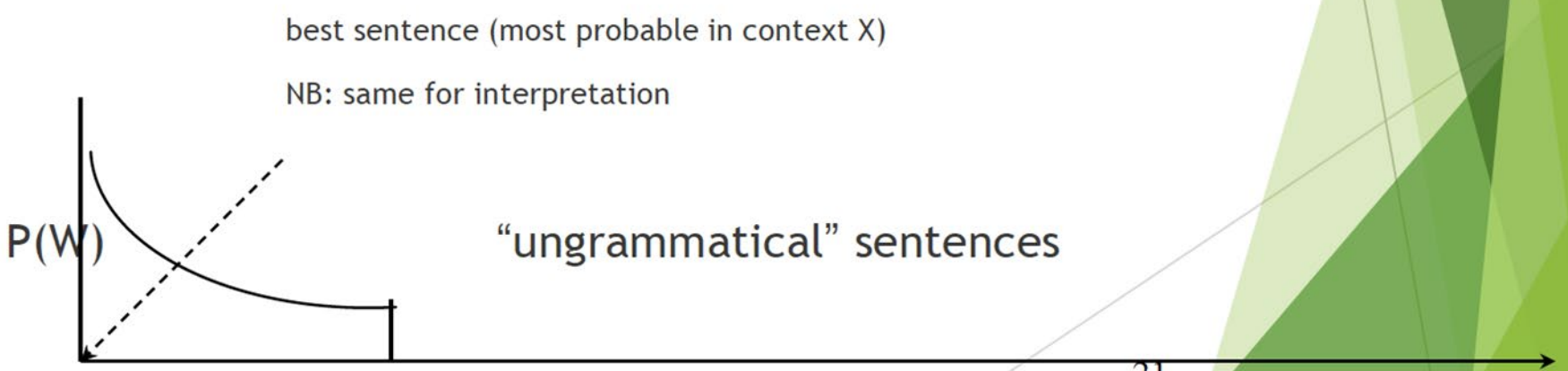
AI · BigData Academy
NLP Lab. 1

Isoft Lab 백성빈, 이지현, 이채빈

Statistical model, Deep learning model

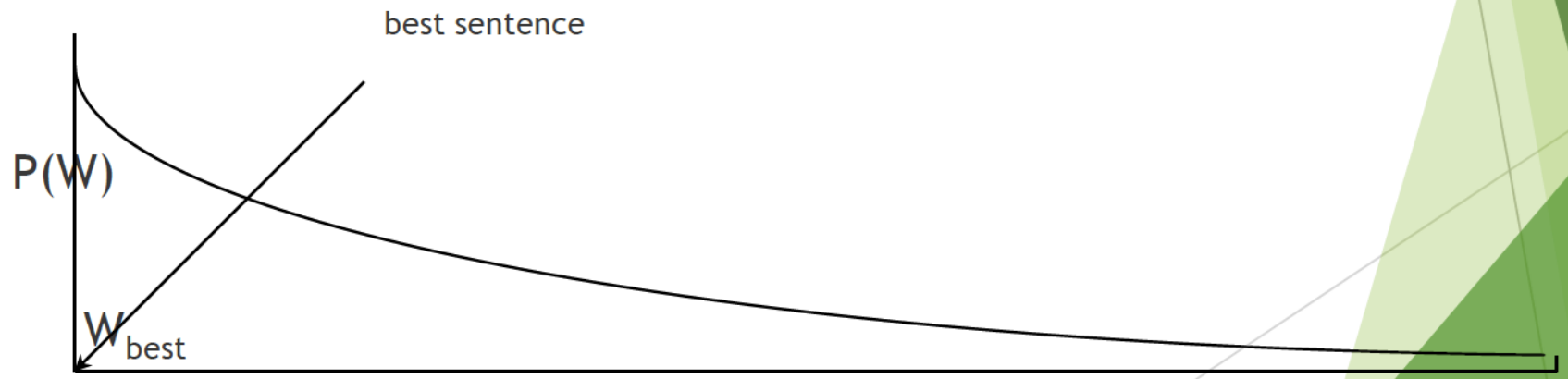
Statistical NLP

- Imagine:
  - Each sentence W = { w_1, w_2, ..., w_n } gets a probability P(W|X) in a context X (think of it in the intuitive sense for now)
  - For every possible context X, sort all the imaginable sentences W according to P(W|X):
  - Ideal situation:

