Lecture 8:
More on Text Data and Representations

Announcements

- A1 due yesterday
- Project proposal due Friday, 10/9
- A2 will be released tonight

Last time: Token embeddings

$$[0\ 0\ 1\ 0\ 0\ 0\ 0\ \ldots.\ 0] \quad X \quad \begin{bmatrix} 0.5 & 0.2 & 0.1 \\ 0.6 & 0.1 & 0.6 \\ 0.5 & 0.8 & 0.2 \\ 0.7 & 0.9 & 0.3 \\ 0.3 & 0.5 & 0.1 \\ & \ldots & \\ 0.7 & 0.8 & 0.1 \end{bmatrix} \quad = \quad [0.5\quad 0.8\quad 0.2]$$

1xN token input (one-hot selection of token)

N x D embedding matrix

D-dim token embedding

Last time: Token embeddings

$$[0\ 0\ 1\ 0\ 0\ 0\ 0\ \dots.\ 0] \quad X \quad \begin{bmatrix} 0.5 & 0.2 & 0.1 \\ 0.6 & 0.1 & 0.6 \\ 0.5 & 0.8 & 0.2 \\ 0.7 & 0.9 & 0.3 \\ 0.3 & 0.5 & 0.1 \\ & \dots & \\ 0.7 & 0.8 & 0.1 \end{bmatrix} \quad = \quad [0.5\quad 0.8\quad 0.2]$$

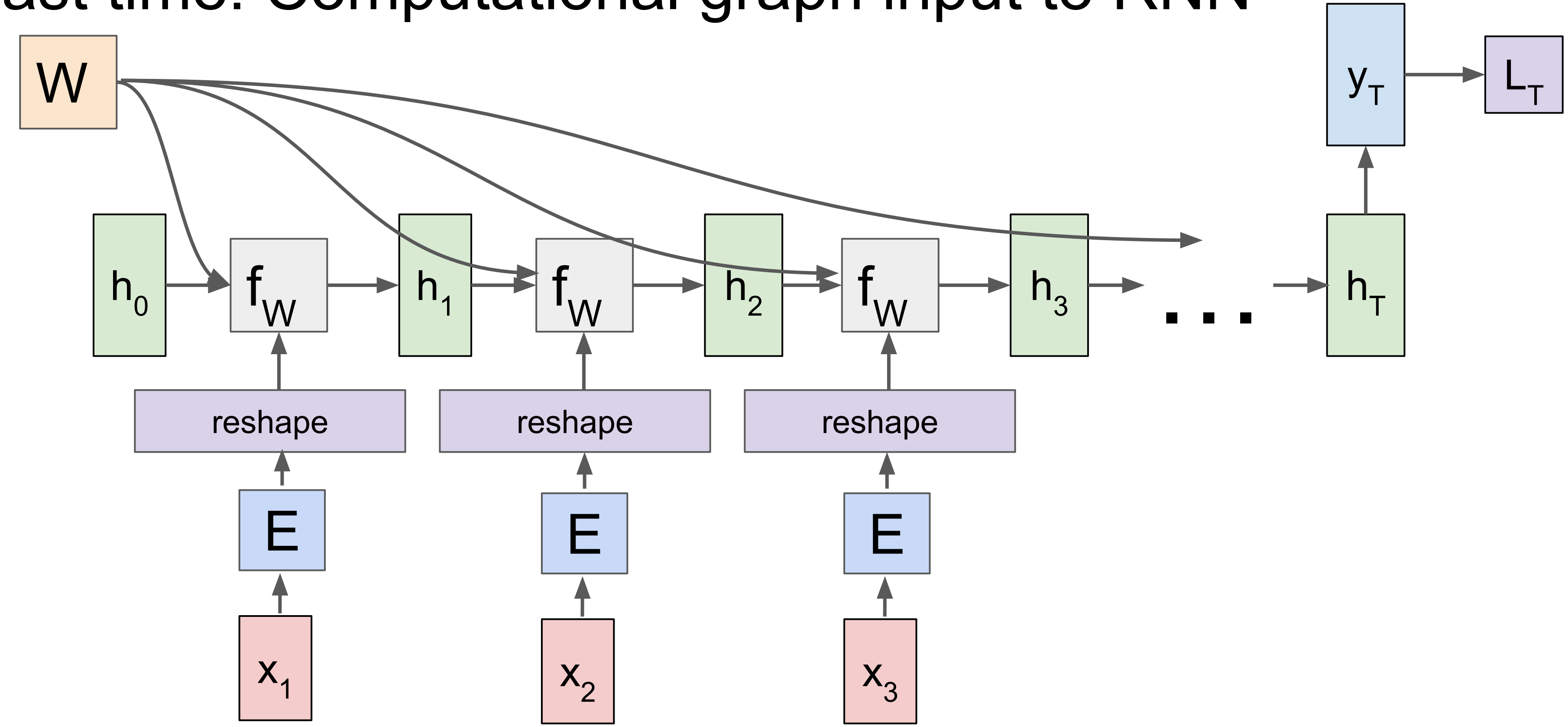
1xN token input (one-hot selection of token)

N x D embedding matrix

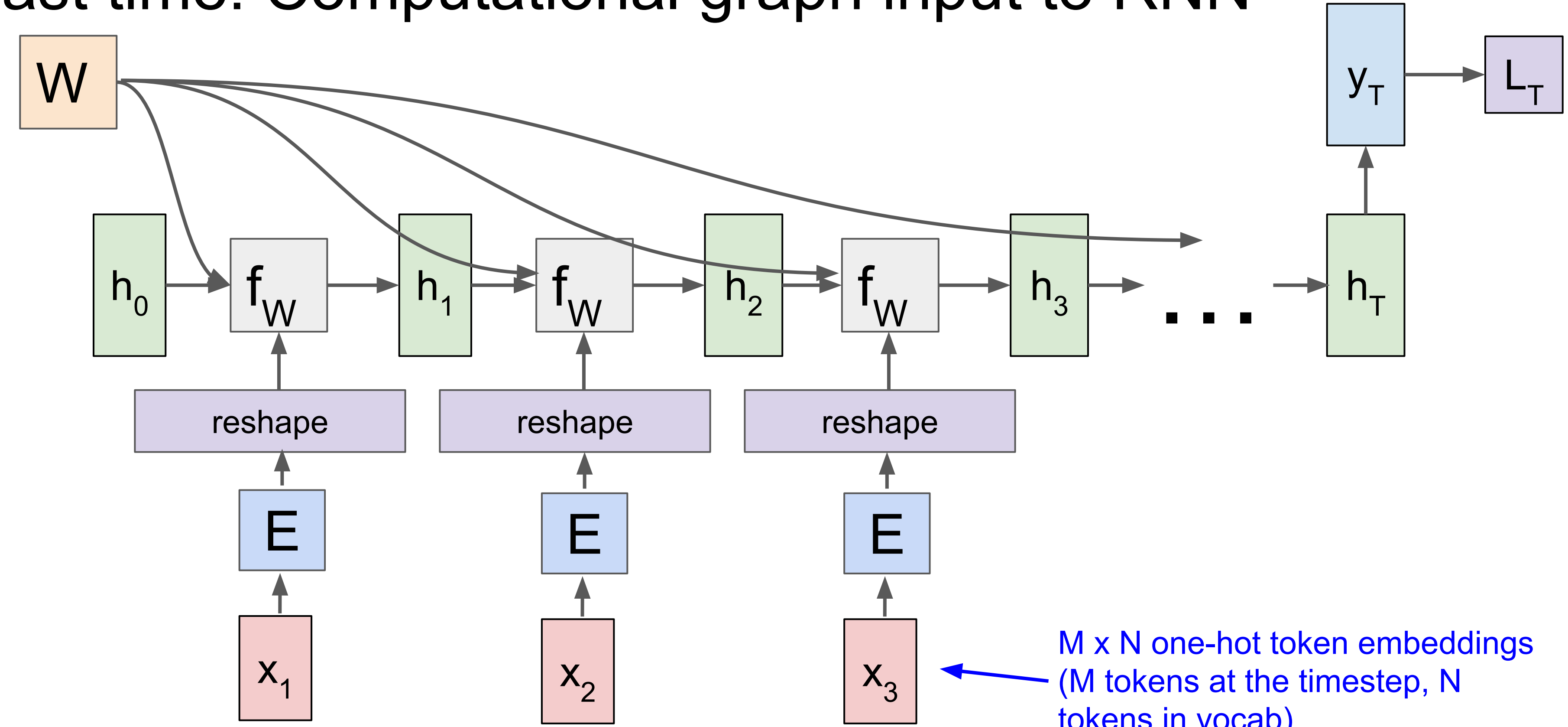
D-dim token embedding

In general, learning embedding matrices are a useful way to map discrete data into a semantically meaningful, continuous space! Will see frequently in **natural language processing**.

Last time: Computational graph input to RNN

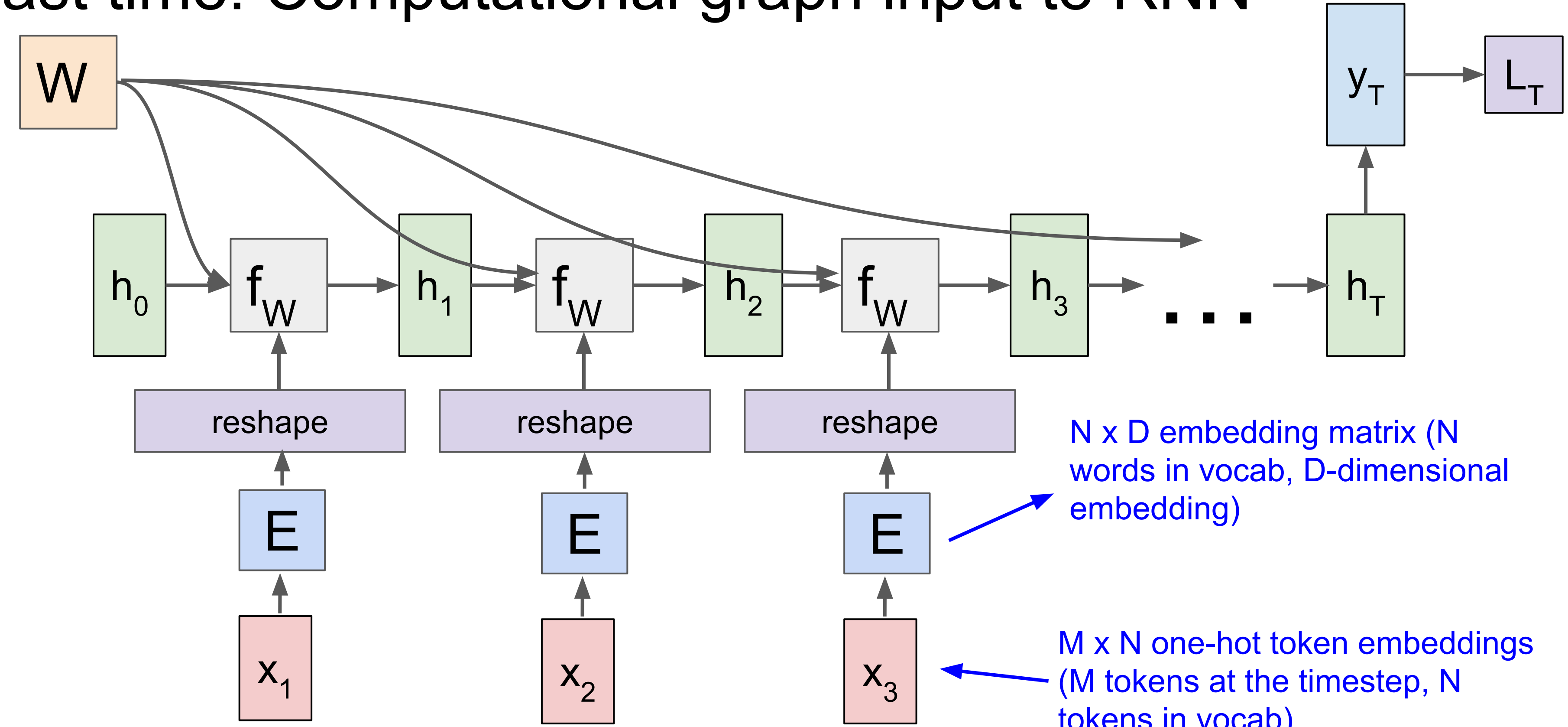


Last time: Computational graph input to RNN



M x N one-hot token embeddings (M tokens at the timestep, N tokens in vocab)

