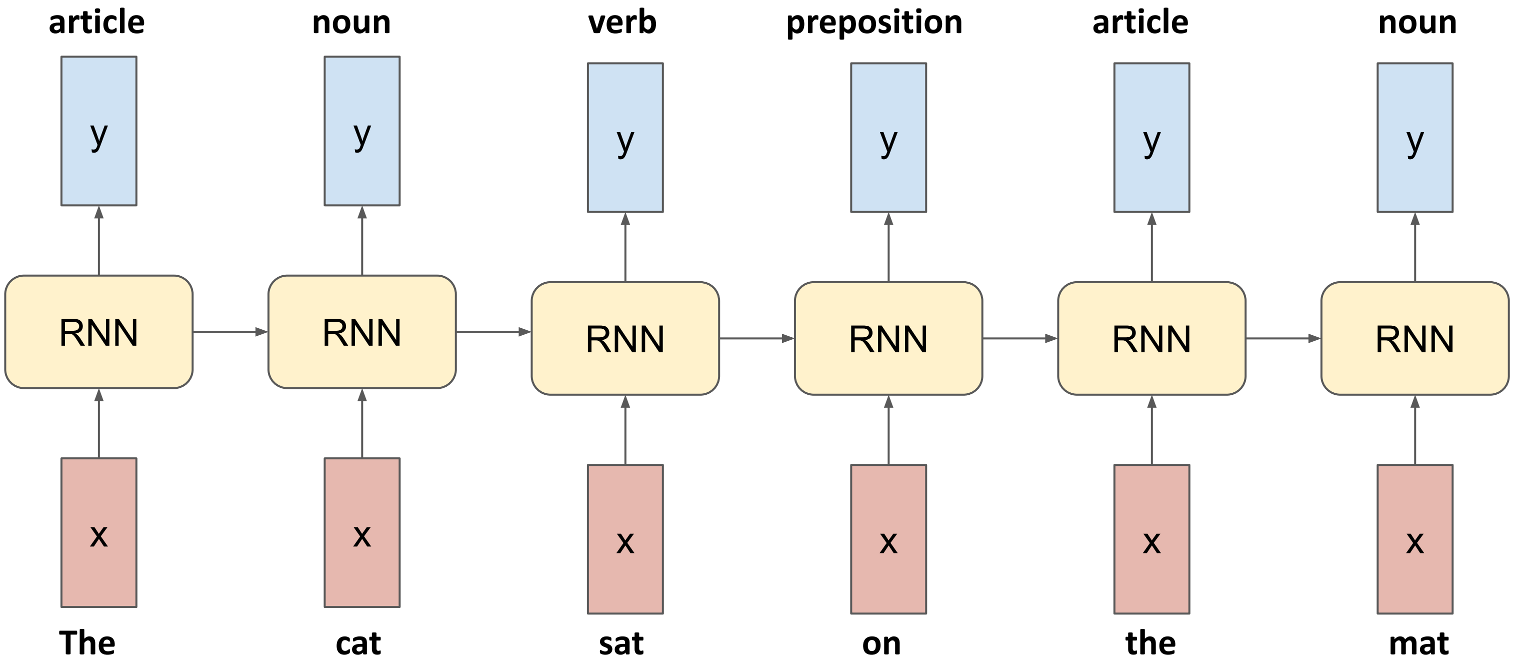
Alpha-helix Beta-sheet

Recurrent Neural Network (RNN)

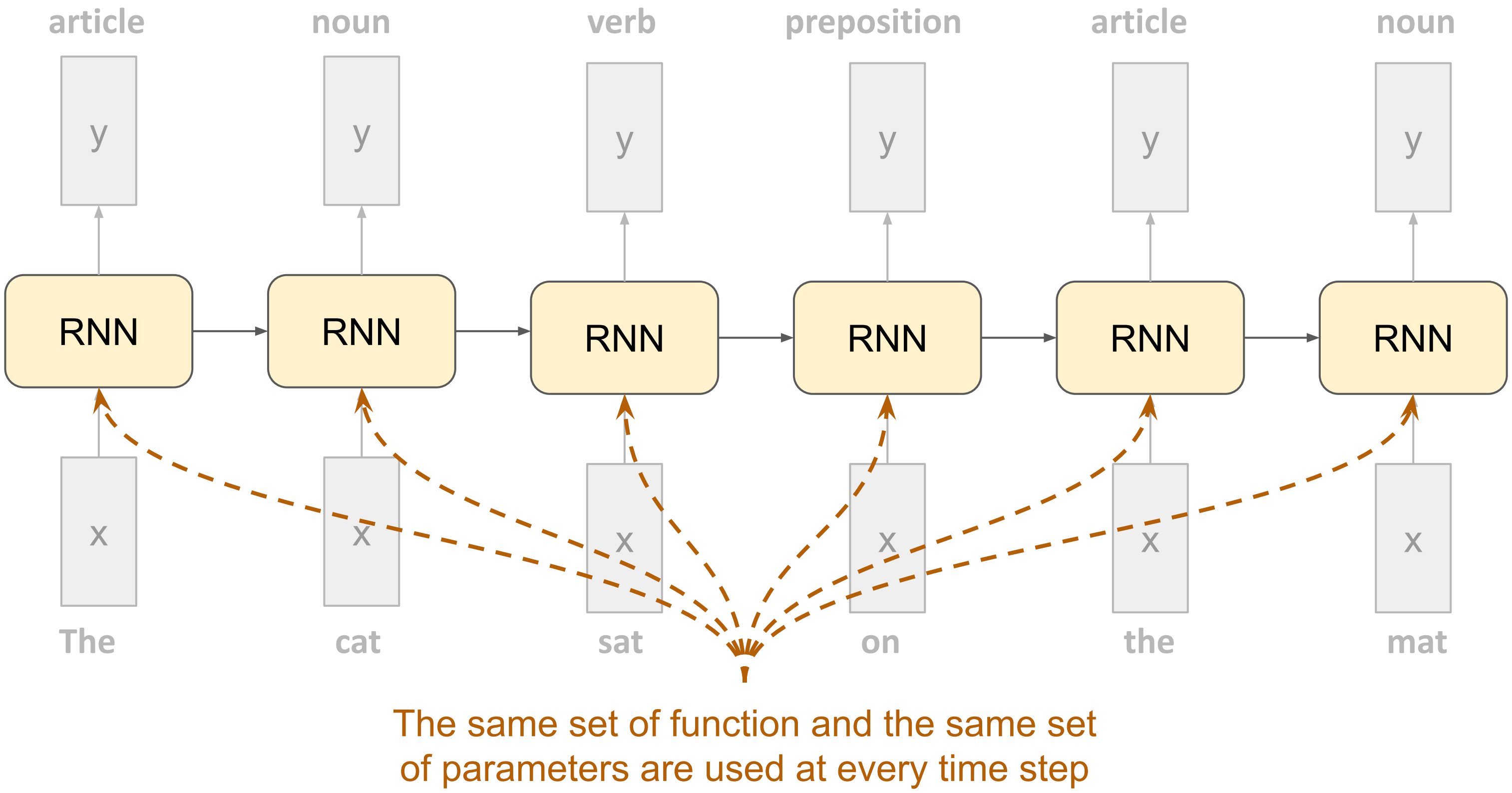


Key idea: RNNs have an “internal state” that is updated as the input sequence is processed