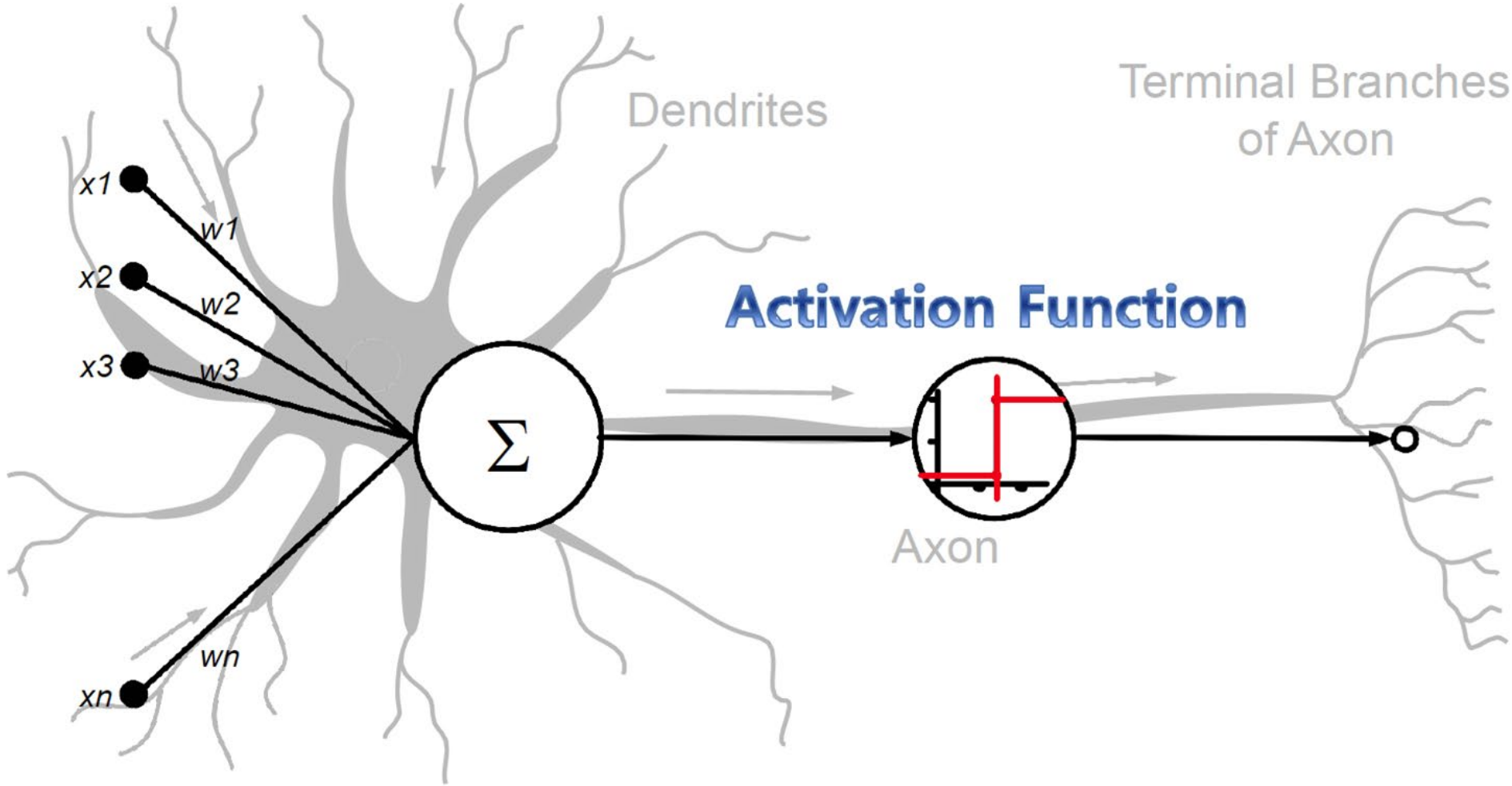


Real World Situation

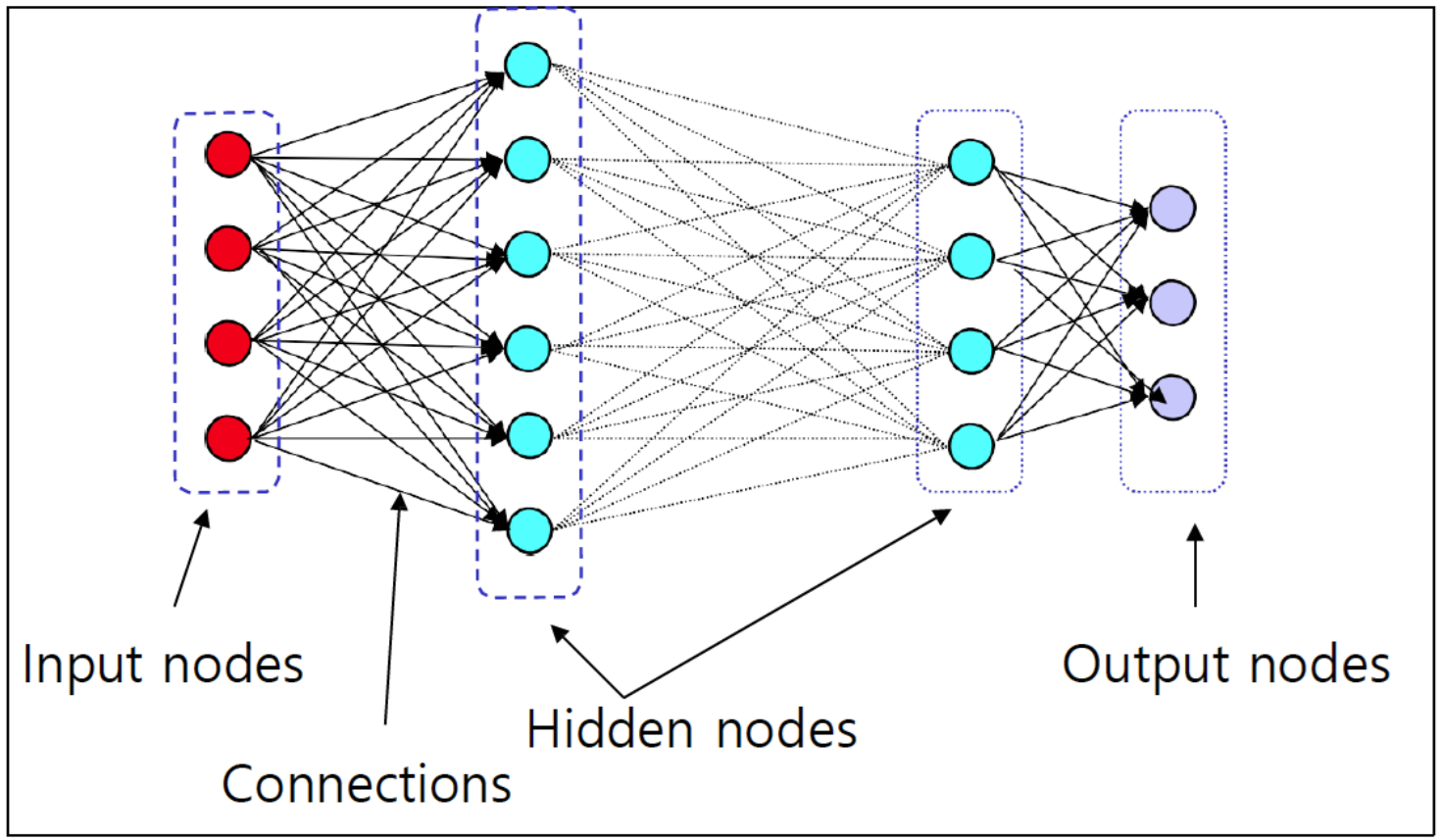
- Unable to specify set of grammatical sentences today using fixed “categorical” rules (maybe never)
- Use statistical “model” based on **REAL WORLD DATA** and care about the best sentence only (disregarding the “grammaticality” issue)



Artificial Neural Networks (ANN)



Layered Networks

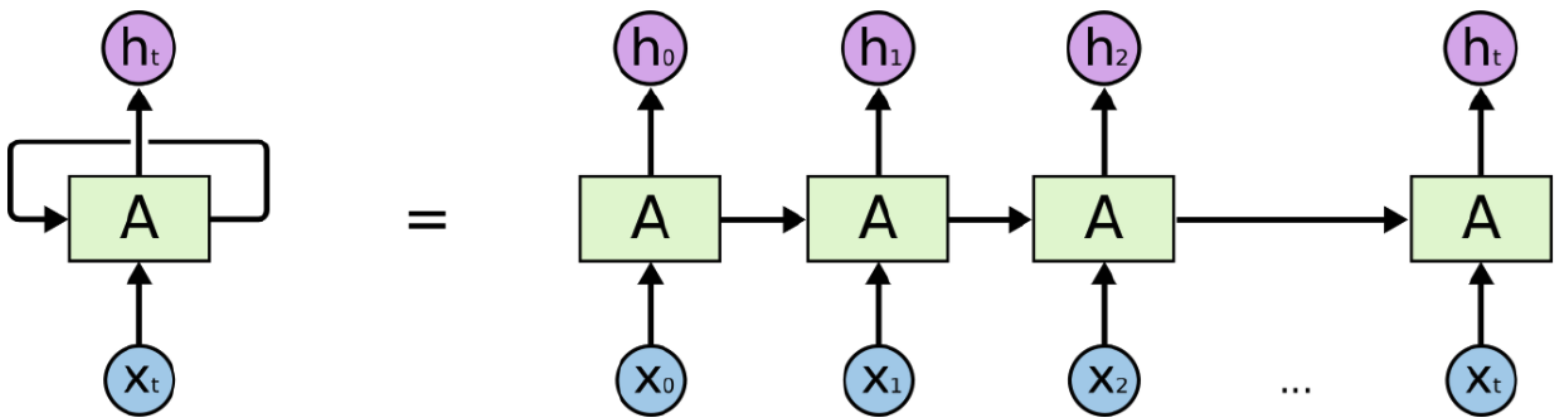


$$Output: \quad y = f(w^1 x + w_i^2 x + w_i^3 x + \cdots + w_i^m x_m)$$
$$= f(\sum_j w_i^j x_j)$$

Vanilla recurrent neural networks (RNNs)

- RNNs have connections from the outputs of previous time steps to inputs of next time steps

- For sequential data, a RNN usually computes hidden state h_t from the previous hidden state h_{t-1} and the input x_t
  - $h_t = \sigma(W_h h_{t-1} + W_x x_t + b)$

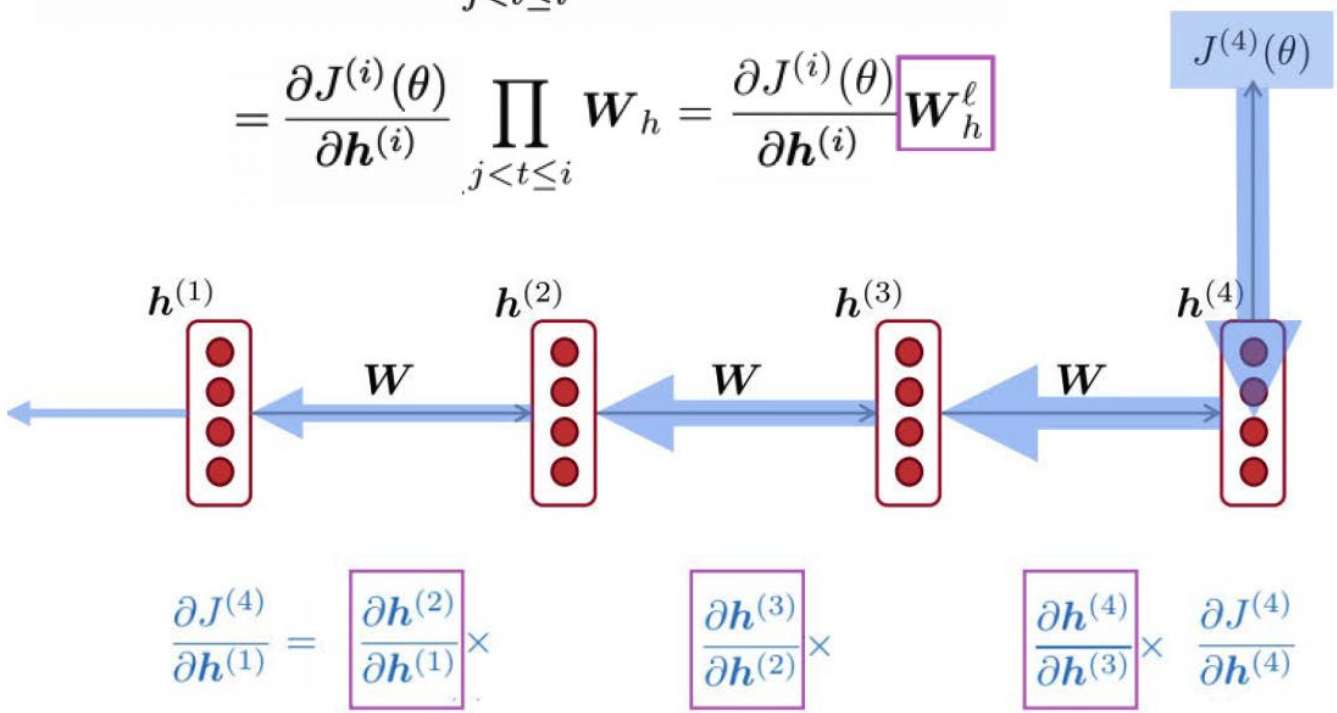


Vanishing gradient problem

- $h_t = \sigma(W_h h_{t-1} + W_x x_t + b)$
- Let's assume σ is the identity function

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$$

$$= \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \boldsymbol{W}_h = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boldsymbol{W}_h^{\ell}$$

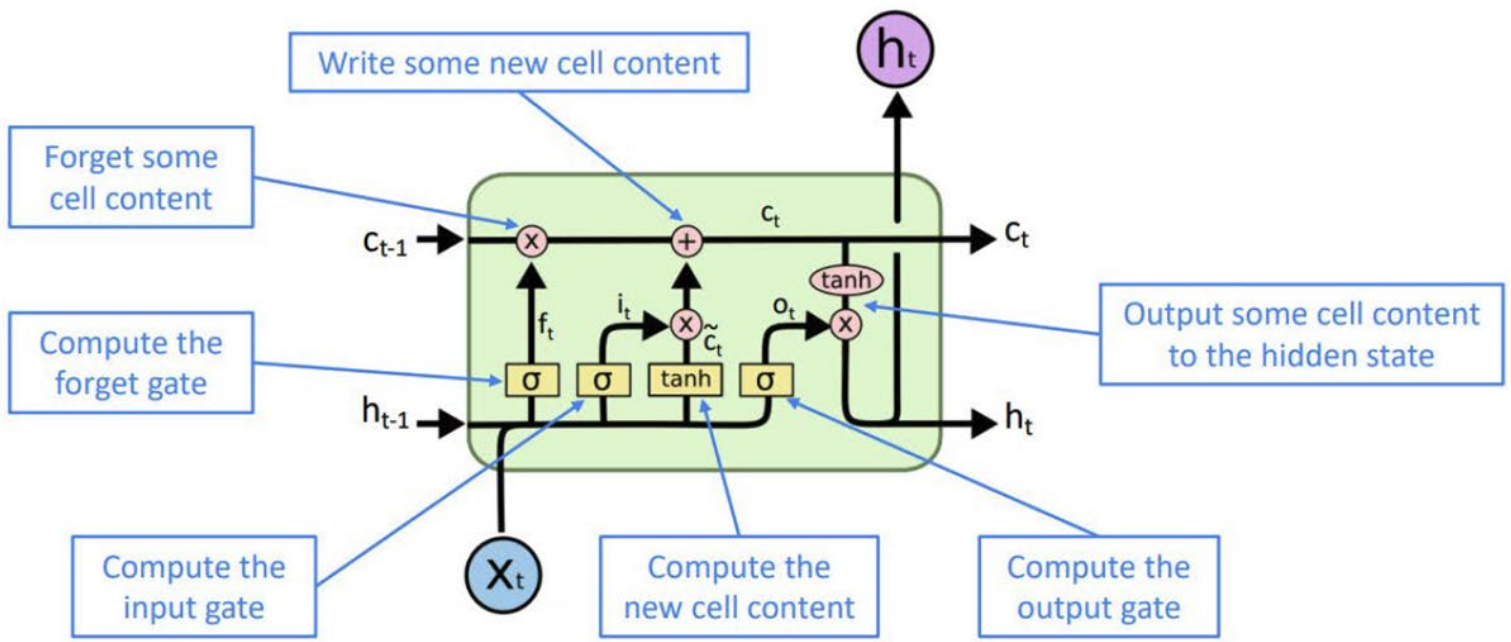


If all $\frac{\partial h^t}{\partial h^{t-1}} < 1 \rightarrow \frac{\partial J^t}{\partial h^1} \approx 0$