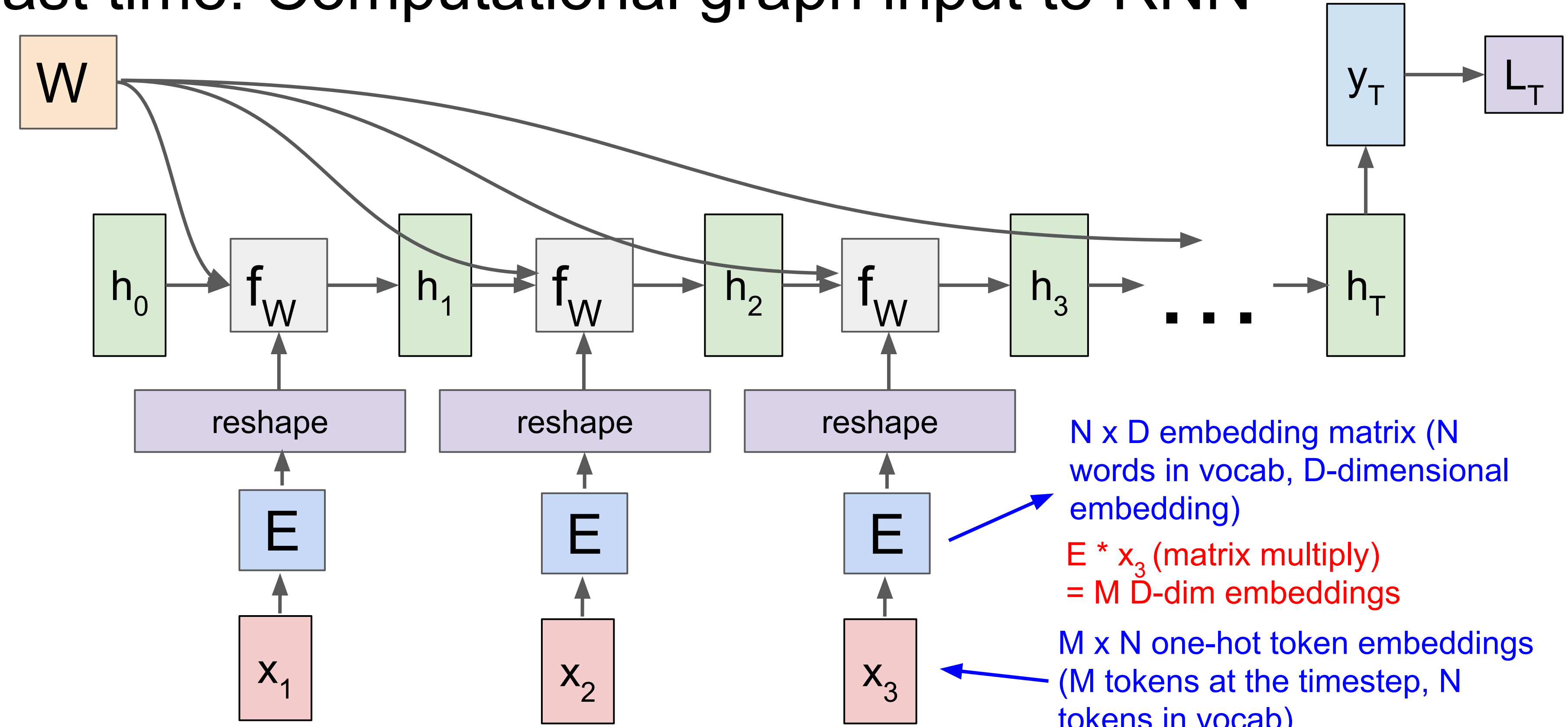
Last time: Computational graph input to RNN



N x D embedding matrix (N words in vocab, D-dimensional embedding)

M x N one-hot token embeddings (M tokens at the timestep, N tokens in vocab)

Last time: Computational graph input to RNN

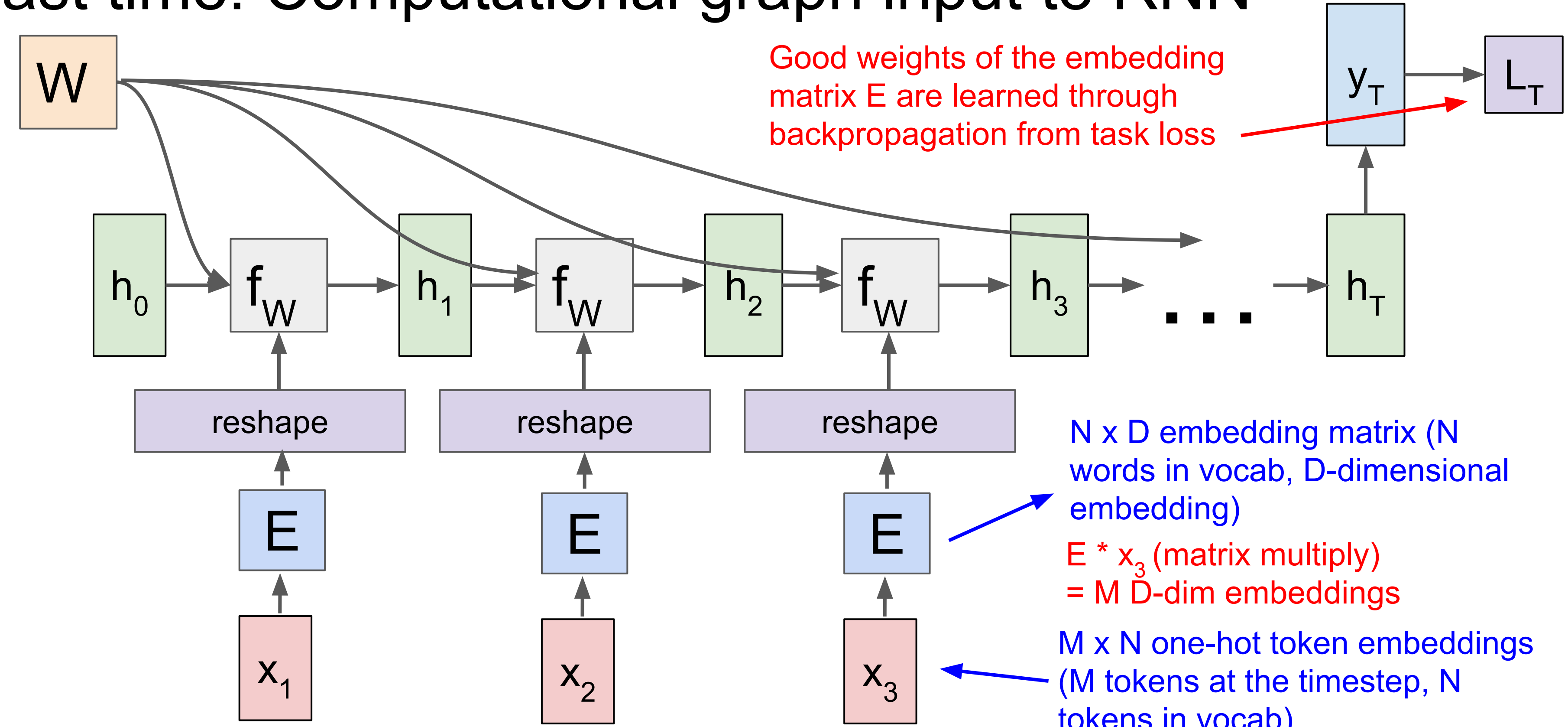


N x D embedding matrix (N words in vocab, D-dimensional embedding)

E * x_3 (matrix multiply) = M D-dim embeddings

M x N one-hot token embeddings (M tokens at the timestep, N tokens in vocab)

Last time: Computational graph input to RNN



Good weights of the embedding matrix E are learned through backpropagation from task loss

N x D embedding matrix (N words in vocab, D-dimensional embedding)

E * x_3 (matrix multiply) = M D-dim embeddings

M x N one-hot token embeddings (M tokens at the timestep, N tokens in vocab)

Today: ~~Token~~ Word Embeddings

[0 0 1 0 0 0 0 …. 0] X

1xN token input (one-hot selection of token)

0.5	0.2	0.1
0.6	0.1	0.6
0.5	0.8	0.2
0.7	0.9	0.3
0.3	0.5	0.1
…		
0.7	0.8	0.1

N x D embedding matrix

= [0.5 0.8 0.2]

D-dim token embedding

Today: ~~Token~~ Word Embeddings

$$[0\ 0\ 1\ 0\ 0\ 0\ 0\ \ldots.\ 0] \quad X \quad \begin{bmatrix} 0.5 & 0.2 & 0.1 \\ 0.6 & 0.1 & 0.6 \\ 0.5 & 0.8 & 0.2 \\ 0.7 & 0.9 & 0.3 \\ 0.3 & 0.5 & 0.1 \\ & \ldots & \\ 0.7 & 0.8 & 0.1 \end{bmatrix} \quad = \quad [0.5\ \ 0.8\ \ 0.2]$$

1xN token input (one-hot selection of token)