Unrolled RNN



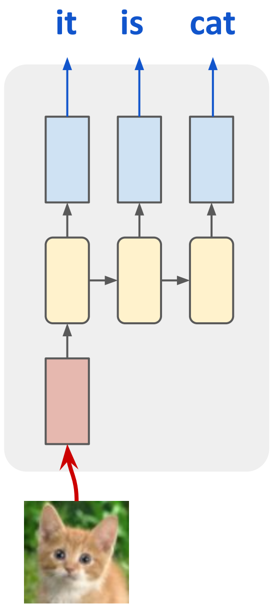
Unrolled RNN



RNNs for sequence processing

one to many

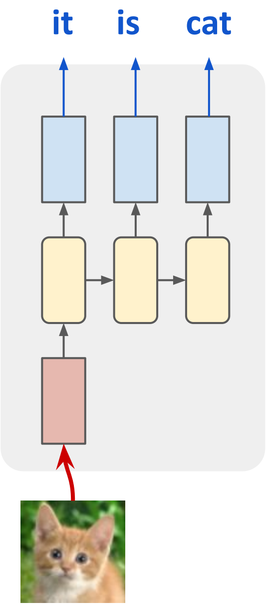
(e.g., image captioning)



RNNs for sequence processing

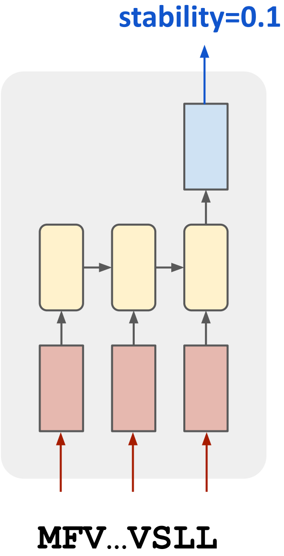
one to many

(e.g., image captioning)



many to one

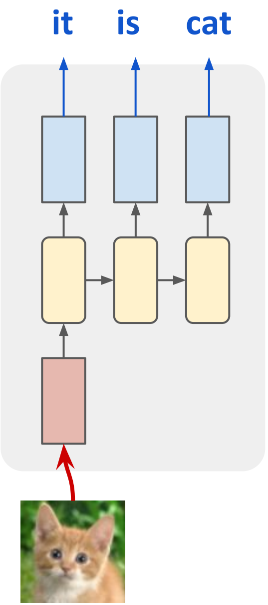
(e.g., protein function prediction)



RNNs for sequence processing

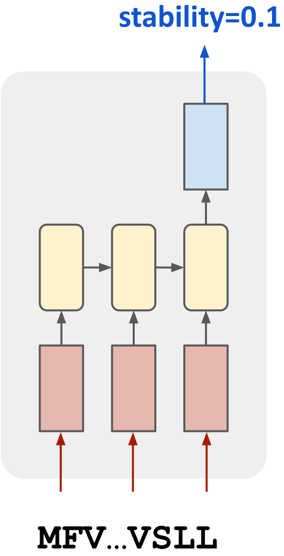
one to many

(e.g., image captioning)



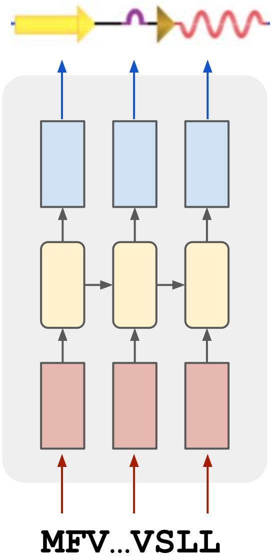
many to one

(e.g., protein function prediction)



many to many

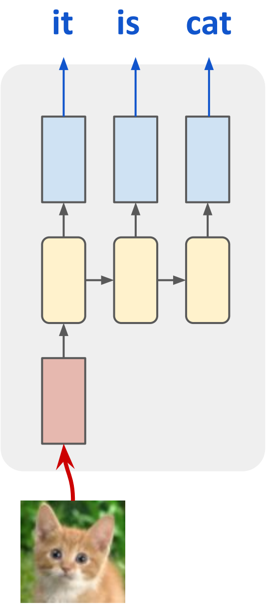
(e.g., protein structure prediction)



RNNs for sequence processing

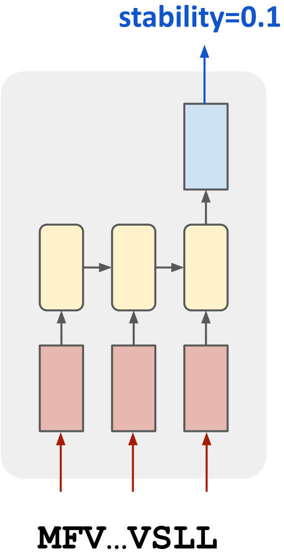
one to many

(e.g., image captioning)



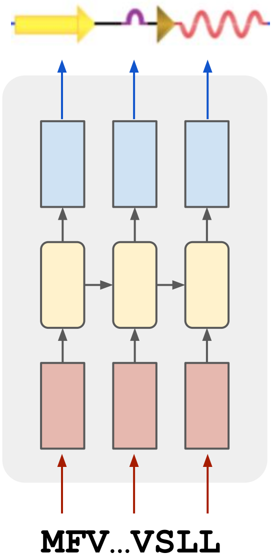
many to one

(e.g., protein function prediction)



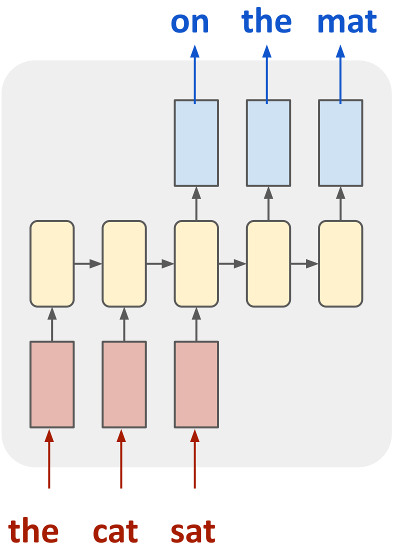
many to many

(e.g., protein structure prediction)

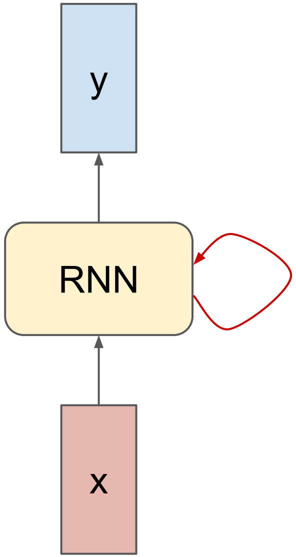


many to many

(e.g., auto completion)



RNN hidden state update



At every step t, the hidden state is updated based on the previous state and the current input

function with parameter W

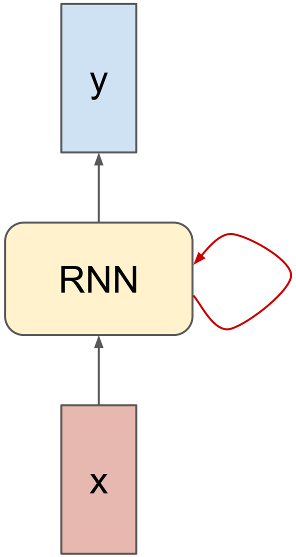
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state

previous state

Input vector at time step t

RNN output



At every step t, the output is generated based on the current state

Another function with parameter O

$$\boxed{y_t} = \boxed{f_o}(\boxed{h_t})$$

new state

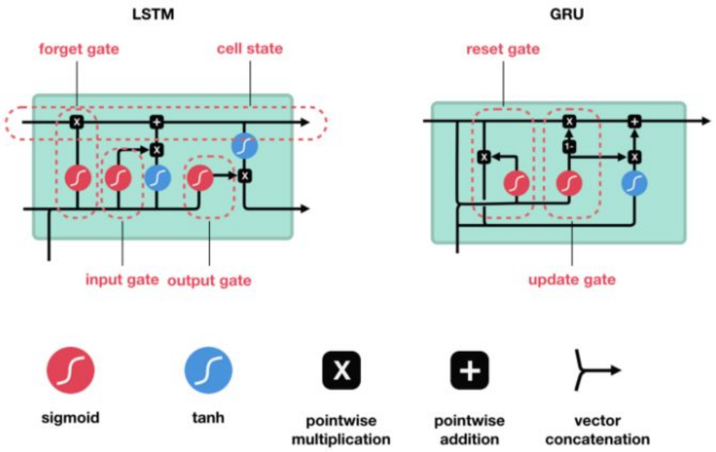
new state

Further readings of RNN

- *Deep Learning*, Chapter 6
- What is function *f*()?
  - f() is usually called "unit" in RNN
  - It defines a "computational graph" the produces h_t based on h_{t-1} and x_t
- Popular RNN variants
  - Long short-term memory (LSTM)
  - Gated recurrent unit (GRU)

function with parameter *W*

$$h_t = \boxed{f_W}(h_{t-1}, x_t)$$



(image source)