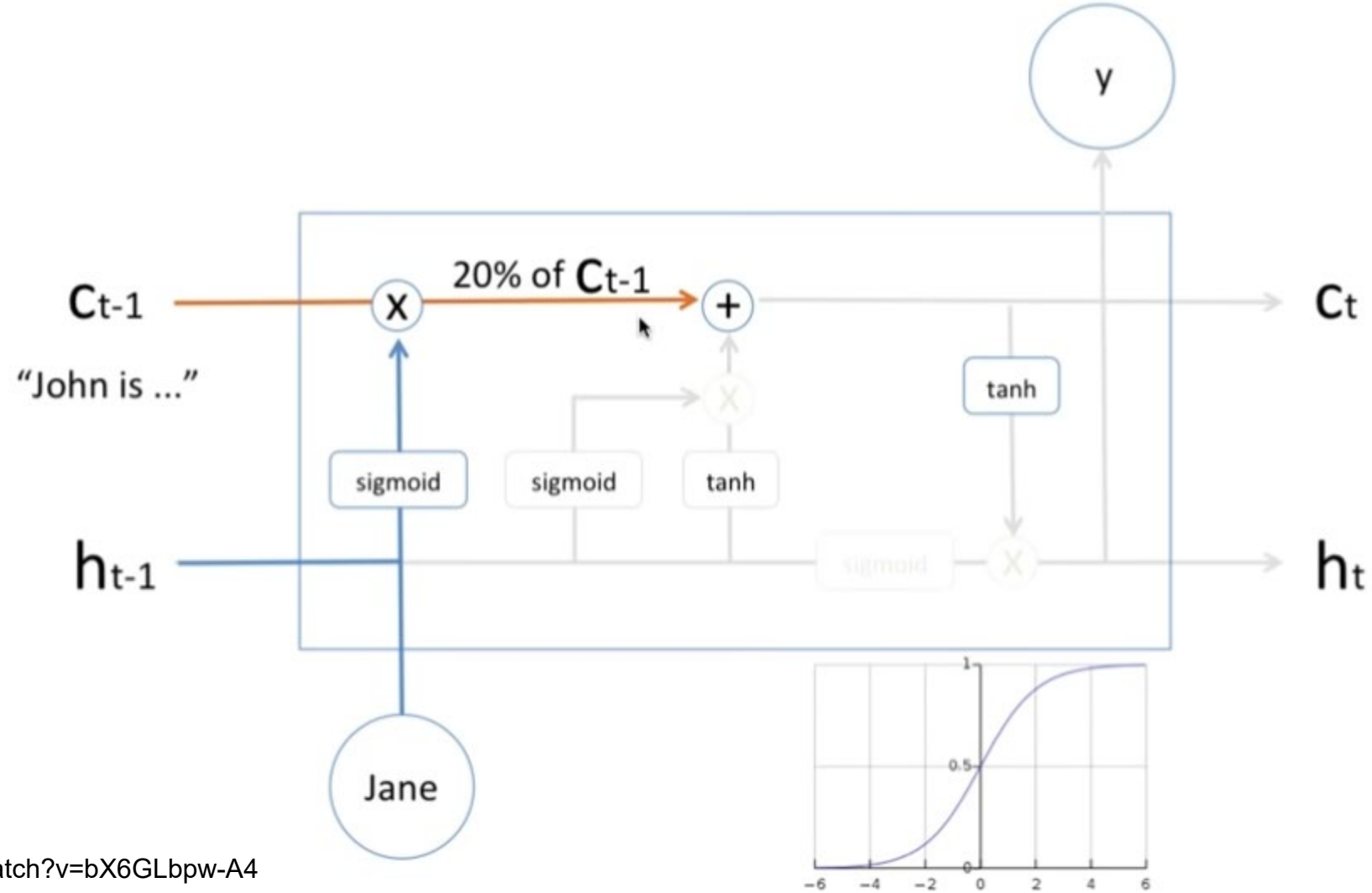
Long short-term memory networks (LSTMs)

- LSTMs explicitly keep and update cell memory $c^{(t)}$ by
  - Removing the previous cell content $c^{(t-1)}$ by multiplying it with $f^{(t)}$
  - Adding the new cell content $\tilde{c}^{(t)}$ multiplied by $i^{(t)}$
- LSTMs produce output $h^{(t)} = o^{(t)} \circ \tanh c^{(t)}$

$$
\begin{aligned}
\boldsymbol{f}^{(t)} &= \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right) \\
\boldsymbol{i}^{(t)} &= \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right) \\
\boldsymbol{o}^{(t)} &= \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right) \\
\tilde{\boldsymbol{c}}^{(t)} &= \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right) \\
\boldsymbol{c}^{(t)} &= \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)} \\
\boldsymbol{h}^{(t)} &= \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}
\end{aligned}
$$