N x D embedding matrix

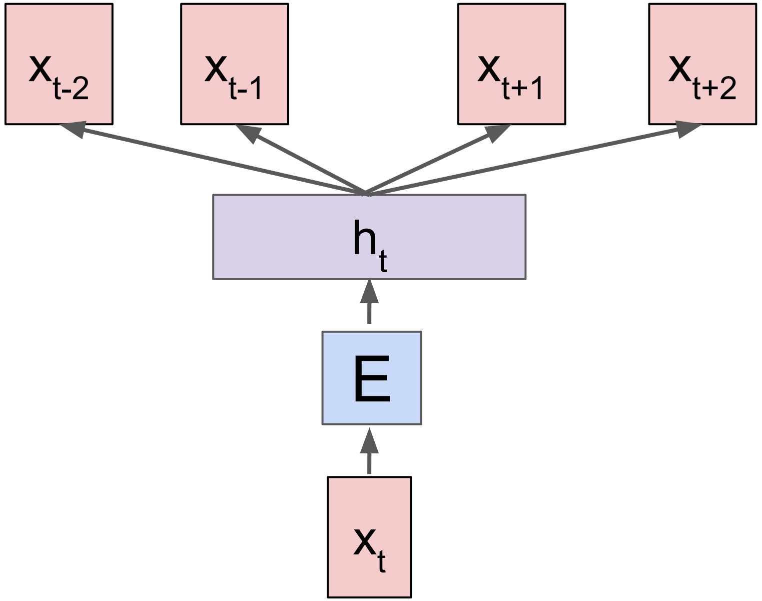
D-dim token embedding

Words come from a discrete vocabulary! Can learn word embeddings using a similar framework

Learning word embeddings

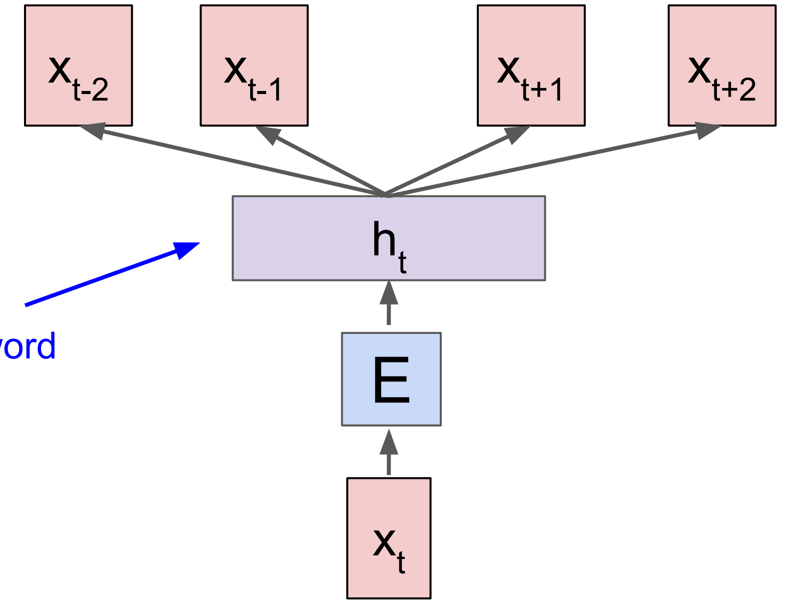
- Allows converting text data into numerical representations that can be used in prediction models
- Key new idea when learning word embeddings: Do not need to learn embedding matrix only from prediction task loss. Instead, can design new loss functions based only on the structure of free text.
    - When goal is to learn an entire (large) dictionary of word embeddings, available labeled training examples may not be sufficient to effectively learn and model relationships between words
    - Loss functions based only on structure of free text allows taking advantage of much more available text data that do not have prediction labels associated!

Skip-gram model



Mikolov, et al. Efficient Estimation of Word Representations in Vector Space, 2013.

Skip-gram model

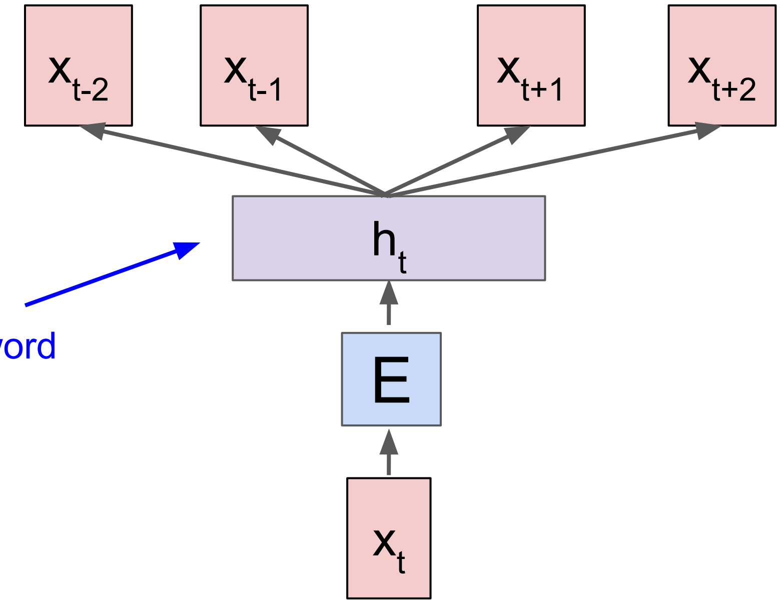


Word embedding (feature vector), of word at the t-th position

Mikolov, et al. Efficient Estimation of Word Representations in Vector Space, 2013.

Skip-gram model

Use word embedding vector to predict the word identity of a set of neighboring positions



Word embedding (feature vector), of word at the t-th position

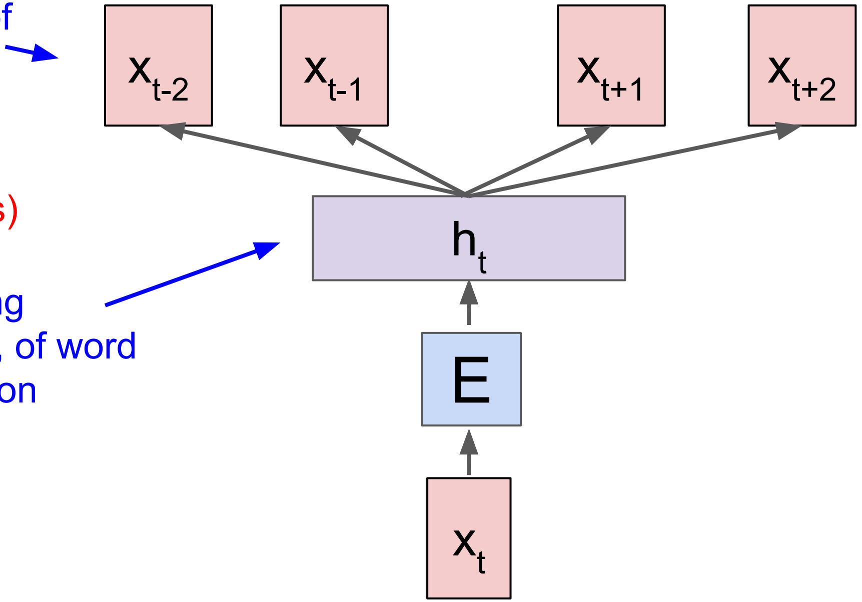
Mikolov, et al. Efficient Estimation of Word Representations in Vector Space, 2013.

Skip-gram model

Use word embedding vector to predict the word identity of a set of neighboring positions

(Each is an N-way classification if the dictionary has N words)

Word embedding (feature vector), of word at the t-th position



Mikolov, et al. Efficient Estimation of Word Representations in Vector Space, 2013.

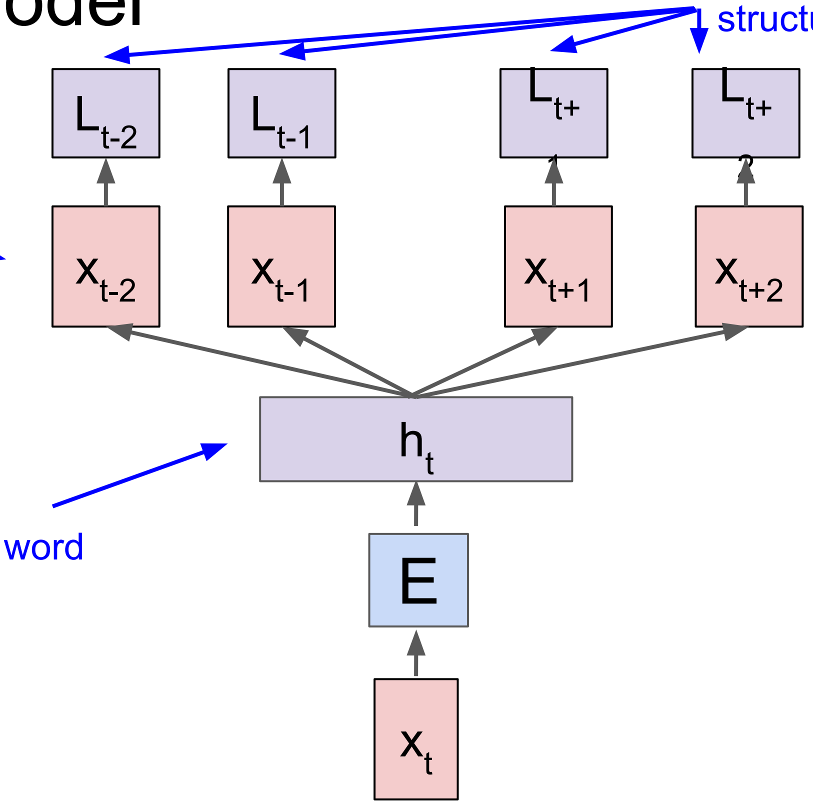
Skip-gram model

Can train using a classification loss (e.g. softmax loss) based only on the text structure, without any external labels!

Use word embedding vector to predict the word identity of a set of neighboring positions

(Each is an N-way classification if the dictionary has N words)

Word embedding (feature vector), of word at the t-th position



Mikolov, et al. Efficient Estimation of Word Representations in Vector Space, 2013.

Skip-gram model

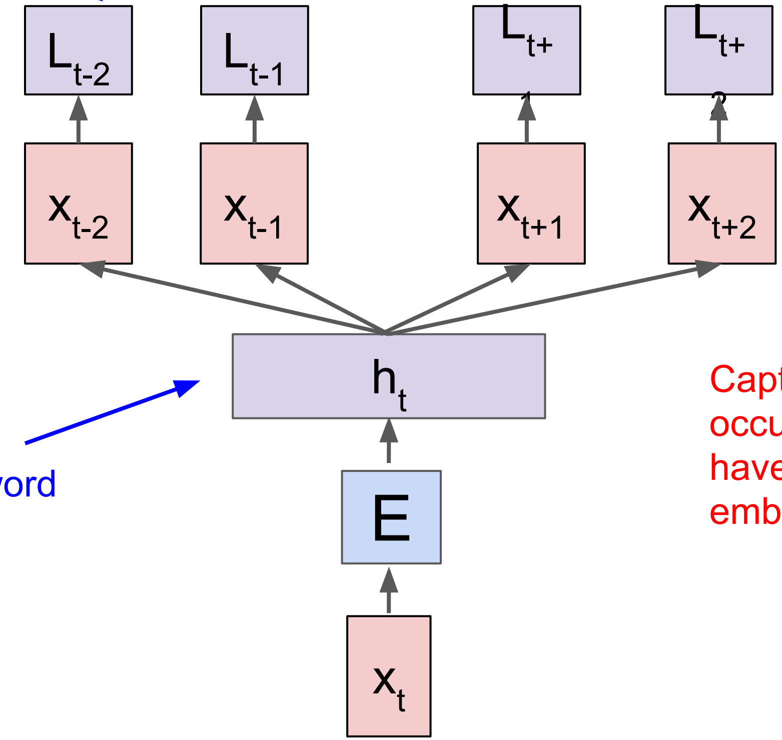
Can train using a classification loss (e.g. softmax loss) based only on the text structure, without any external labels!