Summary: key ideas of RNNs

- Process sequence data with **variable lengths**
    - DNA/RNA/protein sequences, text, audio, time series data
- Capture **sequential** (temporal) information/dependencies in the data
- Parameters **shared** over time steps
- Common to use LSTM or GRU

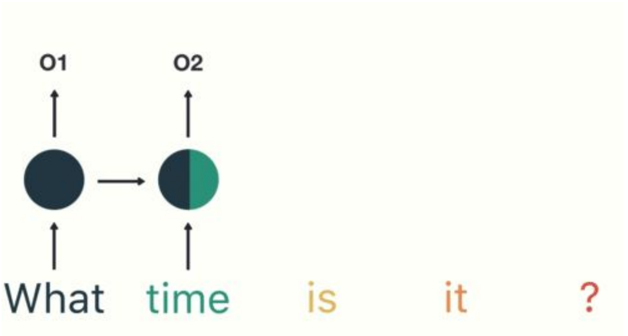


Image from Google

Model 3:
Transformers

Model #3: Transformers

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

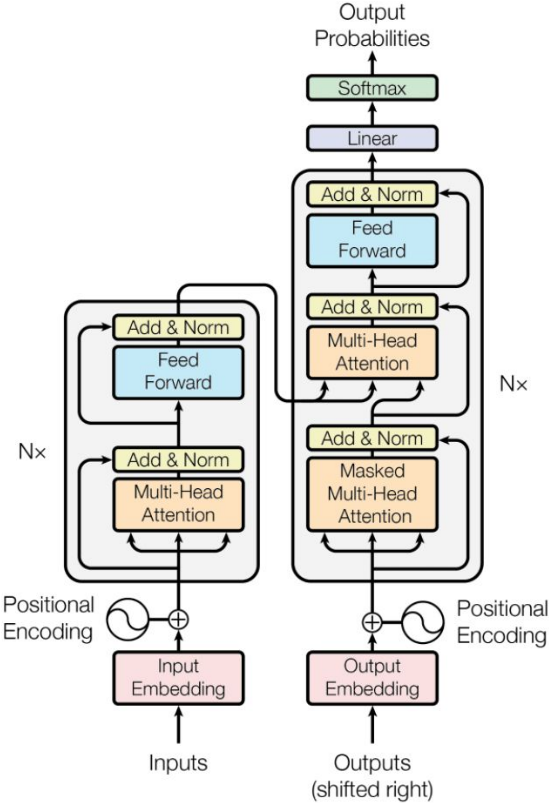
Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

(NeurIPS 2017)

(Citation as of 02/11/2024)



- Encoder-Decoder
- Sequence-to-sequence
- Transforms one sequence into another sequence, using full context of each

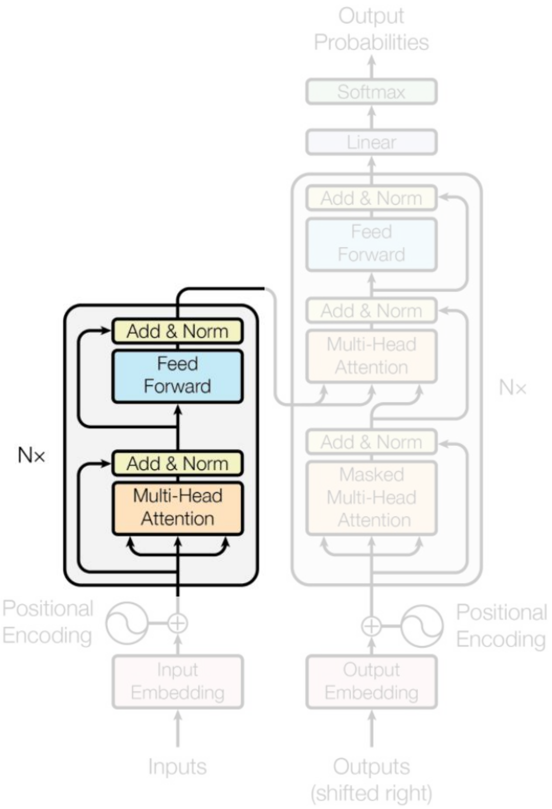
Building blocks of Transformer

N blocks, each has

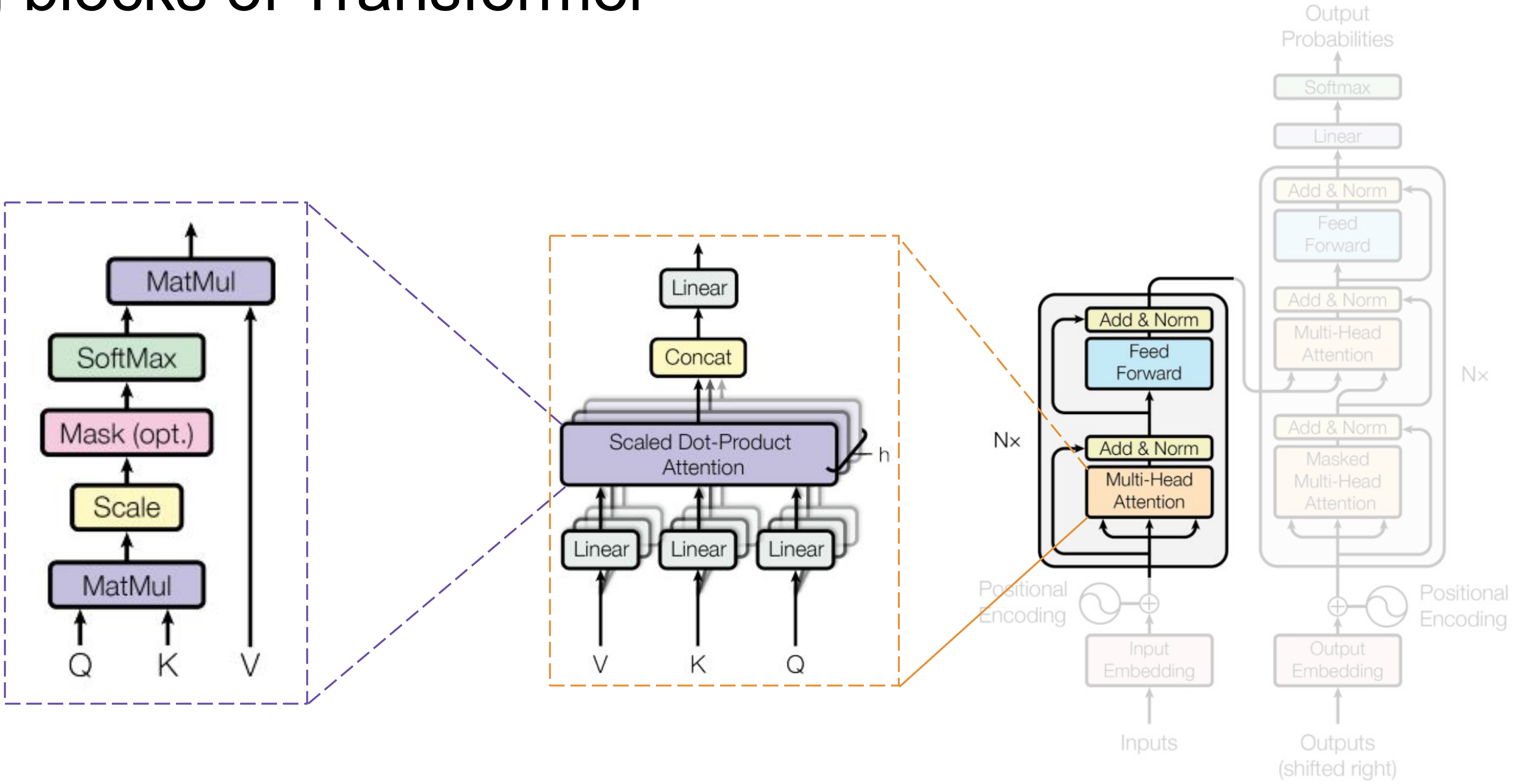
- Multi-head self-attention layer
- Two-layer feed-forward neural nets