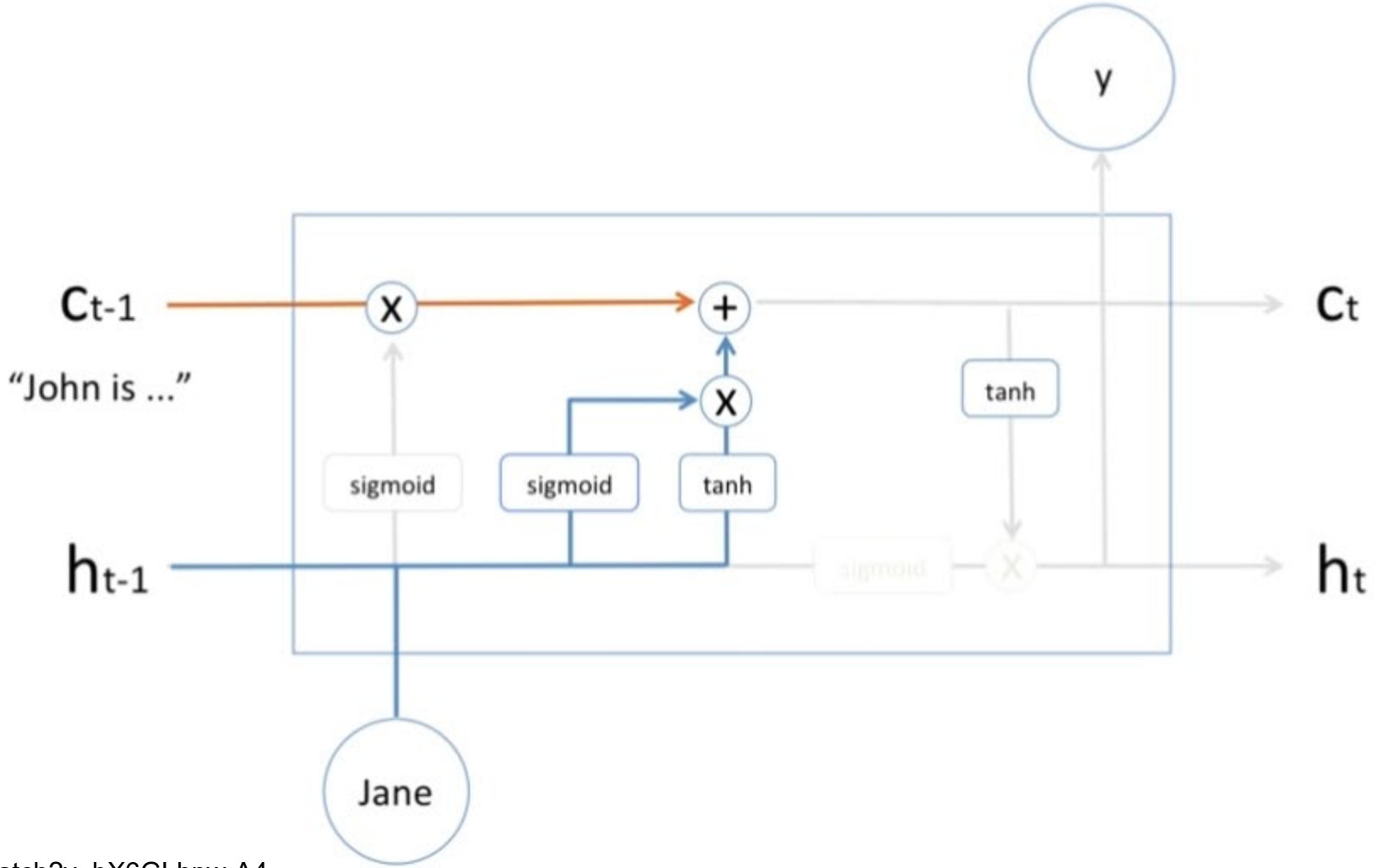


LSTM cell: forget mechanism



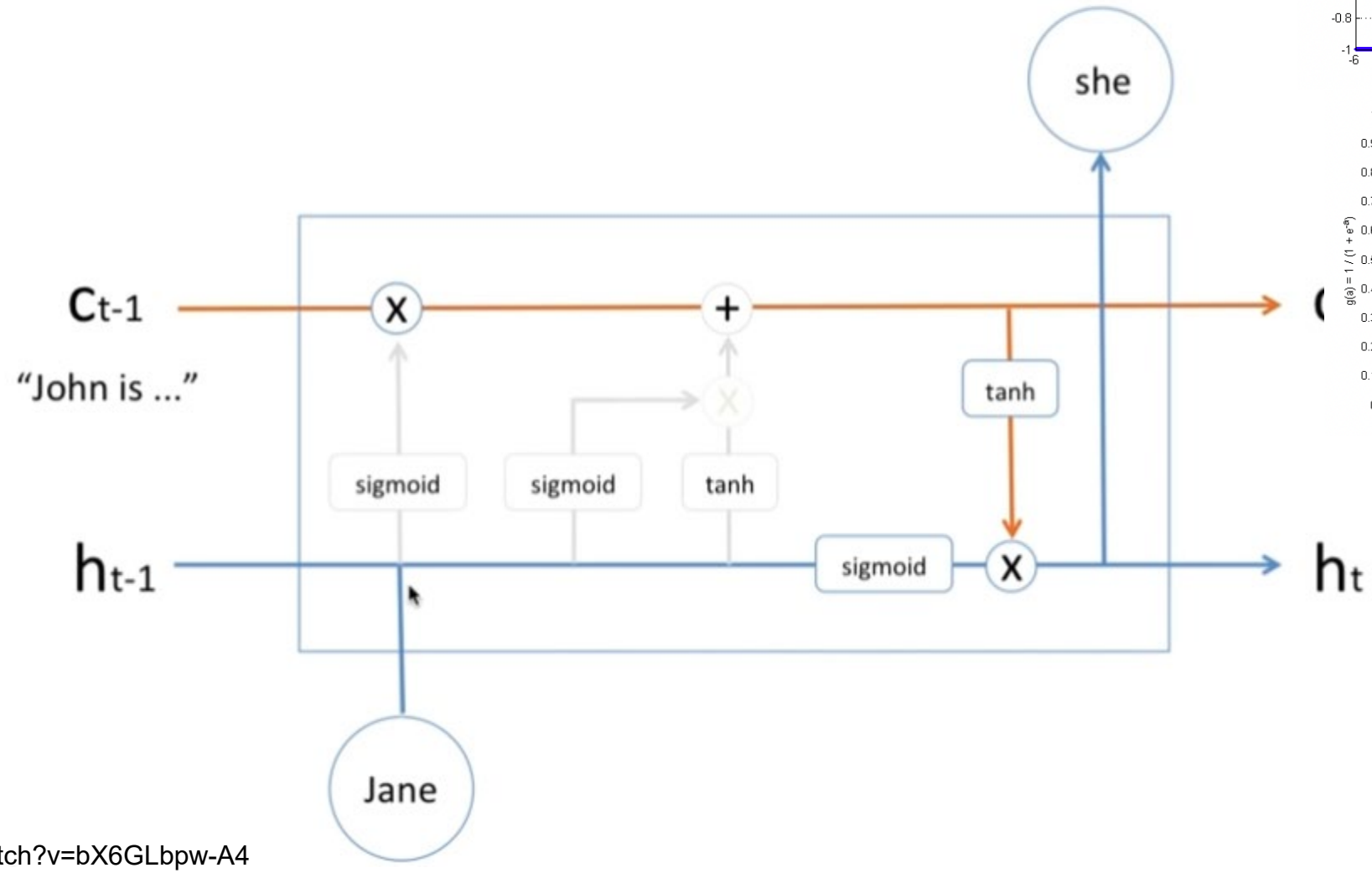
https://www.youtube.com/watch?v=bX6GLbpw-A4

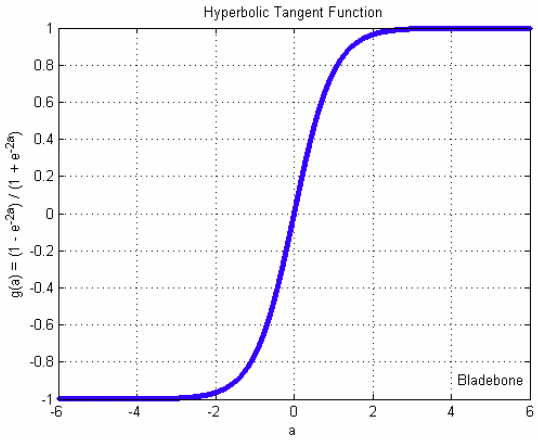
LSTM cell: input mechanism

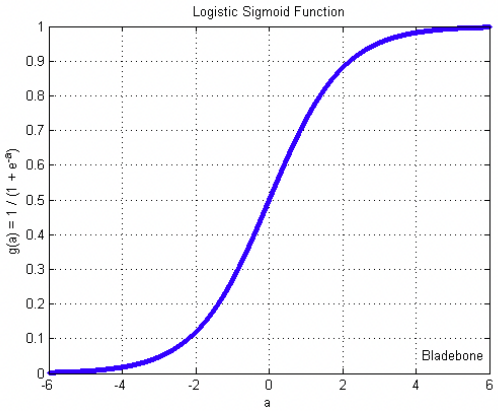


https://www.youtube.com/watch?v=bX6GLbpw-A4

LSTM cell: output mechanism







https://www.youtube.com/watch?v=bX6GLbpw-A4

Embeddings

Word Vector

- Represent words as vectors

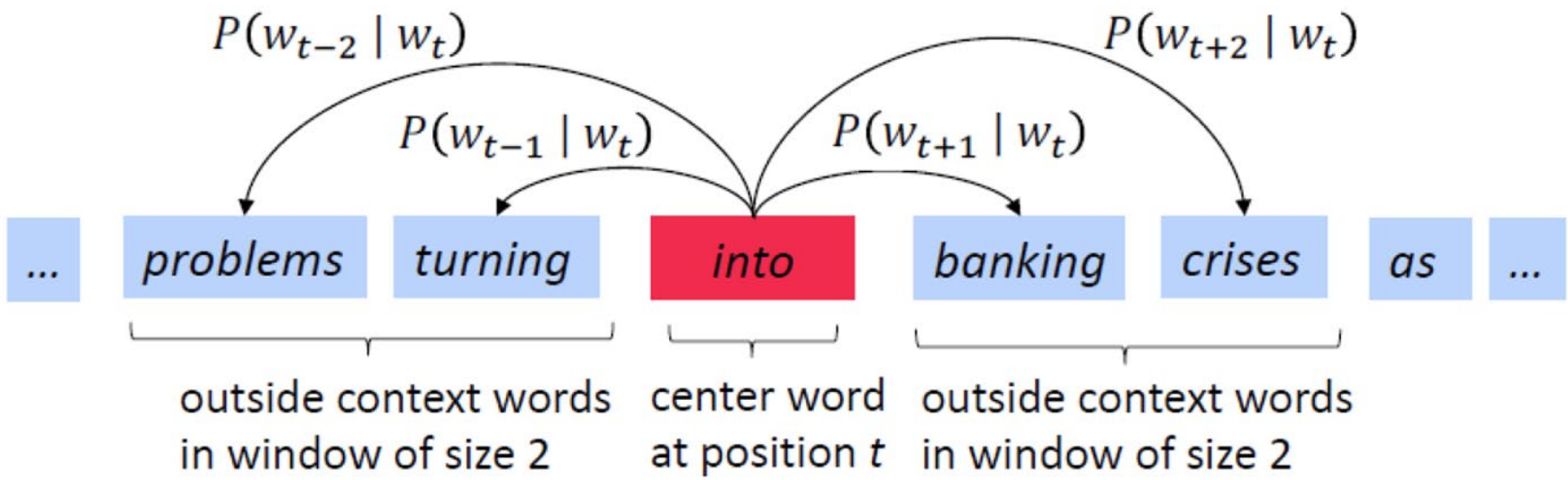
expect =

$$\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

Word Vector

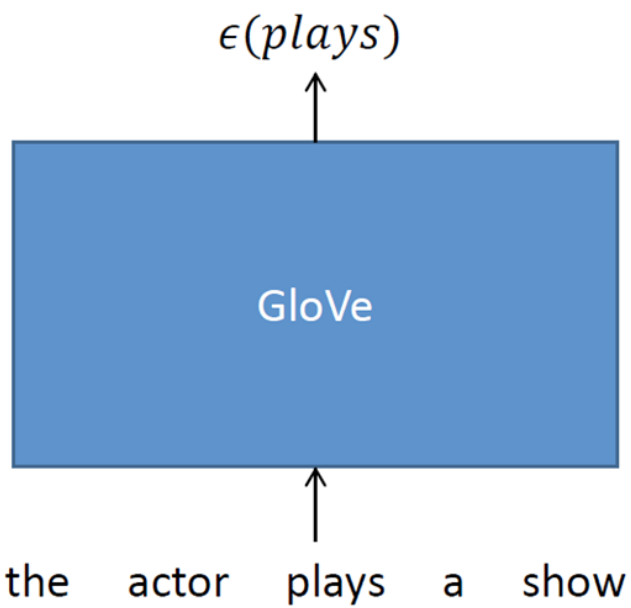
- Distributional semantics : A word's meaning is given by the words that frequently appear close-by
- *"You shall know a word by the company it keeps"*
- Word2vec objective function (skip-grams)