Use word embedding vector to predict the word identity of a set of neighboring positions

(Each is an N-way classification if the dictionary has N words)

Word embedding (feature vector), of word at the t-th position

Captures notion that words occurring in similar contexts should have similar feature vectors (word embeddings)



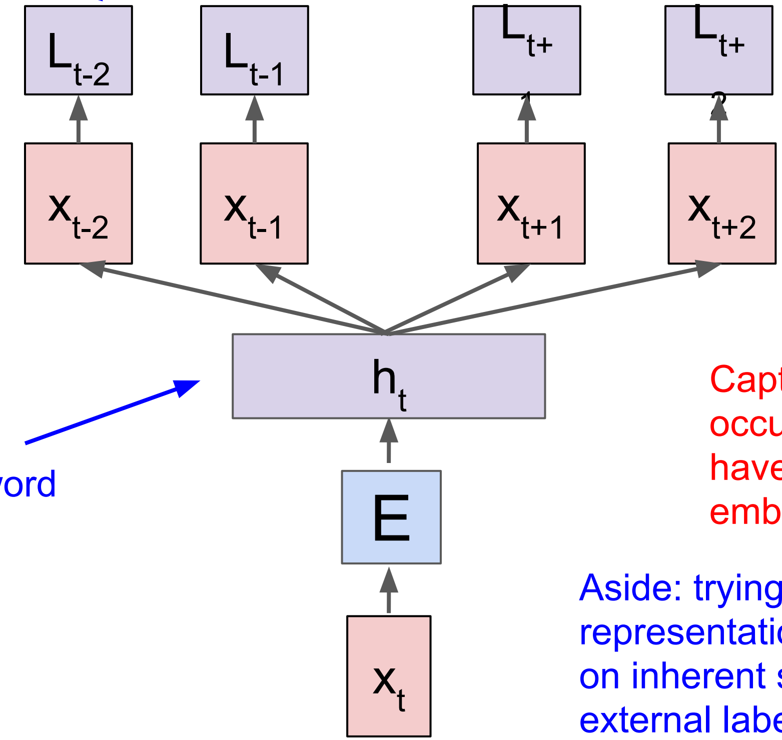
Mikolov, et al. Efficient Estimation of Word Representations in Vector Space, 2013.

Skip-gram model

Can train using a classification loss (e.g. softmax loss) based only on the text structure, without any external labels!

Use word embedding vector to predict the word identity of a set of neighboring positions

(Each is an N-way classification if the dictionary has N words)



Word embedding (feature vector), of word at the t-th position

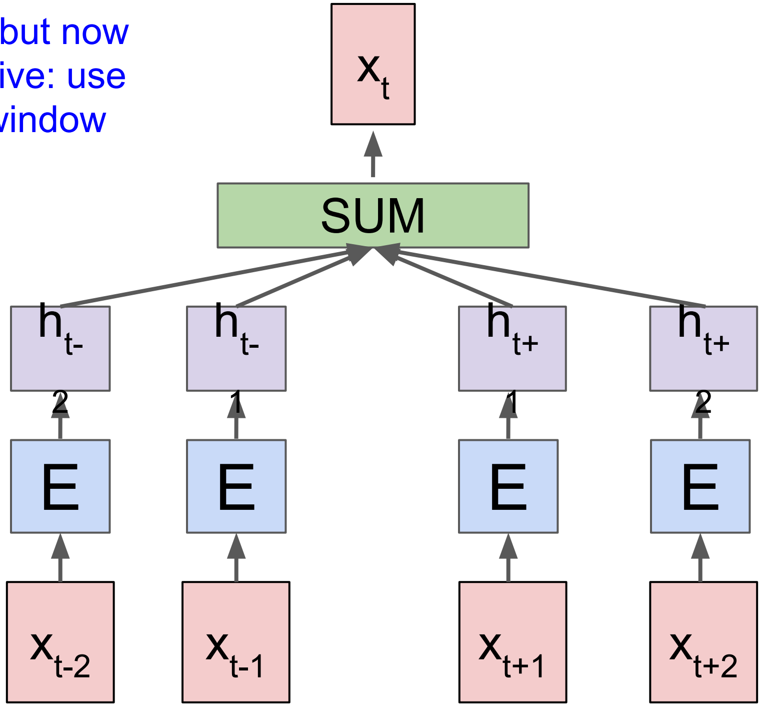
Captures notion that words occurring in similar contexts should have similar feature vectors (word embeddings)

Aside: trying to learn "good" feature representations using loss functions based on inherent structure in data, as opposed to external labels, is a currently active area of research called "**self-supervised learning**"

Mikolov, et al. Efficient Estimation of Word Representations in Vector Space, 2013.

Continuous Bag-of-Words (CBOW) model

Similar idea as Skip-gram, but now slightly different loss objective: use embeddings from context window to predict center word



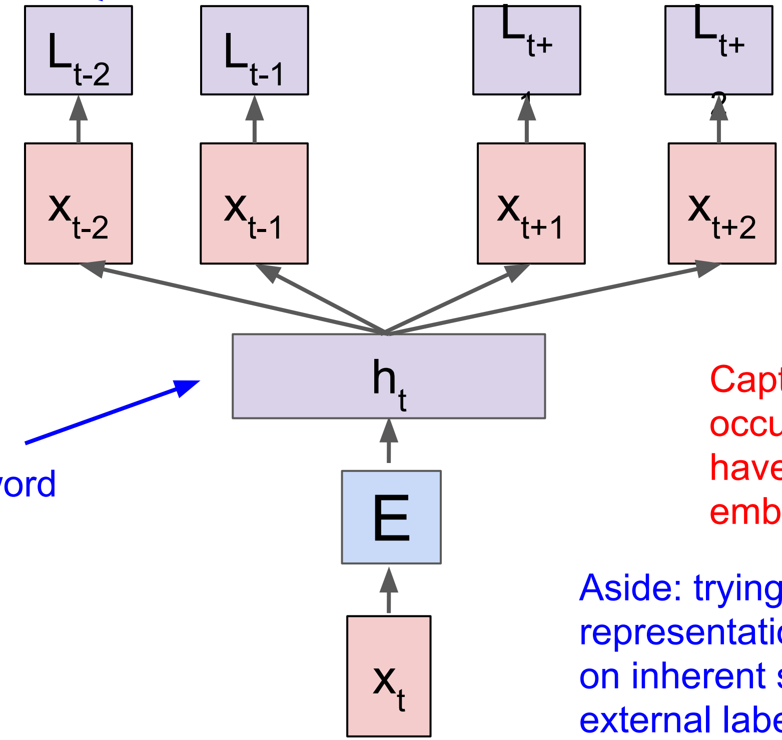
Mikolov, et al. Efficient Estimation of Word Representations in Vector Space, 2013.

Back to skip-gram model...

Can train using a classification loss (e.g. softmax loss) based only on the text structure, without any external labels!

Use word embedding vector to predict the word identity of a set of neighboring positions

(Each is an N-way classification if the dictionary has N words)



Word embedding (feature vector), of word at the t-th position

Captures notion that words occurring in similar contexts should have similar feature vectors (word embeddings)

Aside: trying to learn "good" feature representations using loss functions based on inherent structure in data, as opposed to external labels, is a currently active area of research called "**self-supervised learning**"

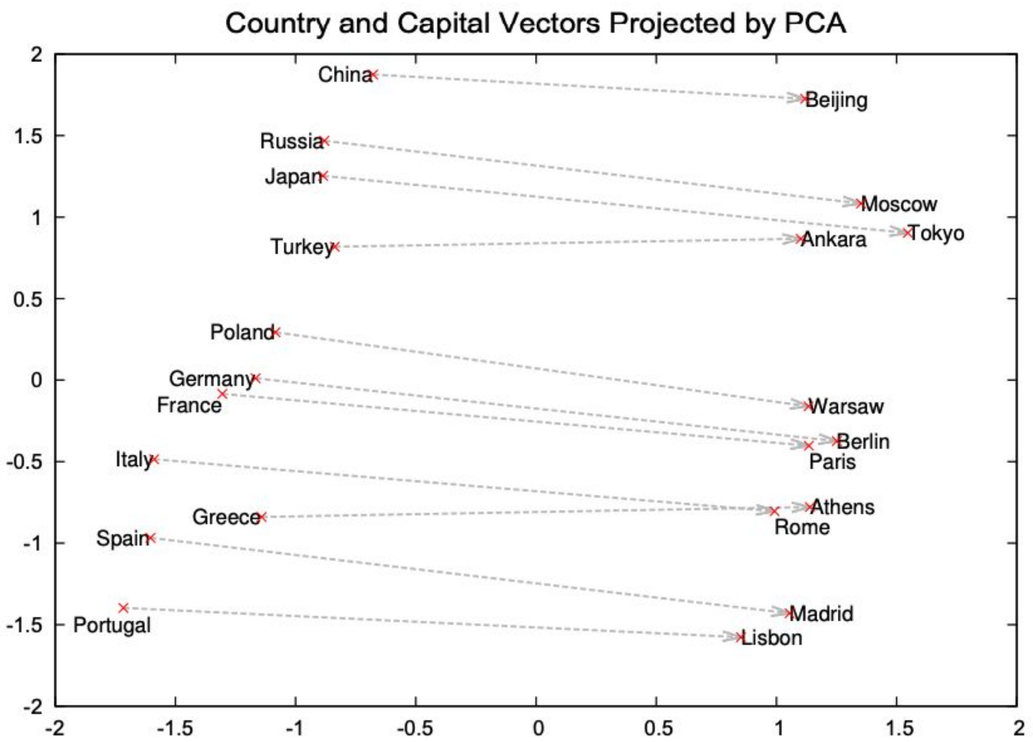
Mikolov, et al. Efficient Estimation of Word Representations in Vector Space, 2013.