Residual connection and layer normalization are used

- Reading: LayerNorm (https://arxiv.org/pdf/1607.06450.pdf)



Building blocks of Transformer



Key ideas: self-attention layer

- **Attention layer**: a layer to learn the dependency between words in the input. The dependency is quantified using “attention weights”
- For each word, a new representation is computed by weight-averaging the old representations of all words, where the weight is the learned attention weight

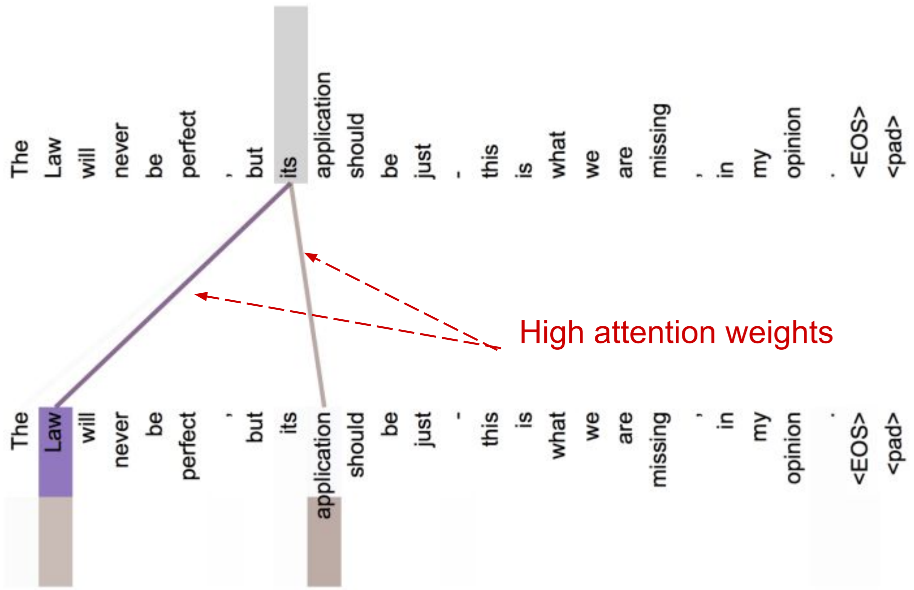


Image credit: CS224N, Stanford

Key ideas: self-attention layer

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

Q: “query” matrix, a vector representation for each word
K: “key” matrix, a vector representation for each word
V: “value” matrix, a vector representation for each word

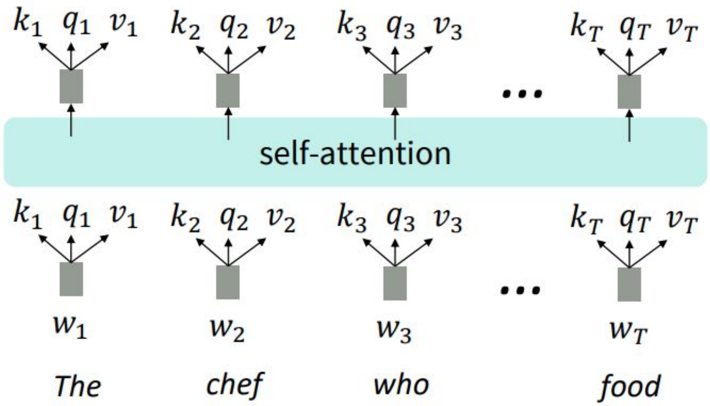
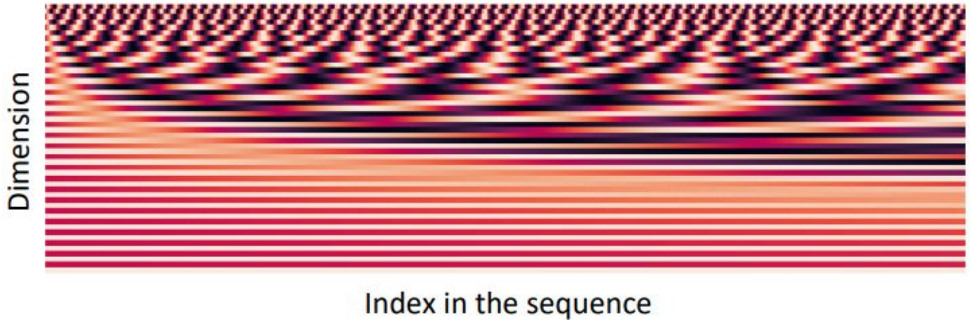


Image credit: CS224N, Stanford

Key ideas: positional encoding

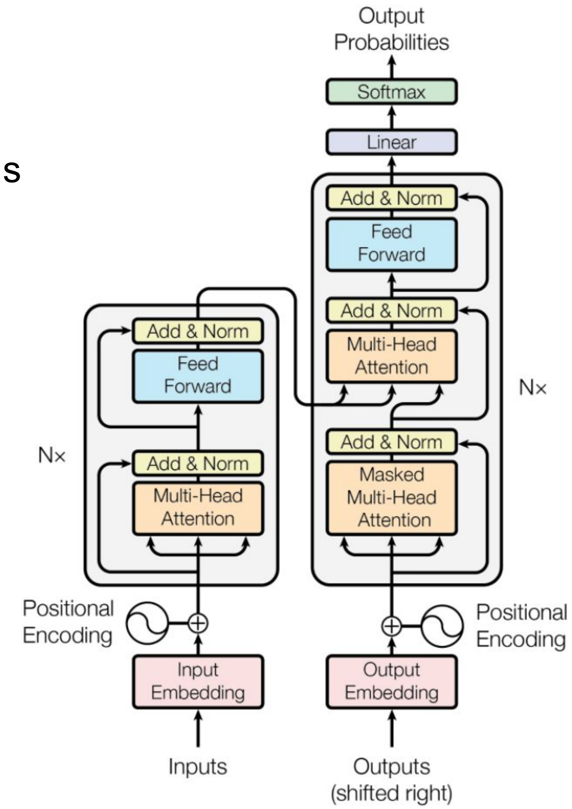
- Self-attention does not know the order of input words
- Positional encodings are added to the word representations, so same words at different locations have different overall representations

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



Summary of Transformer

- Learning temporal relationships without unrolling and without RNNs
- Encoder/Decoder framework, multi-head self-attention modules
- Widely used in state-of-the-art NLP models
- Readings:
    - “Attention is all you need” (https://arxiv.org/abs/1706.03762)
    - PyTorch implementation and tutorial of Transformer



Demo: DL for sequence data

Google Colab

Exercise

- The model in the Colab Notebook was implemented in Tensorflow Keras. As an exercise, re-implement the model in PyTorch
- The model in the Colab Notebook was a CNN. Implement a different neural network (RNN, Transformer), then train and test it.