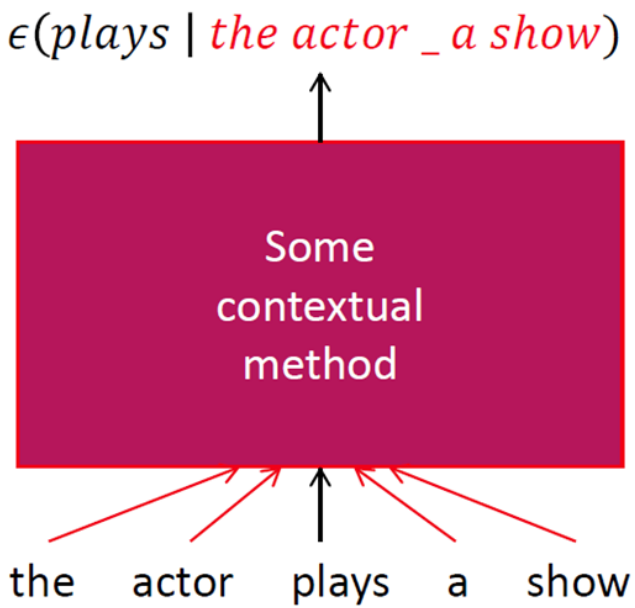
$$J(\theta) = -\frac{1}{T}\log L(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m \\ j\neq 0}} \log P(w_{t+j} \mid w_t;\theta)$$



Contextual word embedding

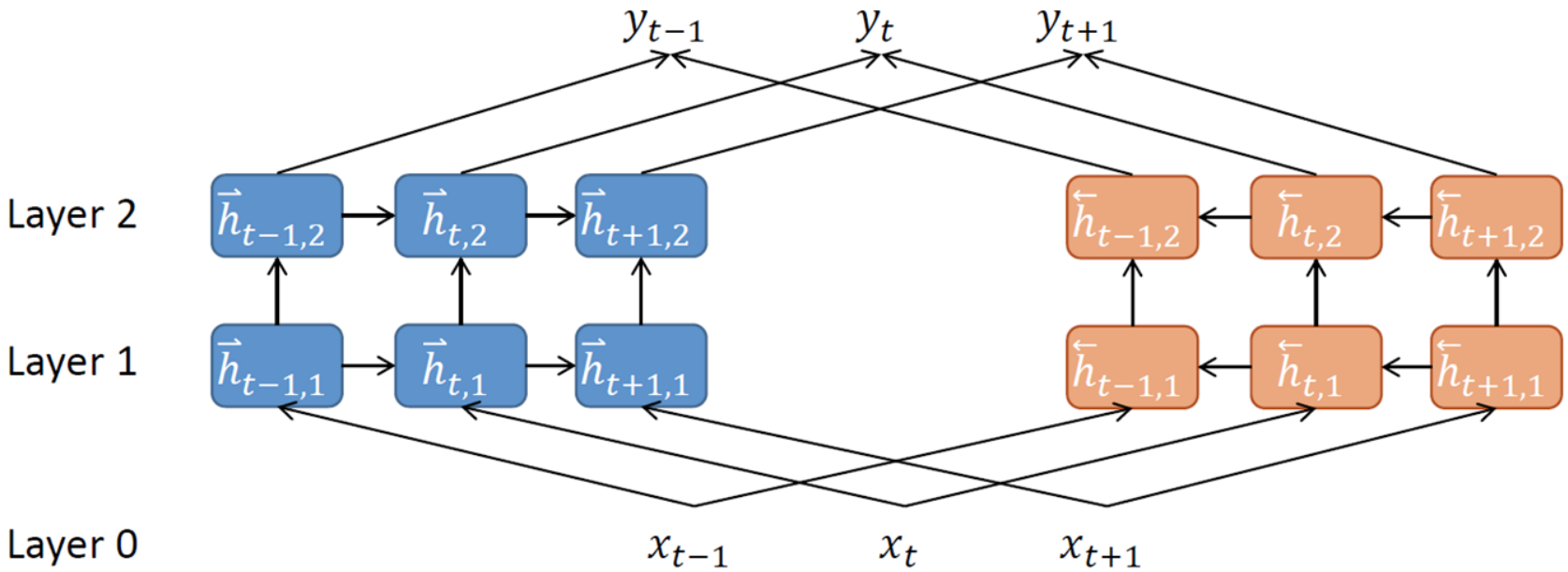
- A word's **contextual embedding** must consider its context





ELMo: Embeddings from Language Model

- Multi-layer bidirectional LSTM language model



$$R_k = \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\}$$
$$= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\},$$

$\mathbf{h}_{k,0}^{LM} = x_k^{LM}$ (token representation; GloVe)

$\mathbf{h}_{k,j}^{LM} = \vec{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM}$ (LSTM state)

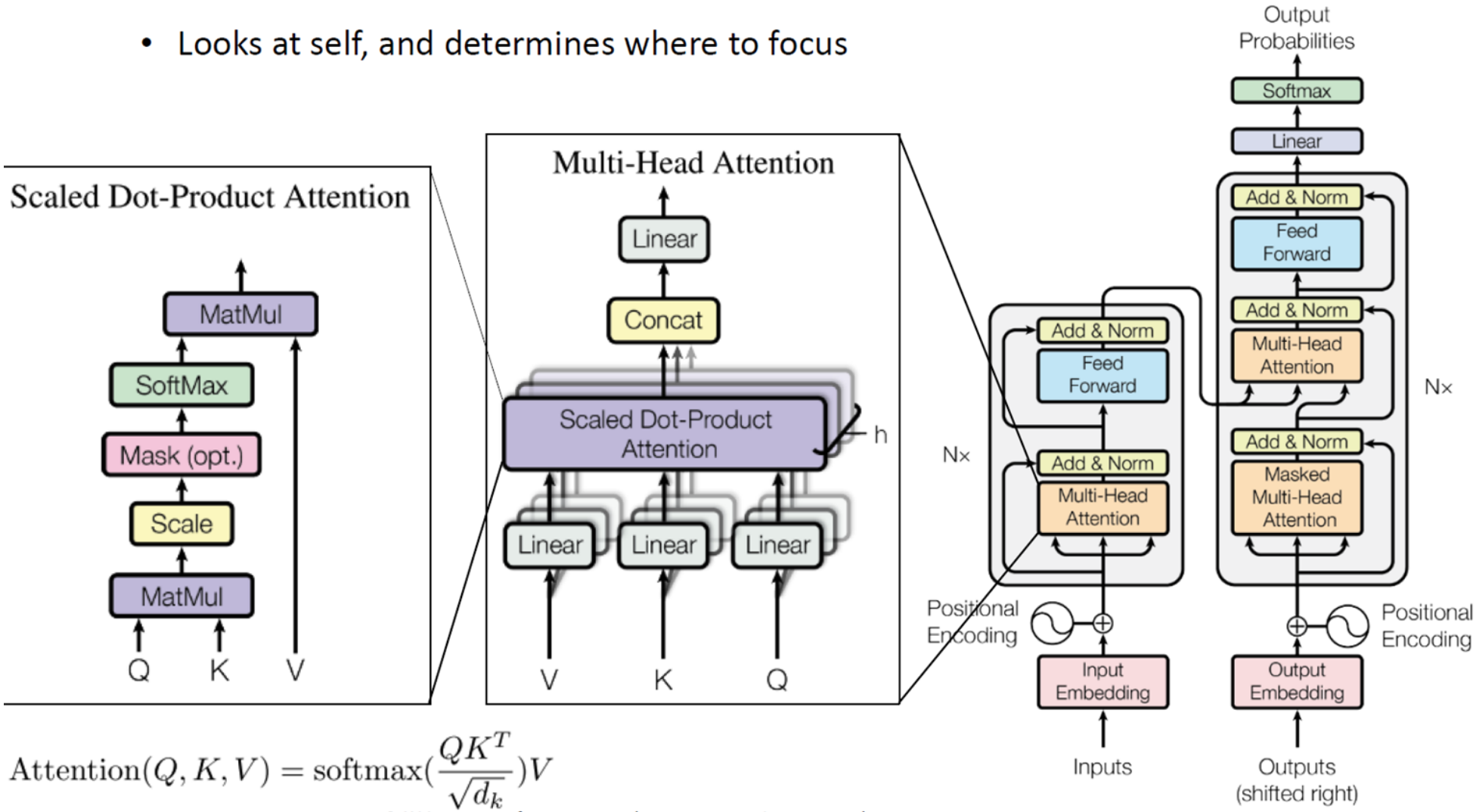
$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}.$$

γ^{task}: scale (hyper-parameter)

s_j^{task}: weight (learned)