Word2Vec

- Skip-gram model with a few improvements:
  - **Negative sampling:** Convert expensive N-way softmax classification (performed in hierarchical fashion) into efficient binary classification tasks.
    - (x_t, x_i) -> label 1, if x_i is in context window of x_t (combine x_t and x_i with dot product to get input vector to classification model)
    - (x_t, x_i) -> label 0 if x_i is outside context window of x_t (sample just a few of these negatives)
  - **Subsampling of frequent words:** in training, discard words with a probability based on their frequency in the data (e.g., “the” is more likely to be discarded)

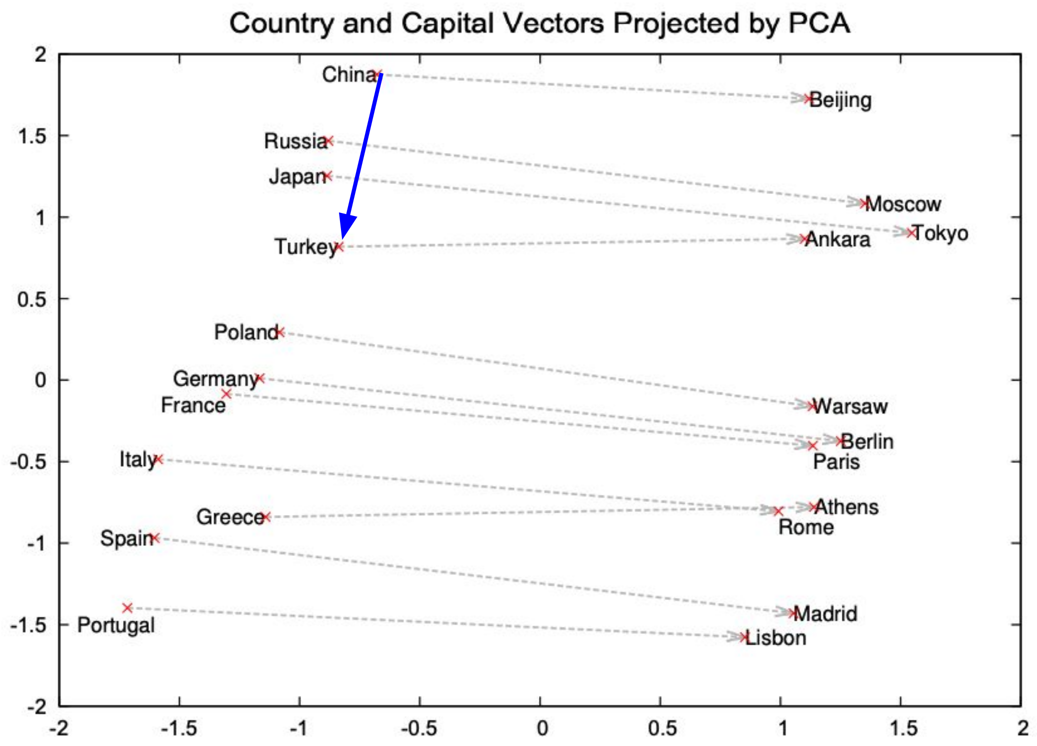
Mikolov, et al. Distributed Representations of Words and Phrases and their Compositionality, 2013.

Word arithmetic with Word2Vec



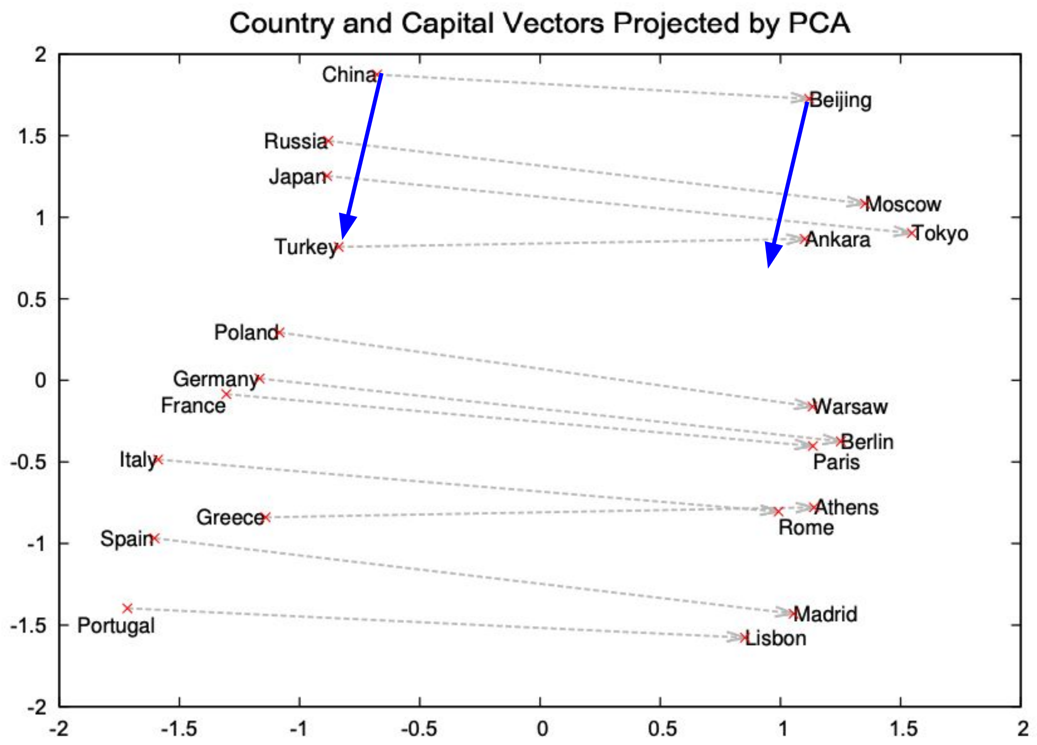
Mikolov, et al. Distributed Representations of Words and Phrases and their Compositionality, 2013.

Word arithmetic with Word2Vec



Mikolov, et al. Distributed Representations of Words and Phrases and their Compositionality, 2013.

Word arithmetic with Word2Vec



Mikolov, et al. Distributed Representations of Words and Phrases and their Compositionality, 2013.

Obtaining word2vec embeddings

- Publicly available models trained on very large word corpuses (e.g. 100 billion words from Google News, with 3 million words and 300-dim embeddings) -> libraries like gensim in python allow easy access!
- Can use directly as a feature extractor for text data, or fine-tune / train on your corpus

Mikolov, et al. Distributed Representations of Words and Phrases and their Compositionality, 2013.

Obtaining word2vec embeddings

- Publicly available models trained on very large word corpuses (e.g. 100 billion words from Google News, with 3 million words and 300-dim embeddings) -> libraries like gensim in python allow easy access!
- Can use directly as a feature extractor for text data, or fine-tune / train on your corpus

- Can also try GloVe embeddings: another approach for learning embeddings based on global co-occurrence matrices in the text. Also publicly available pre-trained models! Performance can be better but generally in similar range, depends on dataset.

Mikolov, et al. Distributed Representations of Words and Phrases and their Compositionality, 2013.
Pennington, et al. GloVe: Global Vectors for Word Representation, 2014.

From word2vec to embeddings for sentences / documents

- Simplest approach: average embeddings of individual words in the text
    - Usually helpful to perform weighted average, where each word is weighted by relative frequency (e.g., by tf-idf score)
- More complex approaches:
    - Extensions of word2vec, e.g. doc2vec
    - Extensions of word embedding idea based on RNNs for sequence embedding

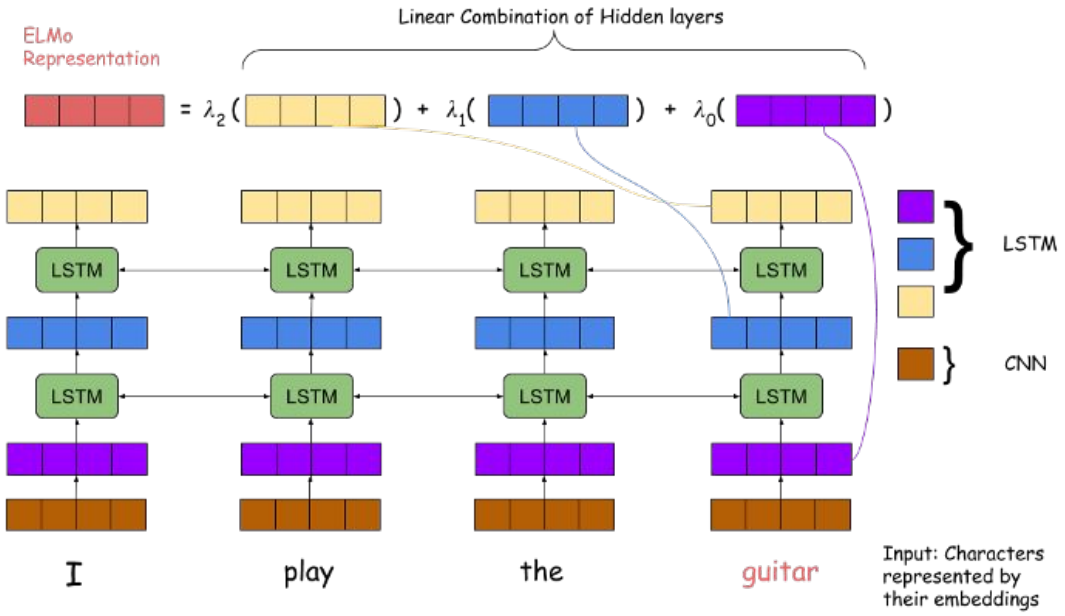
From word2vec to embeddings for sentences / documents

- Simplest approach: average embeddings of individual words in the text
  - Usually helpful to perform weighted average, where each word is weighted by relative frequency (e.g., by tf-idf score)
- More complex approaches:
  - Extensions of word2vec, e.g. doc2vec
  - Extensions of word embedding idea based on RNNs for sequence embedding

Will also see more powerful methods to come, e.g. BERT. Let's now briefly discuss a few more advancements on the way to BERT...