Overview of Large Language Models (LLMs)

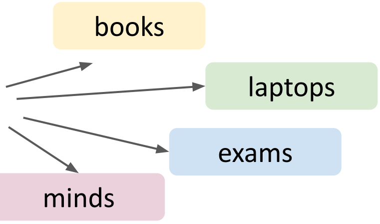
Outline

- **What is LLM?**
- Model architectures & (Pre-)Training
- From language modeling to ChatGPT (next lecture)
- LLM for biological sequence (next lecture)

Language modeling in natural language

- **Language modeling** is the task of predicting what word comes next

"The students opened their ________"



- More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} \mid \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|V|}\}$.

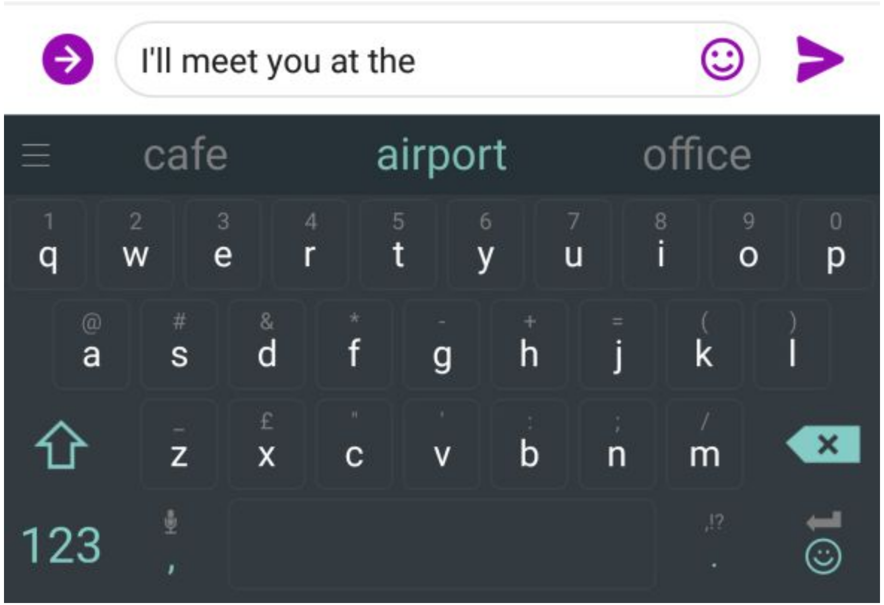
- A system that does this is called a **language model (LM)**

Language Modeling

- You can also think of a Language Model as a system that assigns a probability to a piece of text
- For example, if we have some text $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}$, then the probability of this text (according to the Language Model) is:

$$
\begin{aligned}
P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) &= P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \, \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \, \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)}) \\
&= \prod_{t=1}^{T} \underbrace{P(\boldsymbol{x}^{(t)} | \, \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)})}_{\text{This is what our LM provides}}
\end{aligned}
$$

You use LM every day!

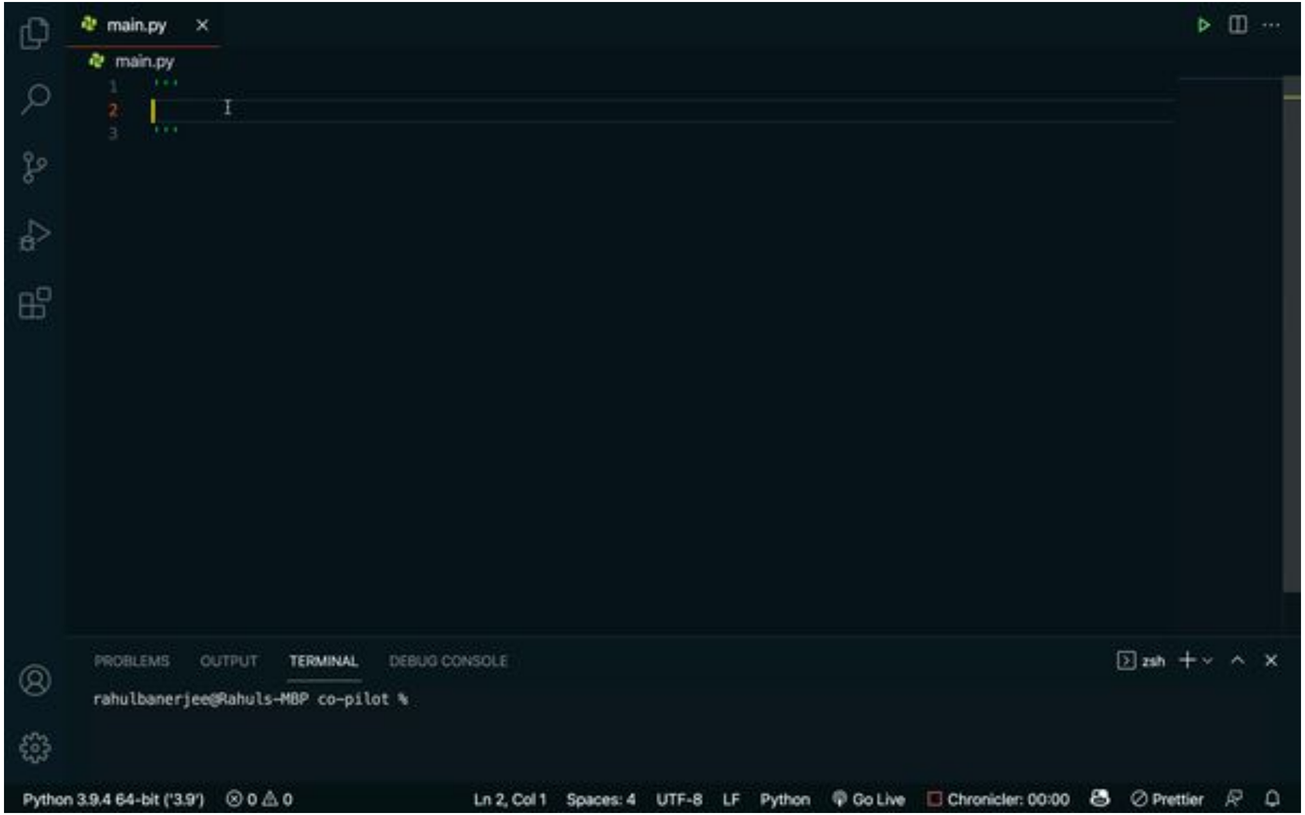


Adapted from CS224N, Stanford

You use LM every day!

Adapted from CS224N, Stanford

You use LM every day!



GitHub Copilot

https://copilot.github.com/

n-gram Language Models

the students opened their ______

- **Question**: How to learn a Language Model?
- **Answer** (pre- Deep Learning): learn an *n*-gram Language Model!

- **Definition:** An *n*-gram is a chunk of *n* consecutive words.
  - unigrams: "the", "students", "opened", "their"
  - bigrams: "the students", "students opened", "opened their"
  - trigrams: "the students opened", "students opened their"
  - four-grams: "the students opened their"

- **Idea:** Collect statistics about how frequent different n-grams are and use these to predict next word.

n-gram Language Models

- First we make a Markov assumption: $x^{(t+1)}$ depends only on the preceding *n*-1 words

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}})$$

(assumption)

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{\boxed{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}}{\boxed{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}}$$

(definition of conditional prob)

- **Question:** How do we get these *n*-gram and (*n*-1)-gram probabilities?
- **Answer:** By counting them in some large corpus of text!

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}$$

(statistical approximation)

n-gram Language Models: Example

Suppose we are learning a 4-gram Language Model.