Transformer

- Parallel self-attention
  - Looks at self, and determines where to focus



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

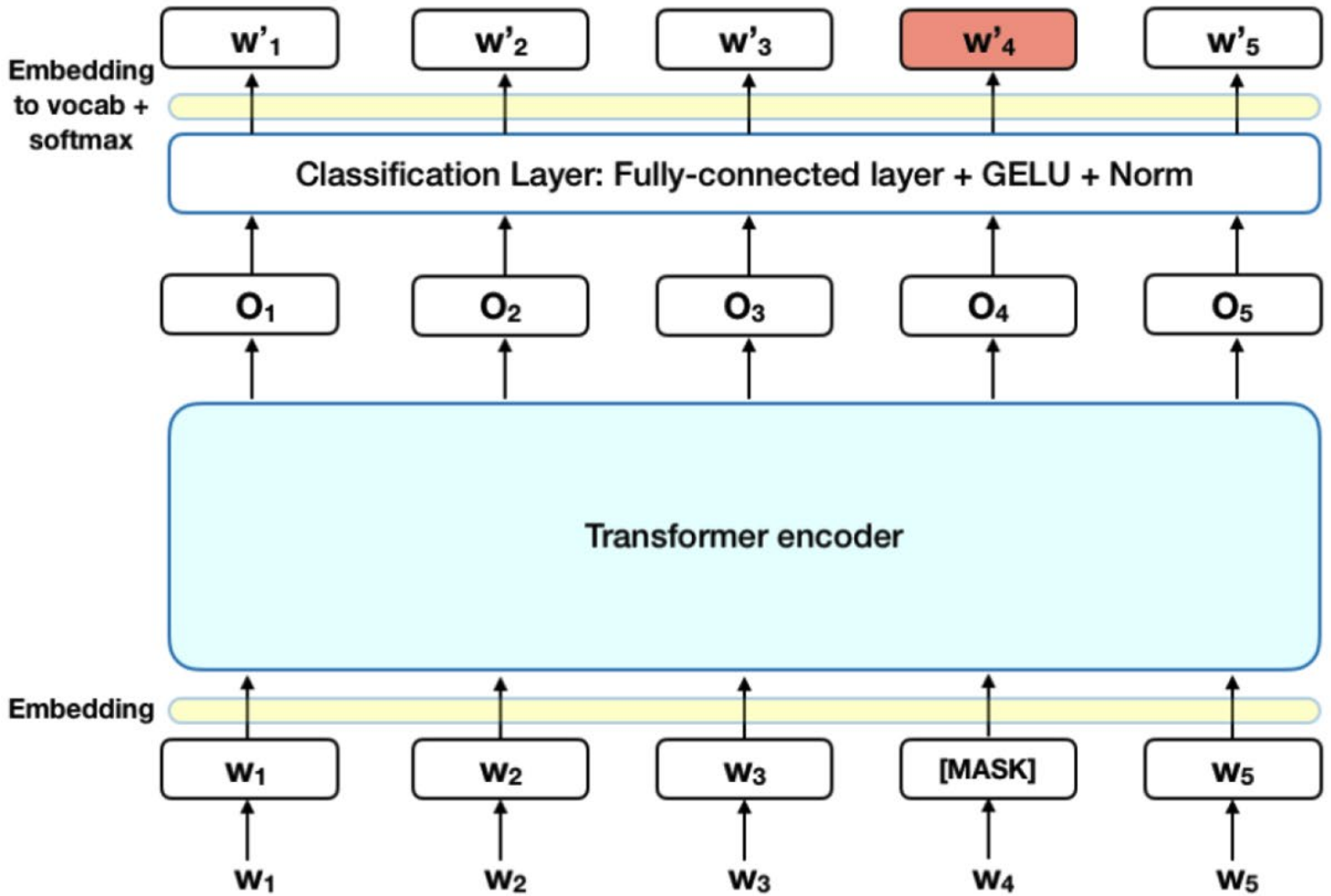
Q,K,V – vectors for every word, output attention summed up;
head means different Q,K,V vector with different weights

Vaswani, Ashish, et al. "Attention is all you need." *NIPS 2017*

BERT: Bidirectional Encoder Representations from Transformers

- Training 1. Masked words prediction
  - 15% of words are [MASK]ed

*GELU: Gaussian error linear unit



BERT: Bidirectional Encoder Representations from Transformers

- Training 2. Next sentence prediction
  - To understand texts more than a sentence

Input = `[CLS] the man went to [MASK] store [SEP]`

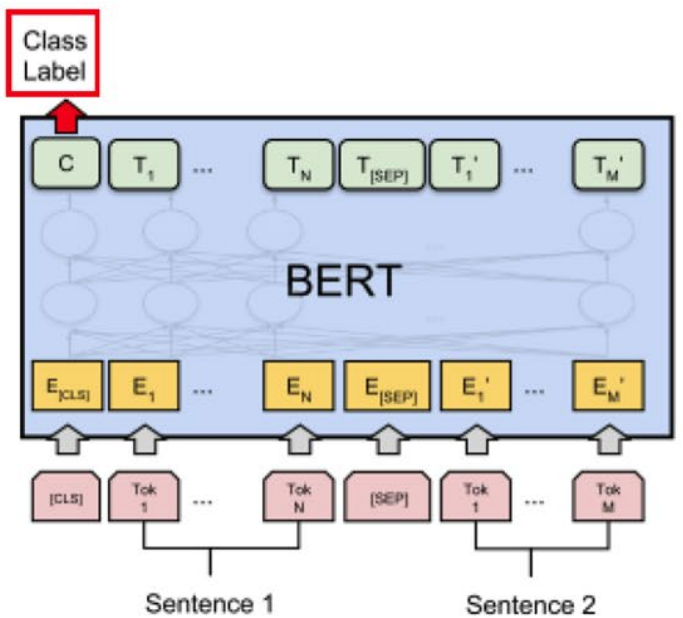
`he bought a gallon [MASK] milk [SEP]`

Label = `IsNext`

Input = `[CLS] the man [MASK] to the store [SEP]`

`penguin [MASK] are flight ##less birds [SEP]`

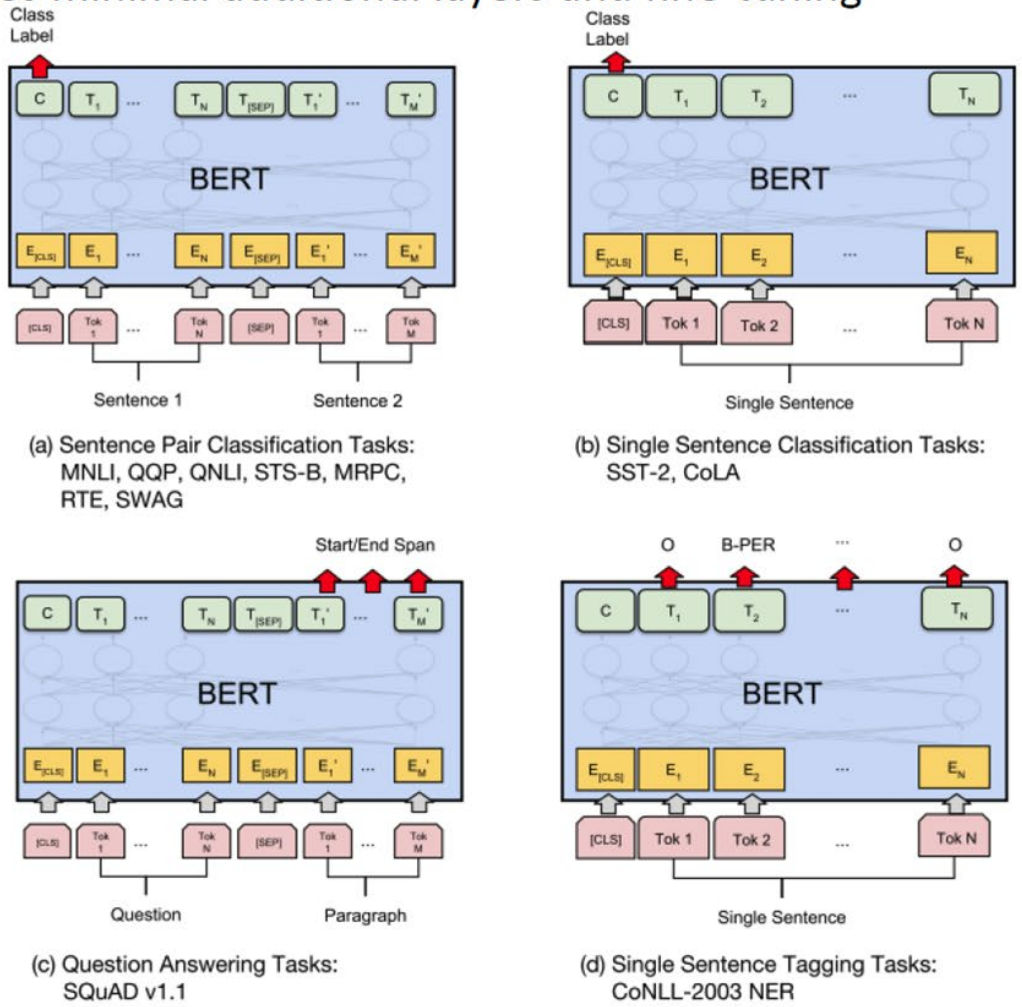
Label = `NotNext`



BERT: Bidirectional Encoder Representations from Transformers

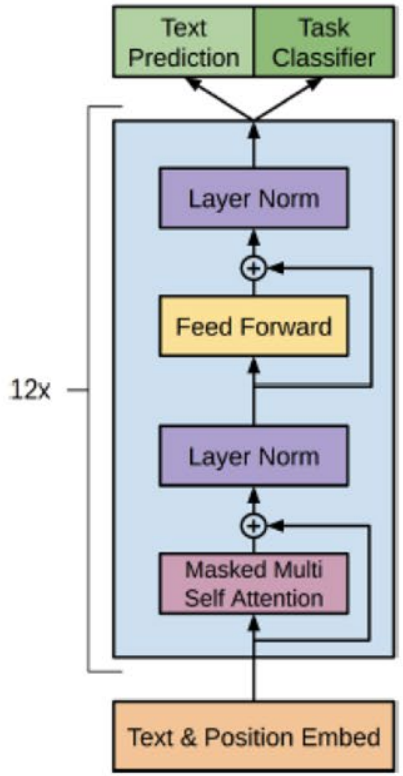
- BERT as universal pre-trained model for NLP
  - BERT requires minimal additional layers and fine-tuning