Context-based word embeddings: ELMo

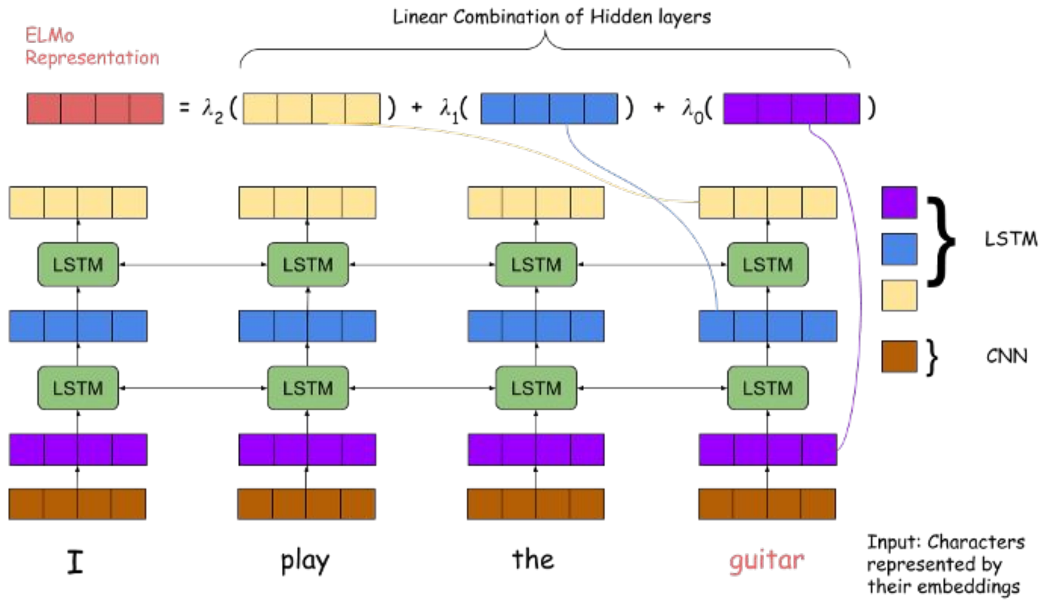
- Key idea: embedding of a word should be able to be different based on context -- the same word can have multiple meanings! (e.g., dog “bark” vs. tree “bark)
- Produce embedding for a word based on its context (captured by bidirectional LSTMs)


ELMo Representation
Linear Combination of Hidden layers
$= \lambda_2(\ ) + \lambda_1(\ ) + \lambda_0(\ )$
LSTM
LSTM
CNN
I
play
the
guitar
Input: Characters represented by their embeddings

Peters, et al. Deep contextualized word representations, 2018.

Context-based word embeddings: ELMo

- Key idea: embedding of a word should be able to be different based on context -- the same word can have multiple meanings! (e.g., dog "bark" vs. tree "bark)
- Produce embedding for a word based on its context (captured by bidirectional LSTMs)
- Training scheme: loss is next-word prediction (common setup for learning language models); also operates with character-level embeddings that are then combined


Linear Combination of Hidden layers
ELMo Representation
$= \lambda_2(\;) + \lambda_1(\;) + \lambda_0(\;)$
LSTM
LSTM
LSTM
CNN
I
play
the
guitar
Input: Characters represented by their embeddings

Peters, et al. Deep contextualized word representations, 2018.

Tackling transfer learning in NLP: ULMFiT

- Universal Language Model Fine-Tuning (ULMFiT)
- Previously, transfer learning in NLP had been less successful than in Computer Vision: generally required a large amount of in-domain data to work
- ULMFiT demonstrated that training a general language model for next-word prediction (using a bidirectional LSTM) could be successfully fine-tuned to achieve state-of-the-art on a variety of NLP tasks: sentiment analysis, question classification, topic classification

Howard and Ruder. Universal Language Model Fine-tuning for Text Classification, 2018.

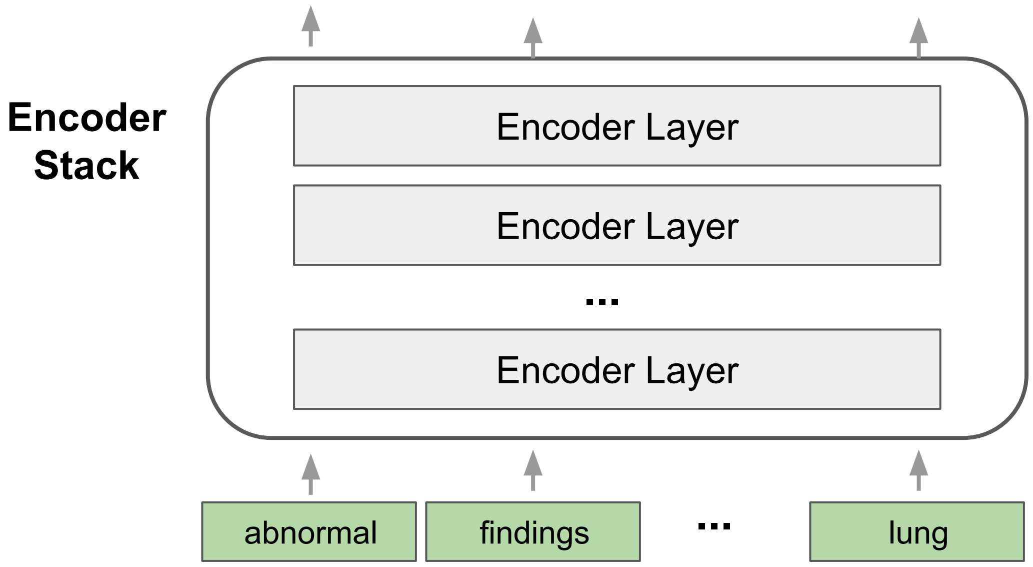
BERT: Highly successful transfer learning through learning bidirectional representations with a “Transformer” architecture

- BERT: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers
- Builds on ELMo idea of bidirectional context embeddings, but introduces advancements with “Transformer” architecture and new training objectives
- Showed that learned model could be a successful “pre-trained” model that could be fine-tuned to achieve state-of-the-art performance on 11 different NLP tasks: an “ImageNet” moment for NLP

Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.

Transformer architecture framework

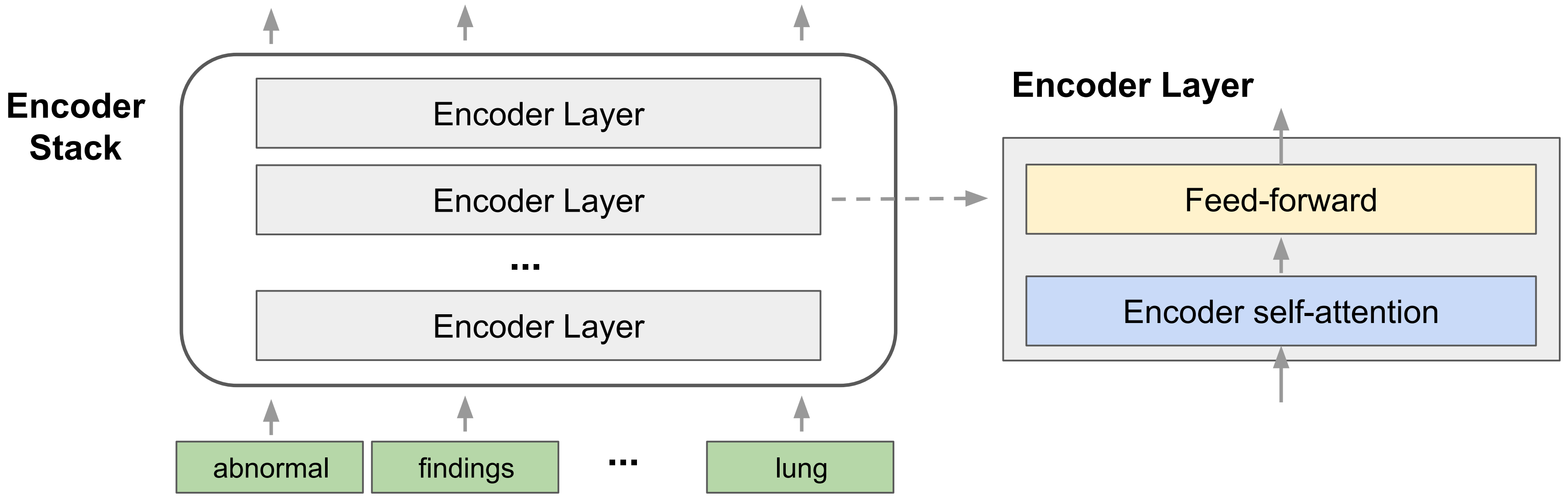
- Recent approach for sequence processing based on “self-attention” proposed by Vaswani et al. 2017. BERT uses a stack of “encoder layers” each with self-attention.



Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.
Vaswani et al. Attention is All You Need, 2017.

Transformer architecture framework

- Recent approach for sequence processing based on “self-attention” proposed by Vaswani et al. 2017. BERT uses a stack of “encoder layers” each with self-attention.



Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.
Vaswani et al. Attention is All You Need, 2017.

Transformer “attention” mechanism

Consider first attention between a sequence x (of length num_x), and a sequence y (of length num_y):

$$a_j = \mathrm{softmax}\left(\frac{Q_j(x)K_j(y)^T}{\sqrt{d_c}}\right)V_j(y)$$

Vaswani et al. Attention is All You Need, 2017.
Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.

Transformer "attention" mechanism

Consider first attention between a sequence x (of length num_x), and a sequence y (of length num_y):

"Query" embedding: [num_x, d_c] where
d_c is embedding dimension

$$a_j = \text{softmax}\left(\frac{Q_j(x)K_j(y)^T}{\sqrt{d_c}}\right)V_j(y)$$

Vaswani et al. Attention is All You Need, 2017.
Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.

Transformer “attention” mechanism

Consider first attention between a sequence x (of length num_x), and a sequence y (of length num_y):

“Query” embedding: [num_x, d_c] where d_c is embedding dimension

“Key” embedding: [num_y, d_c]

$$a_j = \text{softmax}\left(\frac{Q_j(x)K_j(y)^T}{\sqrt{d_c}}\right)V_j(y)$$

Vaswani et al. Attention is All You Need, 2017.
Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.

Transformer “attention” mechanism

Consider first attention between a sequence x (of length num_x), and a sequence y (of length num_y):

“Query” embedding: [num_x, d_c] where d_c is embedding dimension

“Key” embedding: [num_y, d_c]

$$a_j = \text{softmax}\left(\frac{Q_j(x)K_j(y)^T}{\sqrt{d_c}}\right)V_j(y)$$

Embedding dimension

Vaswani et al. Attention is All You Need, 2017.
Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.

Transformer “attention” mechanism

Consider first attention between a sequence x (of length num_x), and a sequence y (of length num_y):

“Query” embedding: [num_x, d_c] where d_c is embedding dimension

“Key” embedding: [num_y, d_c]

$$a_j = \underbrace{\text{softmax}\left(\frac{Q_j(x)K_j(y)^T}{\sqrt{d_c}}\right)}_{\text{Attention weights}} V_j(y)$$

Embedding dimension

Vaswani et al. Attention is All You Need, 2017.
Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.

Transformer "attention" mechanism

Consider first attention between a sequence x (of length num_x), and a sequence y (of length num_y):

"Query" embedding: [num_x, d_c] where d_c is embedding dimension

"Key" embedding: [num_y, d_c]

$$a_j = \underbrace{\mathrm{softmax}\left(\frac{Q_j(x)K_j(y)^T}{\sqrt{d_c}}\right)}_{\text{Attention weights}} V_j(y)$$

Embedding dimension

"Value" embedding: [num_y, d_c]

Vaswani et al. Attention is All You Need, 2017.
Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.