~~*as the proctor started the clock, the*~~ $\underbrace{\textit{students opened their}}_{\text{condition on this}}$ ______

discard

condition on this

$$P(\boldsymbol{w}|\text{students opened their}) = \frac{\text{count(students opened their } \boldsymbol{w})}{\text{count(students opened their)}}$$

For example, suppose that in the corpus:

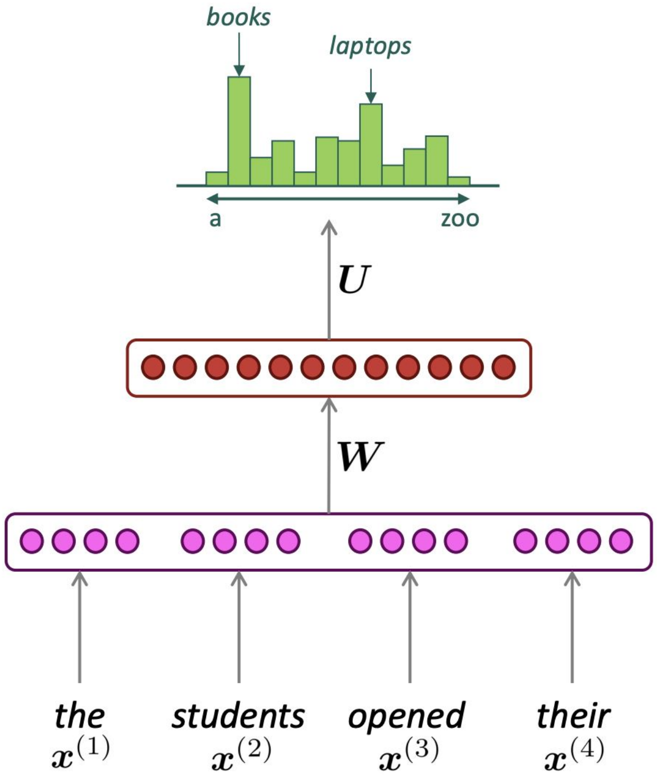
- "students opened their" occurred 1000 times
- "students opened their books" occurred 400 times
  - → P(books | students opened their) = 0.4
- "students opened their exams" occurred 100 times
  - → P(exams | students opened their) = 0.1

How to build a *neural* language model?

- Recall the Language Modeling task:
  - Input: sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$
  - Output: prob. dist. of the next word $P(\boldsymbol{x}^{(t+1)} | \, \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$

~~*as the proctor started the clock*~~ $\underbrace{\textit{the students opened their}}_{\text{fixed window}}$ ______

discard



We can mask everywhere in a sentence

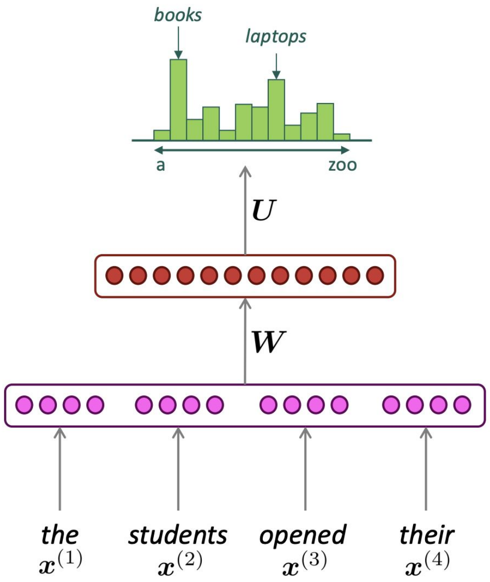
- Autoregressive language models (predict the next word given preceding words)

  “The students opened their ________”

- Masked language models (predict the masked word given surrounding words)

  “The students ________ their book”

- Training process of language models (self-supervised training)
  - Randomly mask one or more words in a given sentence (e.g., from Wikipedia)
  - Train the LM (a neural network) to predict the correct word for the masked positions



What can we learn from reconstructing the input?

Georgia Institute of Technology is located in ___________, Georgia.

What can we learn from reconstructing the input?

I put ___ fork down on the table.

What can we learn from reconstructing the input?

The woman walked across the street,
checking for traffic over ___ shoulder.

What can we learn from reconstructing the input?

I went to the ocean to see the fish, turtles, seals, and _____.

What can we learn from reconstructing the input?

Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was ___.

What can we learn from reconstructing the input?

I was thinking about the sequence that goes
1, 1, 2, 3, 5, 8, 13, 21, ____

Outline

- What is LLM?
- **Model architectures & (Pre-)Training**
- From language modeling to ChatGPT (next lecture)
- LLM for biological sequence (next lecture)

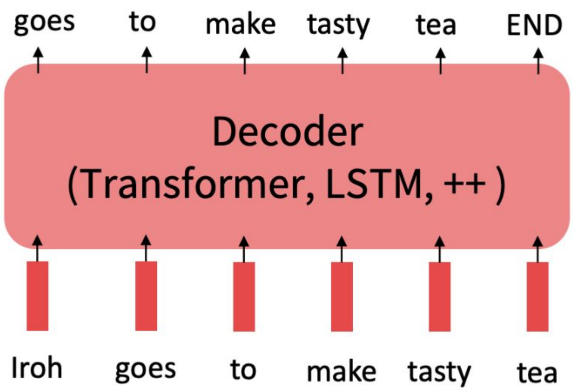
Pretraining through language modeling

Recall the **language modeling** task:

- Model $p_\theta(w_t | w_{1:t-1})$, the probability distribution over words given their past contexts.
- There's lots of data for this! (In English.)

Pretraining through language modeling:

- Train a neural network to perform language modeling on a large amount of text.
- Save the network parameters.

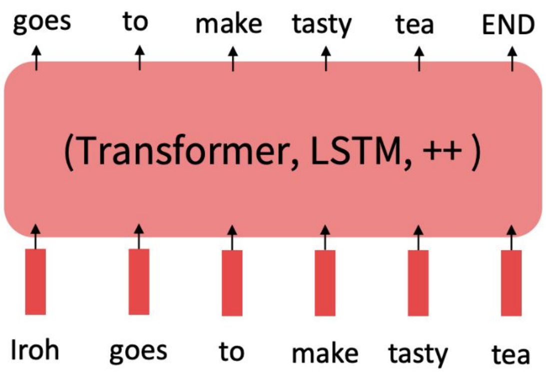


The Pretraining / Finetuning Paradigm

Pretraining can improve NLP applications by serving as parameter initialization.

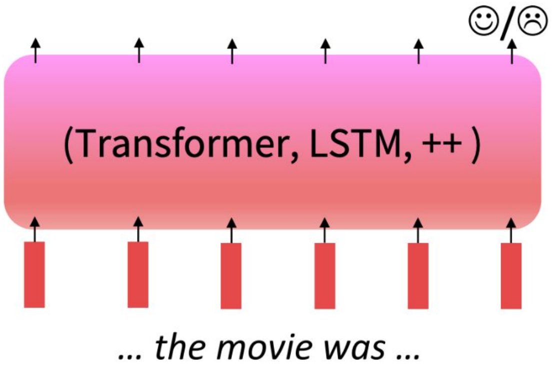
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



Step 2: Finetune (on your task)

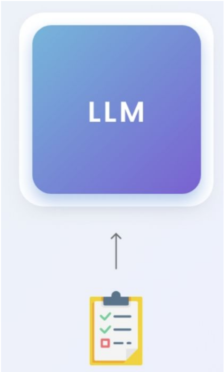
Not many labels; adapt to the task!



Pre-training on related unlabeled data helps!

Task: Train an LM to generate product review

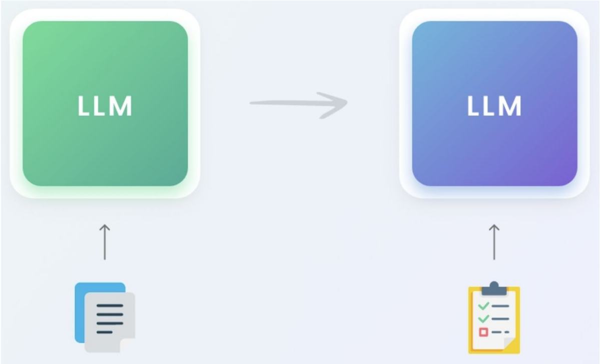
"Conventional" approach



Small-scale data (e.g., Amazon product reviews)

Likely to generate low-quality texts, with grammar or semantic errors

Pre-training & fine-tuning paradigm



Large-scale related data (e.g., Wikipedia articles)

First learn how to write in English, without grammar or semantic errors

Small-scale data (e.g., Amazon product reviews)

Then adapt the "writing skills" to write special-purpose texts

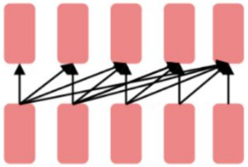
Stochastic gradient descent and pretrain/finetune

Why should pretraining and finetuning help, from a "training neural nets" perspective?