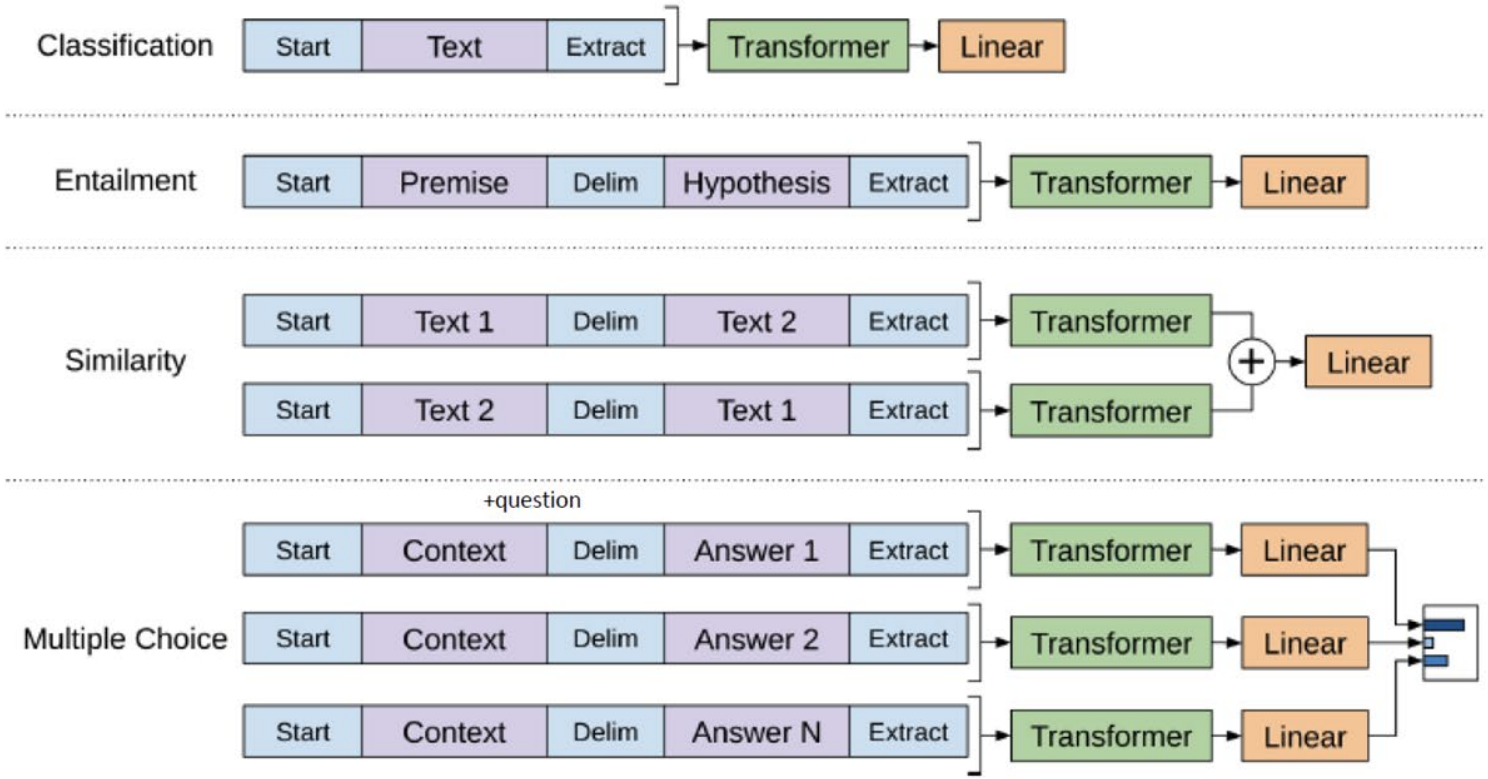
*GLUE benchmark task



Devlin, Jacob, et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." *NAACL 2019*

GPT(Generative Pre-Training)





$$L_1(\mathcal{U}) = \sum_i \log P(u_i|u_{i-k}, \ldots, u_{i-1}; \Theta)$$

*auto-regressive

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y|x^1, \ldots, x^m)$$

$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda * L_1(\mathcal{C})$$

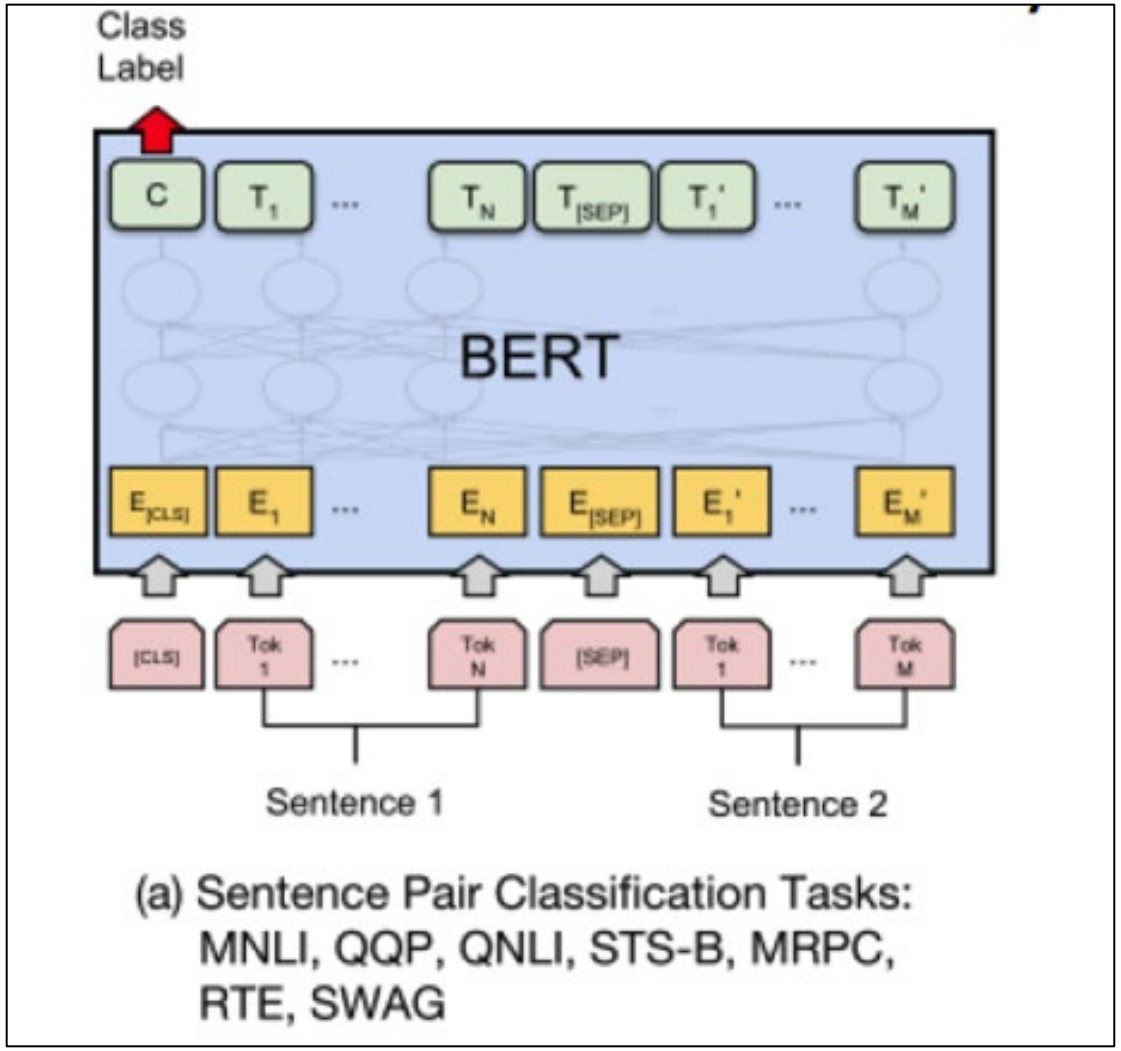
*GPT-2/3: zero/few-shot learning

In pre-training, optimize $L_1(u)$
u: Unlabeled dataset
Θ: Model parameters

In fine-tuning, optimize $L_3(c)$
c: Labeled dataset
λ: Hyper-parameter weight

Task 별 아키텍처

Sentence pair classification



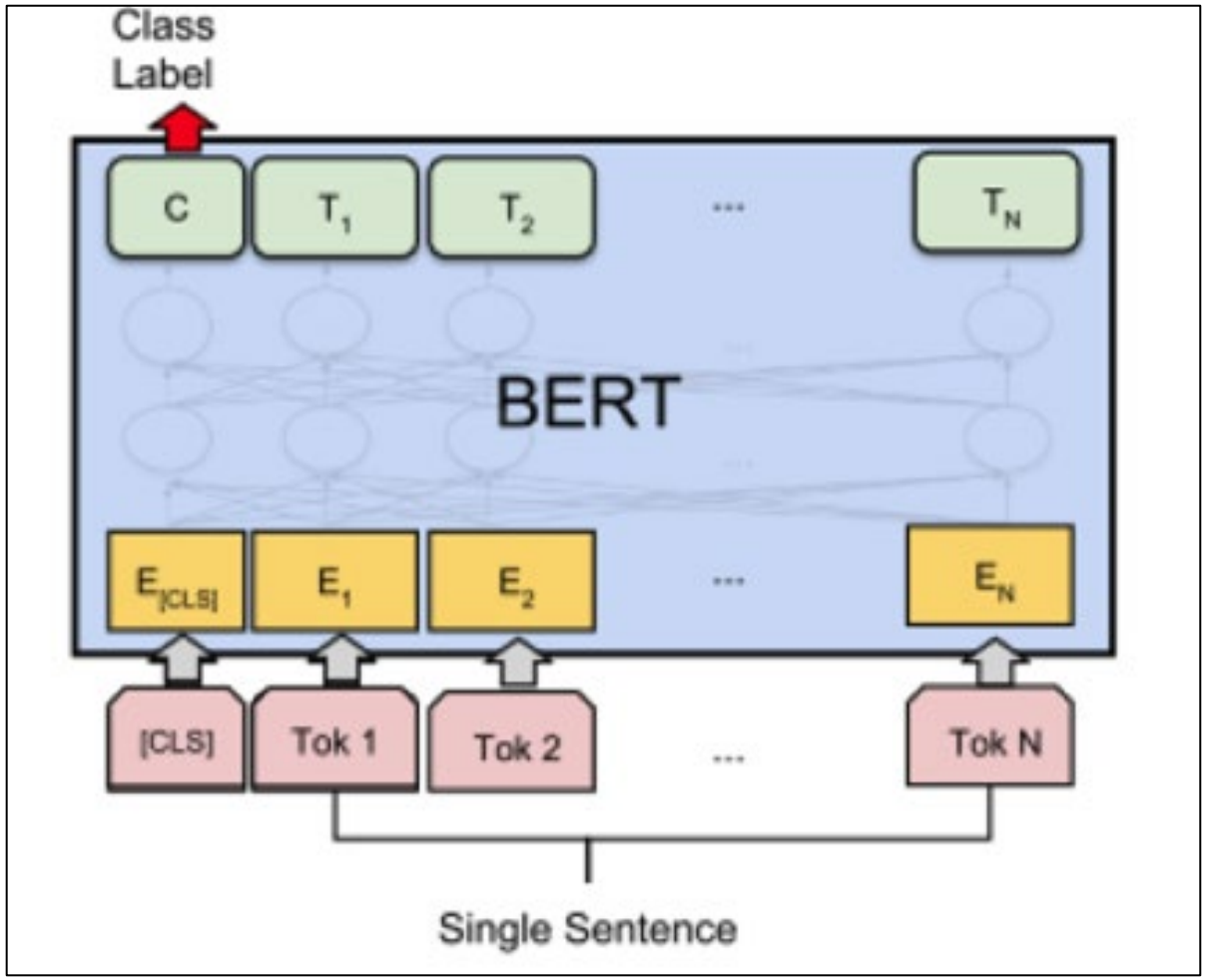
Sentence pair classification

SNLI(Stanford Natural Language Interence) Corpus

https://nlp.stanford.edu/projects/snli/

Text	Judgments	Hypothesis
A man inspects the uniform of a figure in some East Asian country.	contradiction C C C C C	The man is sleeping.
An older and younger man smiling.	neutral N N E N N	Two men are smiling and laughing at the cats playing on the floor.
A black race car starts up in front of a crowd of people.	contradiction C C C C C	A man is driving down a lonely road.
A soccer game with multiple males playing.	entailment E E E E E	Some men are playing a sport.
A smiling costumed woman is holding an umbrella.	neutral N N E C N	A happy woman in a fairy costume holds an umbrella.