Transformer “attention” mechanism

Consider first attention between a sequence x (of length num_x), and a sequence y (of length num_y):

“Query” embedding: [num_x, d_c] where d_c is embedding dimension

“Key” embedding: [num_y, d_c]

$$a_j = \underbrace{\text{softmax}\left(\frac{Q_j(x)K_j(y)^T}{\sqrt{d_c}}\right)}_{\text{Attention weights}} V_j(y)$$

Attention-weighted outputs of jth attention head: [num_x, d_c]

Embedding dimension

“Value” embedding: [num_y, d_c]

Attention weights

Vaswani et al. Attention is All You Need, 2017.
Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.

Transformer “attention” mechanism

Consider first attention between a sequence x (of length num_x), and a sequence y (of length num_y):

“Query” embedding: [num_x, d_c] where d_c is embedding dimension

“Key” embedding: [num_y, d_c]

$$\underbrace{a_j}_{\text{Attention-weighted outputs of jth attention head: [num\_x, d\_c]}} = \underbrace{\text{softmax}\left(\frac{Q_j(x)K_j(y)^T}{\sqrt{d_c}}\right)}_{\text{Attention weights}} V_j(y)$$

Embedding dimension

“Value” embedding: [num_y, d_c]

Attention-weighted outputs of jth attention head: [num_x, d_c]

If using multiple attention heads, combine outputs with another matrix multiply

Attention weights

Vaswani et al. Attention is All You Need, 2017.
Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.

Transformer "self-attention" mechanism

"Self-attention" is just this attention mechanism with x = y!

Consider first attention between a sequence x (of length num_x), and a sequence y (of length num_y):

"Query" embedding: [num_x, d_c] where d_c is embedding dimension

"Key" embedding: [num_y, d_c]

$$a_j = \underbrace{\text{softmax}\left(\frac{Q_j(x)K_j(y)^T}{\sqrt{d_c}}\right)}_{\text{Attention weights}} V_j(y)$$

Attention-weighted outputs of jth attention head: [num_x, d_c]

Embedding dimension

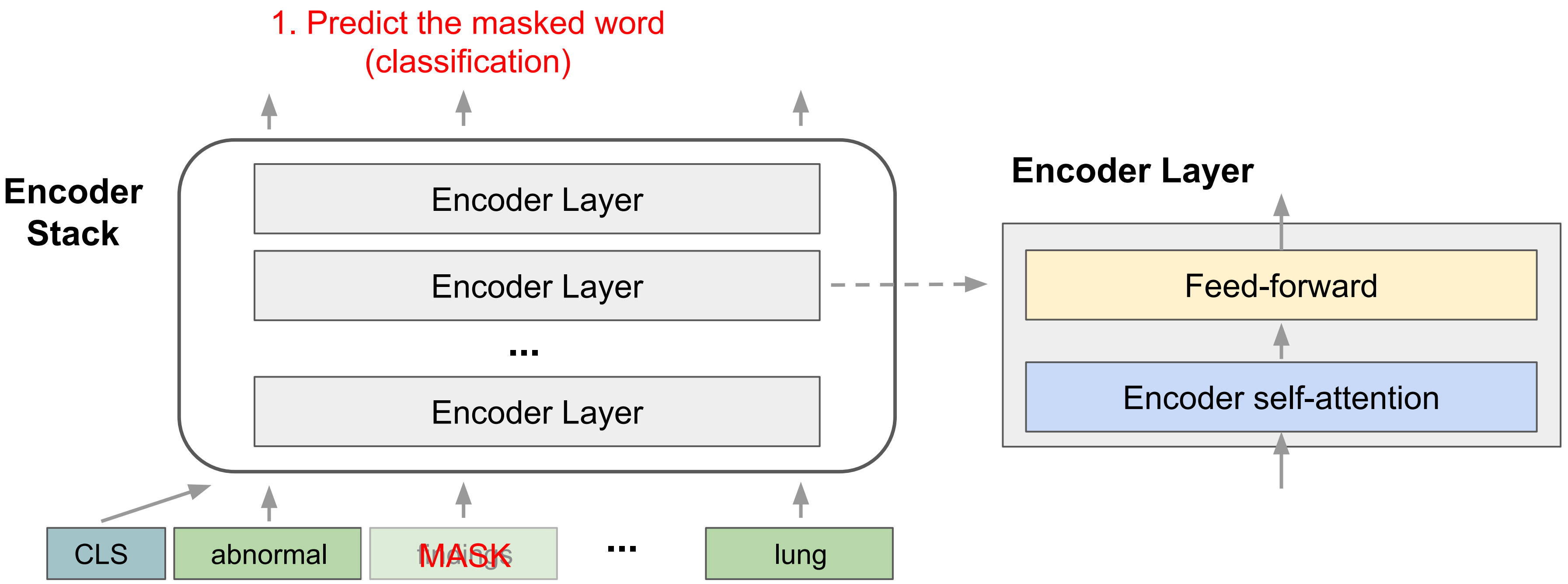
"Value" embedding: [num_y, d_c]

If using multiple attention heads, combine outputs with another matrix multiply

Attention weights

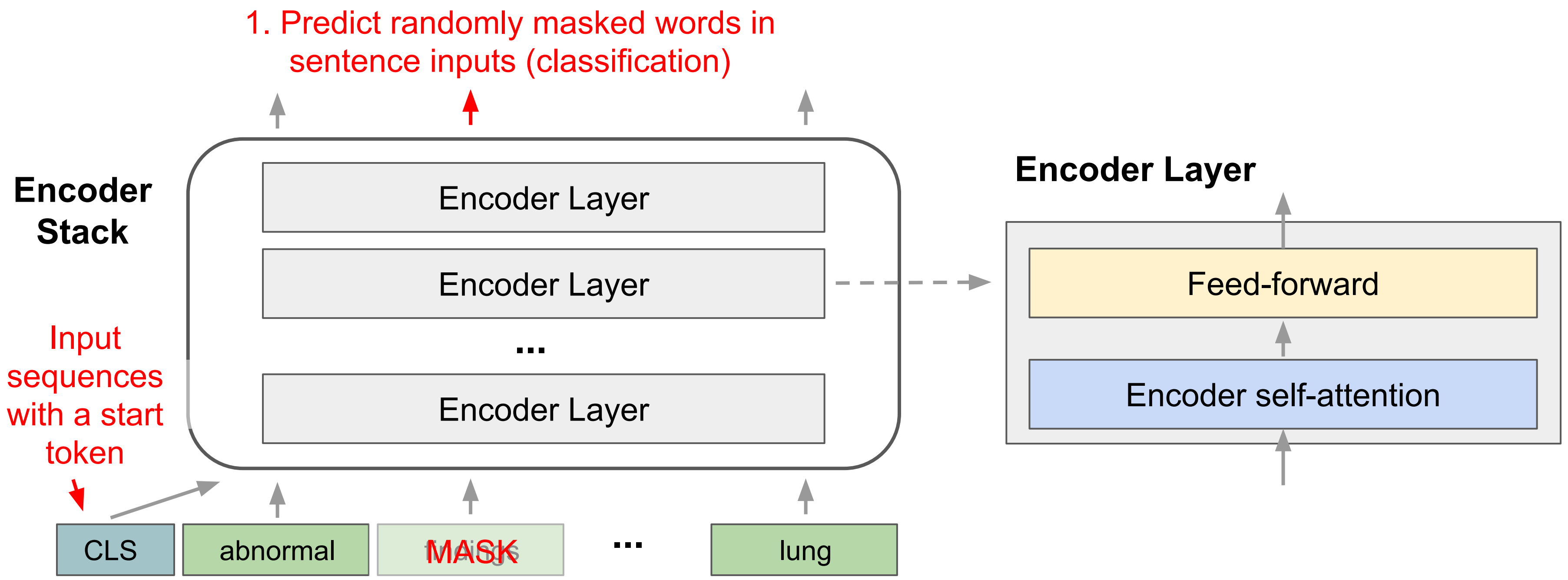
Vaswani et al. Attention is All You Need, 2017.
Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.

Training BERT



Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.
Vaswani et al. Attention is All You Need, 2017.

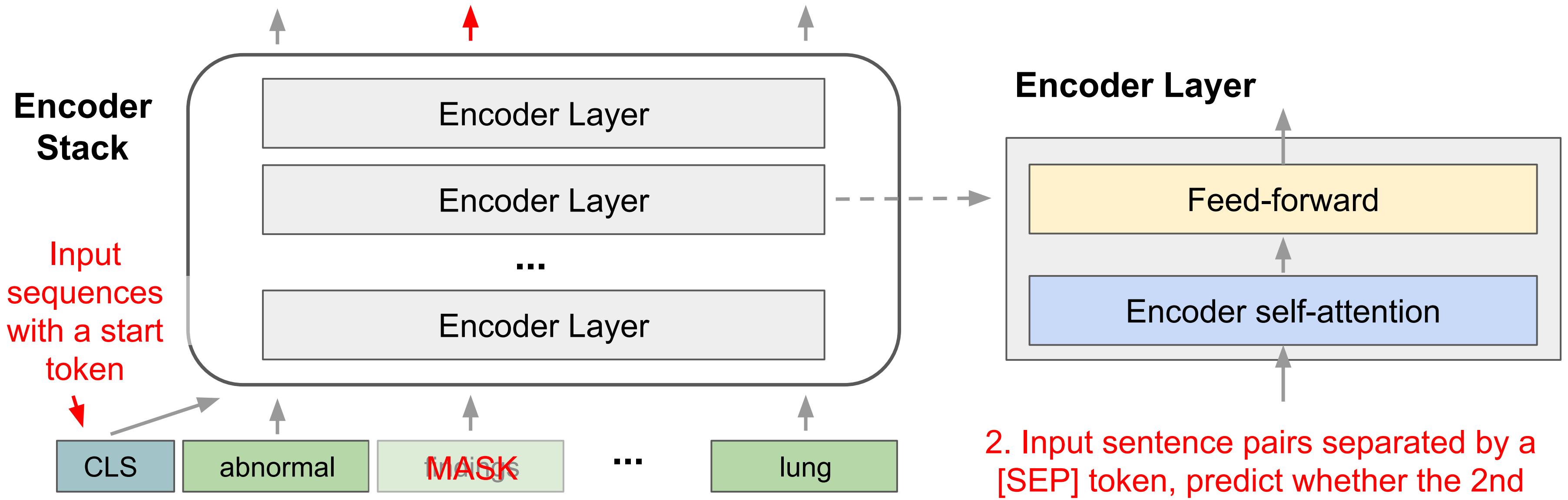
Training BERT



Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.
Vaswani et al. Attention is All You Need, 2017.

Training BERT

1. Predict randomly masked words in sentence inputs (classification)



2. Input sentence pairs separated by a [SEP] token, predict whether the 2nd sentence follows the 1st in the text

Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.
Vaswani et al. Attention is All You Need, 2017.

BERT: Highly successful transfer learning through learning bidirectional representations with a “Transformer” architecture

- Covered main idea of BERT, but there are more details that you can find in the original paper.
- Can extract embeddings from corresponding positions output from the encoder layers (or multiple layers). Can also utilize [CLS] embedding as a sentence embedding to pass on to additional layers when fine-tuning
- Fine-tuning using BERT was shown to achieve state-of-the-art performance across 11 different NLP tasks spanning sentiment analysis, question answering, textual entailment, etc.

Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.

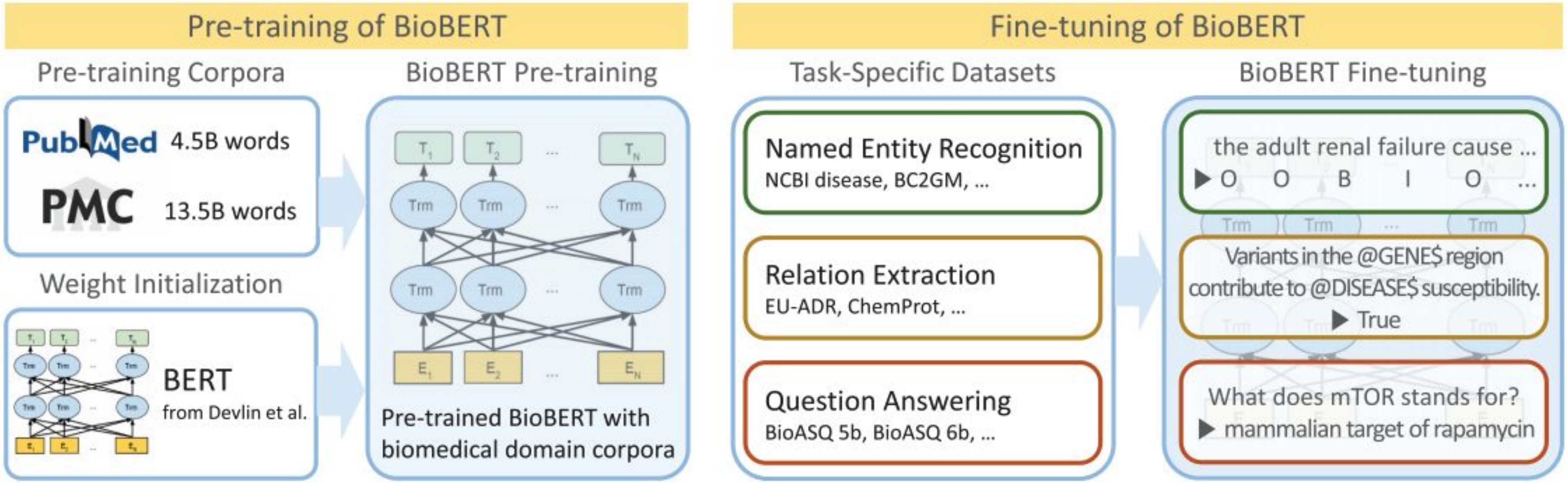
BERT: Highly successful transfer learning through learning bidirectional representations with a "Transformer" architecture

- Covered main idea of BERT, but there are more details that you can find in the original paper.
- Can extract embeddings from corresponding positions output from the encoder layers (or multiple layers). Can also utilize [CLS] embedding as a sentence embedding to pass on to additional layers when fine-tuning
- Fine-tuning using BERT was shown to achieve state-of-the-art performance across 11 different NLP tasks spanning sentiment analysis, question answering, textual entailment, etc.

Aside: GPT language models (now on GPT-3, the 3rd generation) from OpenAI also worth being aware of. Some differences from BERT, but similar concepts, also transformer-based -- if you understand BERT you can understand GPT. Architectural difference (using decoders w/ self-attention) allows also generating novel text, has received lots of press. But as of GPT-3, OpenAI is no longer open sourcing code, only the API.

Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018.

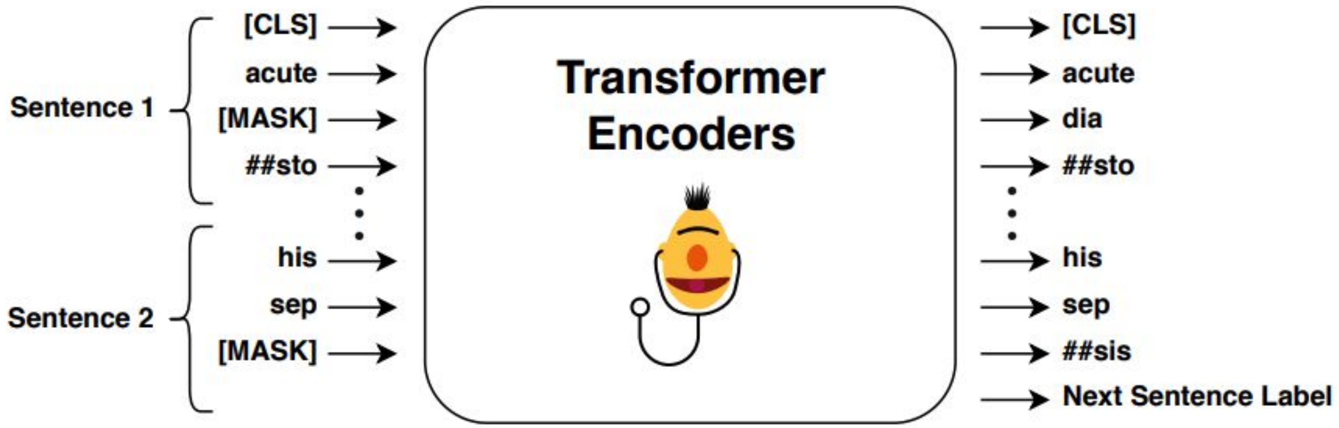
BioBERT: training on biomedical text corpora



Lee et al. BioBERT: a pre-trained biomedical language representation model for biomedical text mining, 2019.

ClinicalBERT: training on clinical notes (from MIMIC)

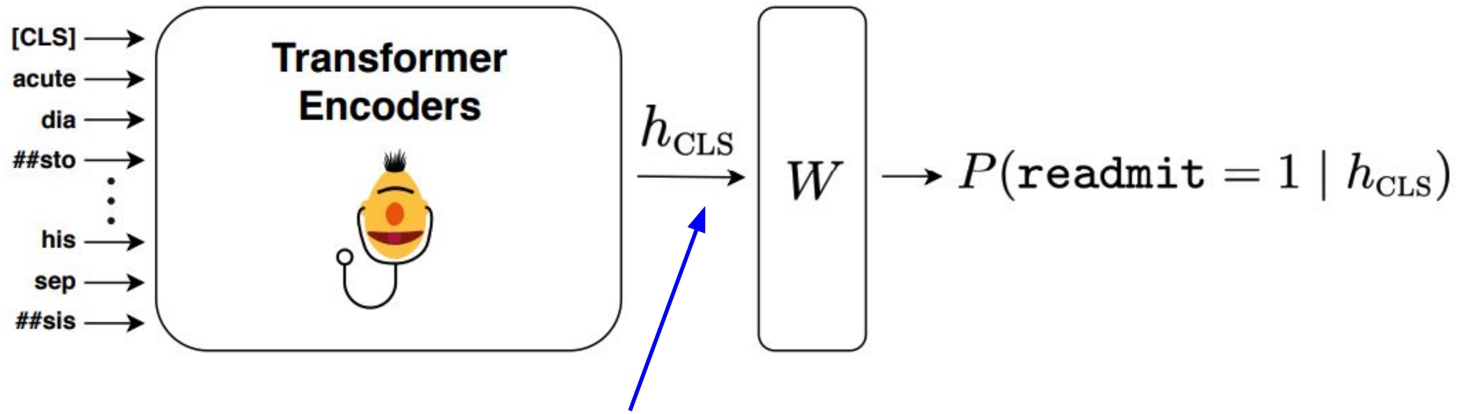
Training ClinicalBERT with the masked prediction and next sentence objectives:



Huang et al. ClinicalBert: Modeling Clinical Notes and Predicting Hospital Readmission, 2019.

ClinicalBERT: training on clinical notes (from MIMIC)

Fine-tuning ClinicalBERT for prediction of 30-day hospital readmission:

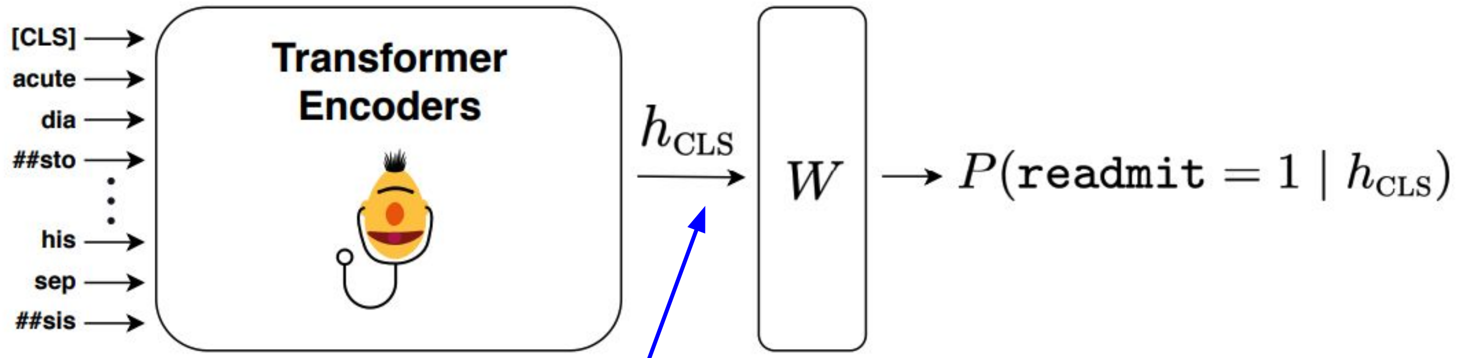


Use hidden state corresponding to [CLS] token

Huang et al. ClinicalBert: Modeling Clinical Notes and Predicting Hospital Readmission, 2019.

ClinicalBERT: training on clinical notes (from MIMIC)

Fine-tuning ClinicalBERT for prediction of 30-day hospital readmission:



Use hidden state corresponding to [CLS] token

When performing prediction from long sequences, obtain predictions for each sentence separately and then combine

Huang et al. ClinicalBert: Modeling Clinical Notes and Predicting Hospital Readmission, 2019.

Summary

- Text embeddings are useful ways to utilize data such as clinical notes in your models
- Saw earlier (but still commonly used) embedding methods such as word2vec, as well as very recent approaches such as BERT
- Many of these embedding approaches available online: can extract pre-trained embeddings from large corpora (e.g. Wikipedia and Google News), or fine-tune on your own data. Usually good to leverage the large data from pre-trained models!
- Also versions trained specifically on biomedical data, e.g. BioBERT and ClinicalBERT

Next Time: Fireside Chat on “Interdiscplinary Work in AI in Healthcare” with Dr. Matt Lungren, MD, Co-Director of the Stanford Artificial Intelligence in Medicine & Imaging Center -- Please come live to the zoom if you can!