- Consider, provides parameters $\hat{\theta}$ by approximating $\min_{\theta} \mathcal{L}_{\text{pretrain}}(\theta)$.
  - (The pretraining loss.)
- Then, finetuning approximates $\min_{\theta} \mathcal{L}_{\text{finetune}}(\theta)$, starting at $\hat{\theta}$.
  - (The finetuning loss)
- The pretraining may matter because stochastic gradient descent sticks (relatively) close to $\hat{\theta}$ during finetuning.
  - So, maybe the finetuning local minima near $\hat{\theta}$ tend to generalize well!
  - And/or, maybe the gradients of finetuning loss near $\hat{\theta}$ propagate nicely!

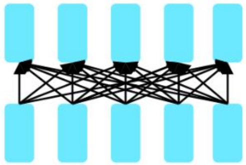
Three types of LM architectures

The neural architecture influences the type of pretrianing, and natural use cases



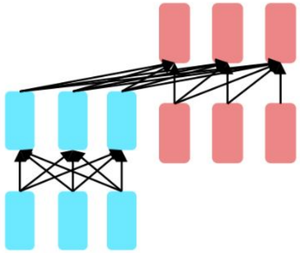
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words



Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?

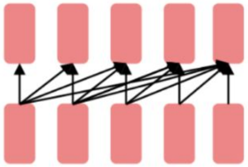


Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

Three types of LM architectures

The neural architecture influences the type of pretrianing, and natural use cases



Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?

Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

Pretraining decoders

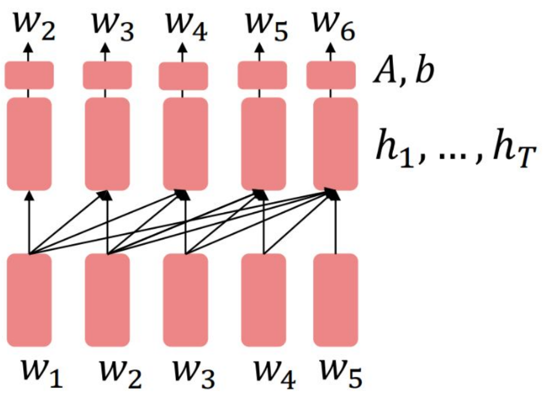
It's natural to pretrain decoders as language models and then use them as generators, finetuning their $p_\theta(w_t|w_{1:t-1})$!

This is helpful in tasks **where the output is a sequence** with a vocabulary like that at pretraining time!

- Dialogue (context=dialogue history)
- Summarization (context=document)

$$h_1, \dots, h_T = \text{Decoder}(w_1, \dots, w_T)$$
$$w_t \sim Ah_{t-1} + b$$

Where A, b were pretrained in the language model!



[Note how the linear layer has been pretrained.]

Pretraining decoders

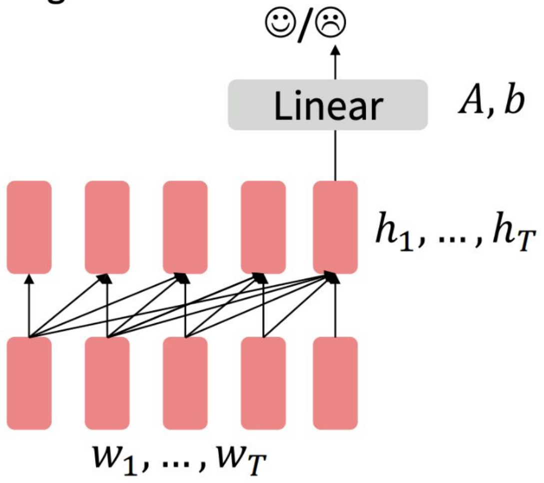
When using language model pretrained decoders, we can ignore that they were trained to model $p(w_t|w_{1:t-1})$.

We can finetune them by training a classifier on the last word's hidden state.

$$h_1, \dots, h_T = \text{Decoder}(w_1, \dots, w_T)$$
$$y \sim Ah_T + b$$

Where A and b are randomly initialized and specified by the downstream task.

Gradients backpropagate through the whole network.



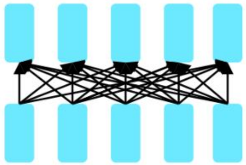
[Note how the linear layer hasn't been pretrained and must be learned from scratch.]

Three types of LM architectures

The neural architecture influences the type of pretrianing, and natural use cases

Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words



Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?

Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

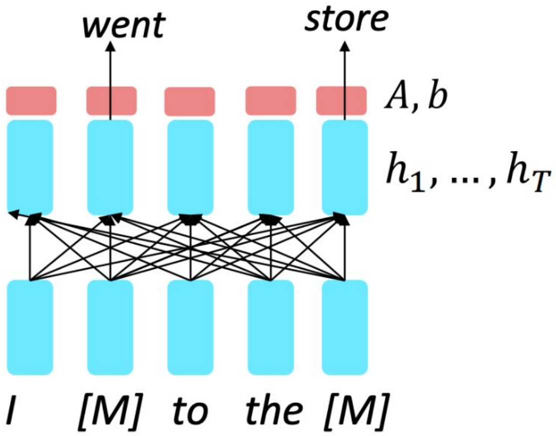
Pretraining encoders: what pretraining objective to use?

So far, we've looked at language model pretraining. But **encoders get bidirectional context,** so we can't do language modeling!

Idea: replace some fraction of words in the input with a special [MASK] token; predict these words.

$$h_1, \dots, h_T = \text{Encoder}(w_1, \dots, w_T)$$
$$y_i \sim Aw_i + b$$

Only add loss terms from words that are "masked out." If $\tilde{x}$ is the masked version of x, we're learning $p_\theta(x|\tilde{x})$. Called **Masked LM.**



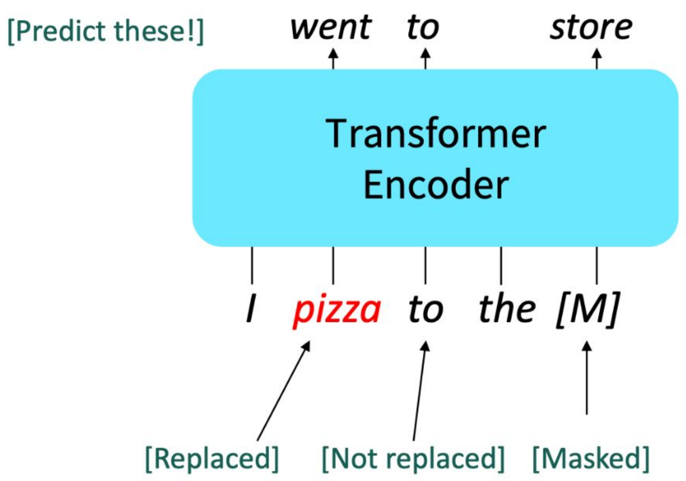
[Devlin et al., 2018]

BERT: Bidirectional Encoder Representations from Transformers

Devlin et al., 2018 proposed the "Masked LM" objective and **released the weights of a pretrained Transformer,** a model they labeled BERT.

Some more details about Masked LM for BERT:

- Predict a random 15% of (sub)word tokens.
  - Replace input word with [MASK] 80% of the time
  - Replace input word with a random token 10% of the time
  - Leave input word unchanged 10% of the time (but still predict it!)
- Why? Doesn't let the model get complacent and not build strong representations of non-masked words. (No masks are seen at fine-tuning time!)



[Devlin et al., 2018]

Three types of LM architectures

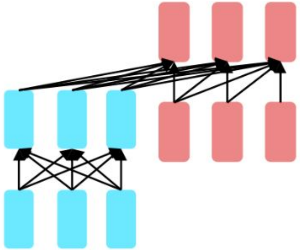
The neural architecture influences the type of pretrianing, and natural use cases

Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



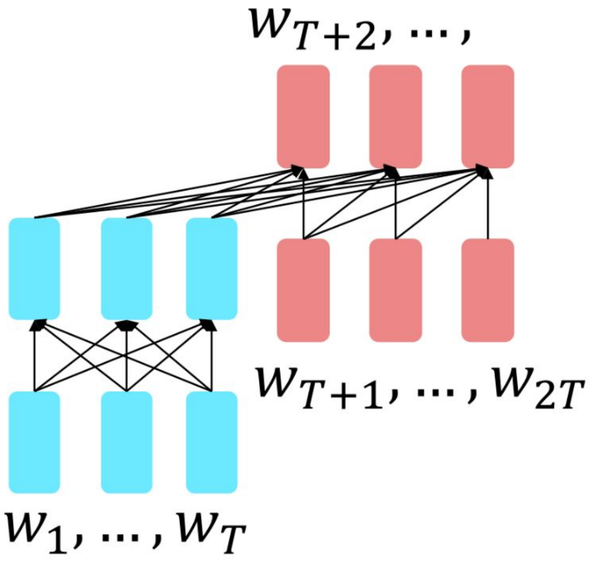
Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

Pretraining encoder-decoders: what pretraining objective to use?

For **encoder-decoders**, we could do something like **language modeling**, but where a prefix of every input is provided to the encoder and is not predicted.

$$h_1, \dots, h_T = \text{Encoder}(w_1, \dots, w_T)$$
$$h_{T+1}, \dots, h_2 = Decoder(w_1, \dots, w_T, h_1, \dots, h_T)$$
$$y_i \sim Ah_i + b, i > T$$

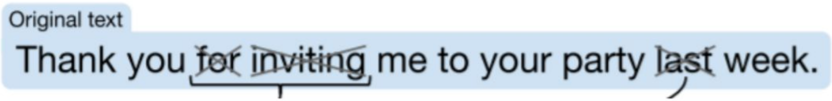


The **encoder** portion benefits from bidirectional context; the **decoder** portion is used to train the whole model through language modeling.

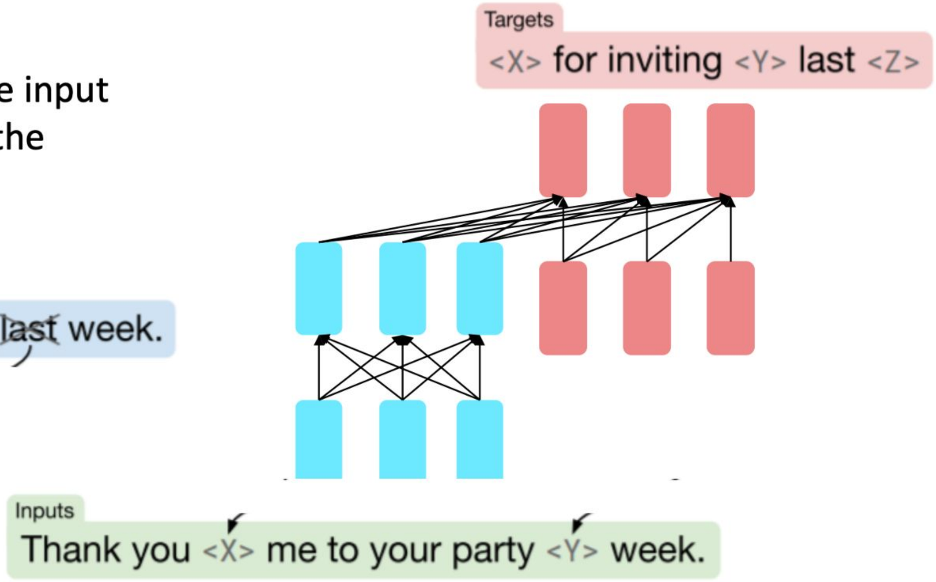
[Raffel et al., 2018]

T5: Denoising the span corruption

Replace different-length spans from the input with unique placeholders; decode out the spans that were removed!

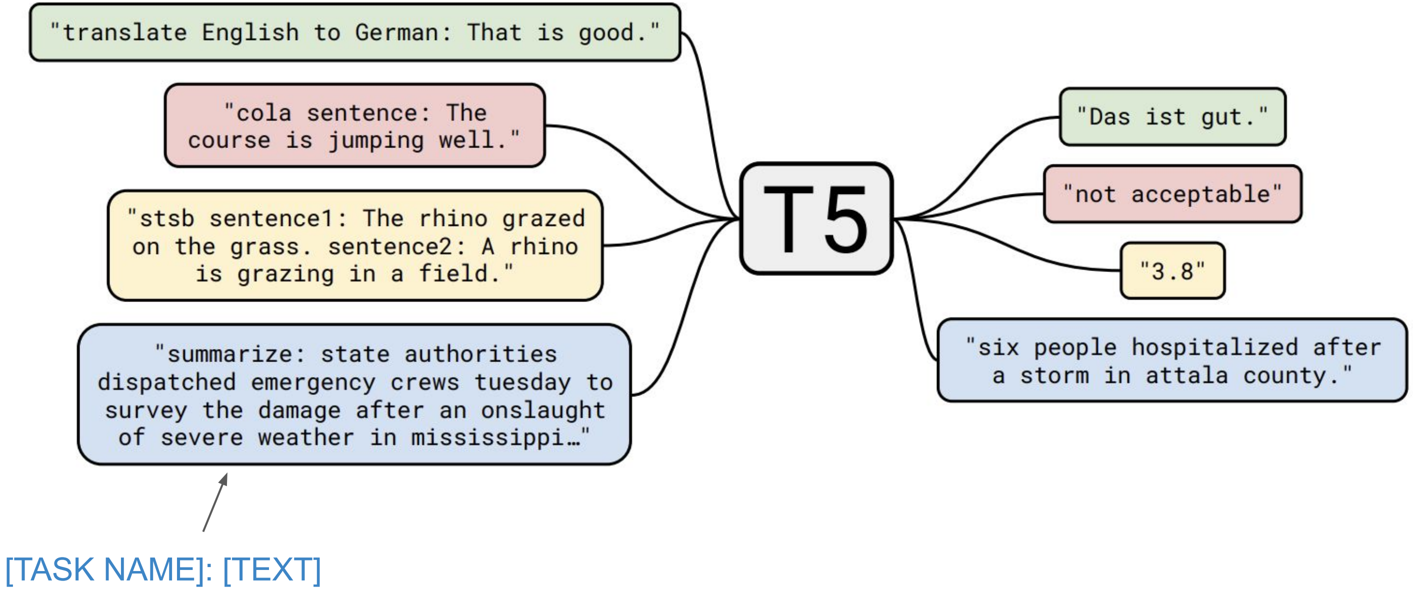


This is implemented in text preprocessing: it's still an objective that looks like **language modeling** at the decoder side.



[Raffel et al., 2018]

T5: a unified sequence-to-sequence model



[Raffel et al., 2018]

Summary of today

- Biological sequences
  - DNA, RNA, protein
- CNNs
  - Extract spatial features using convolution filters
- RNNs
  - Capture temporal dependency
  - LSTM, GRU
- Transformers
  - Self-attention layer
  - Widely used in recent state-of-the-art NLP models
- Overview of language models (LMs)
  - LM: predicting a word given context
- Next:
  - Delve into LLMs
  - How to build ChatGPT?
  - LLMs for biological sequences