Single sentence classification task



<Data Sample>

lend some dignity to a dumb story 0

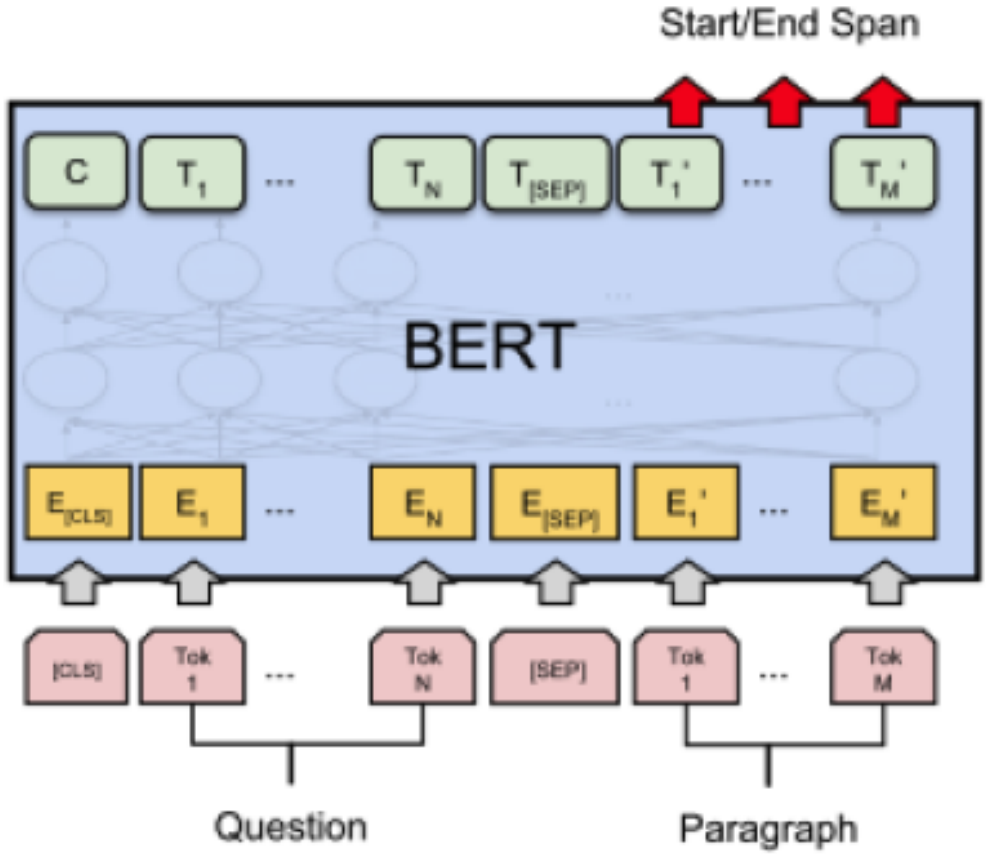
the greatest musicians 1

cold movie 0

with his usual intelligence and subtlety 1

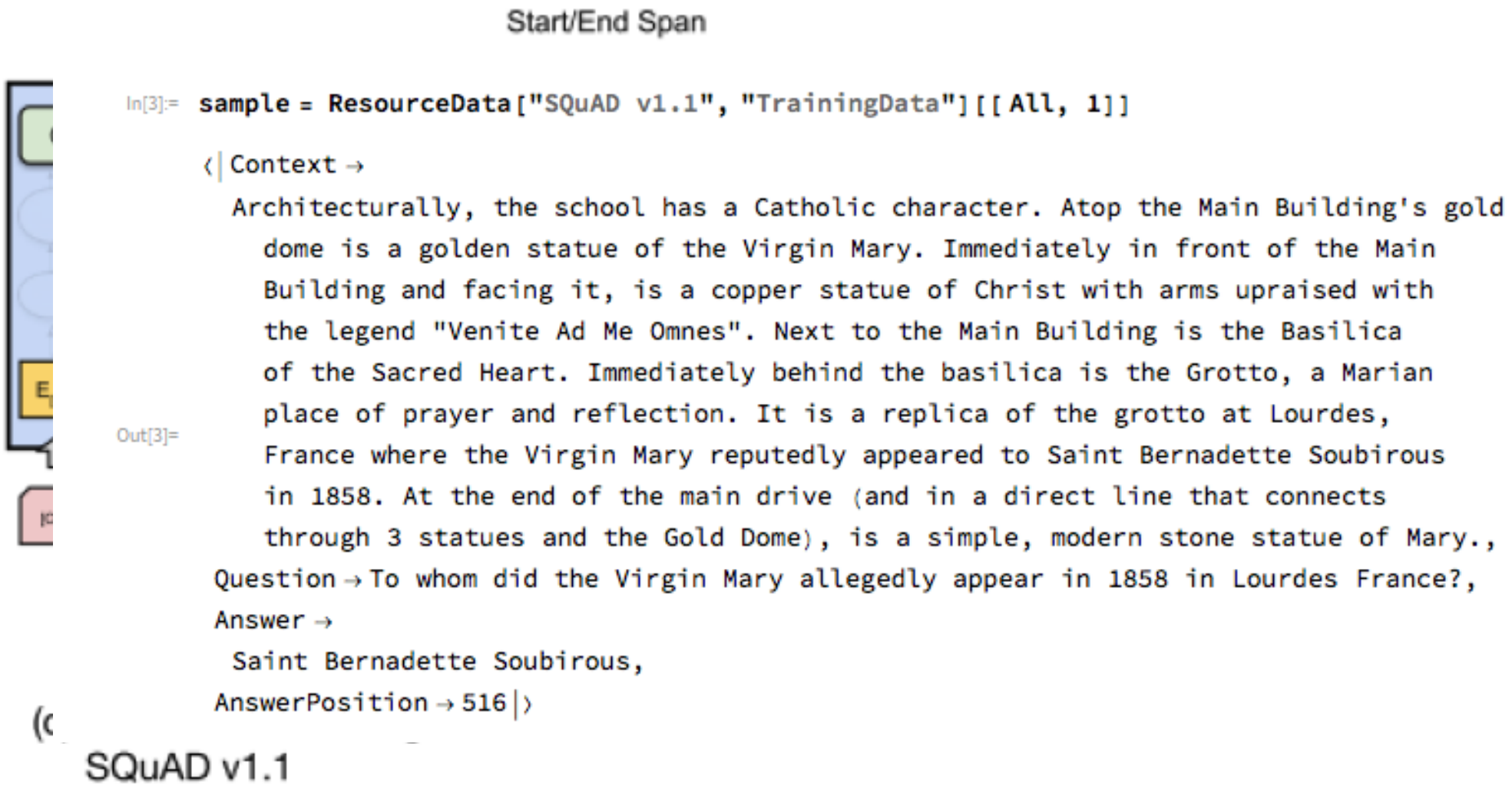
redundant concept 0

Question Answering

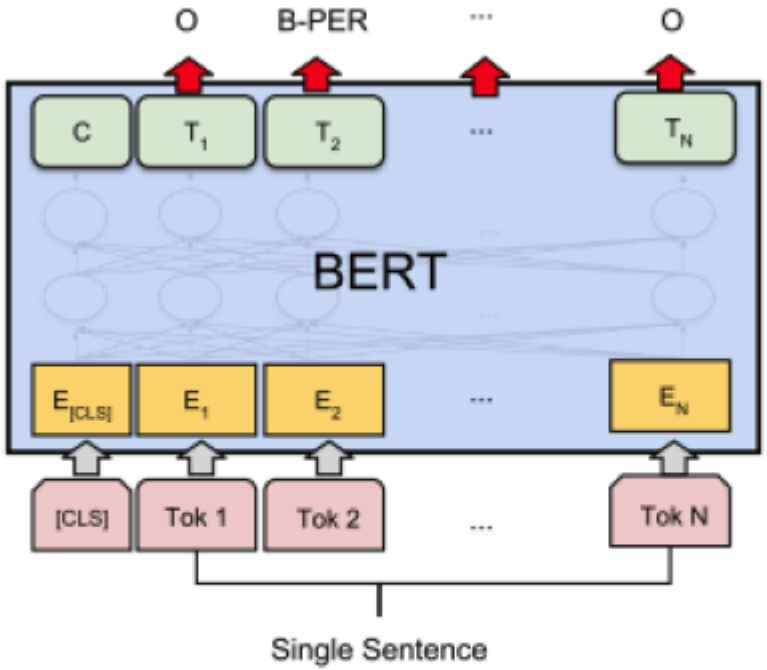


(c) Question Answering Tasks:
SQuAD v1.1

Question Answering



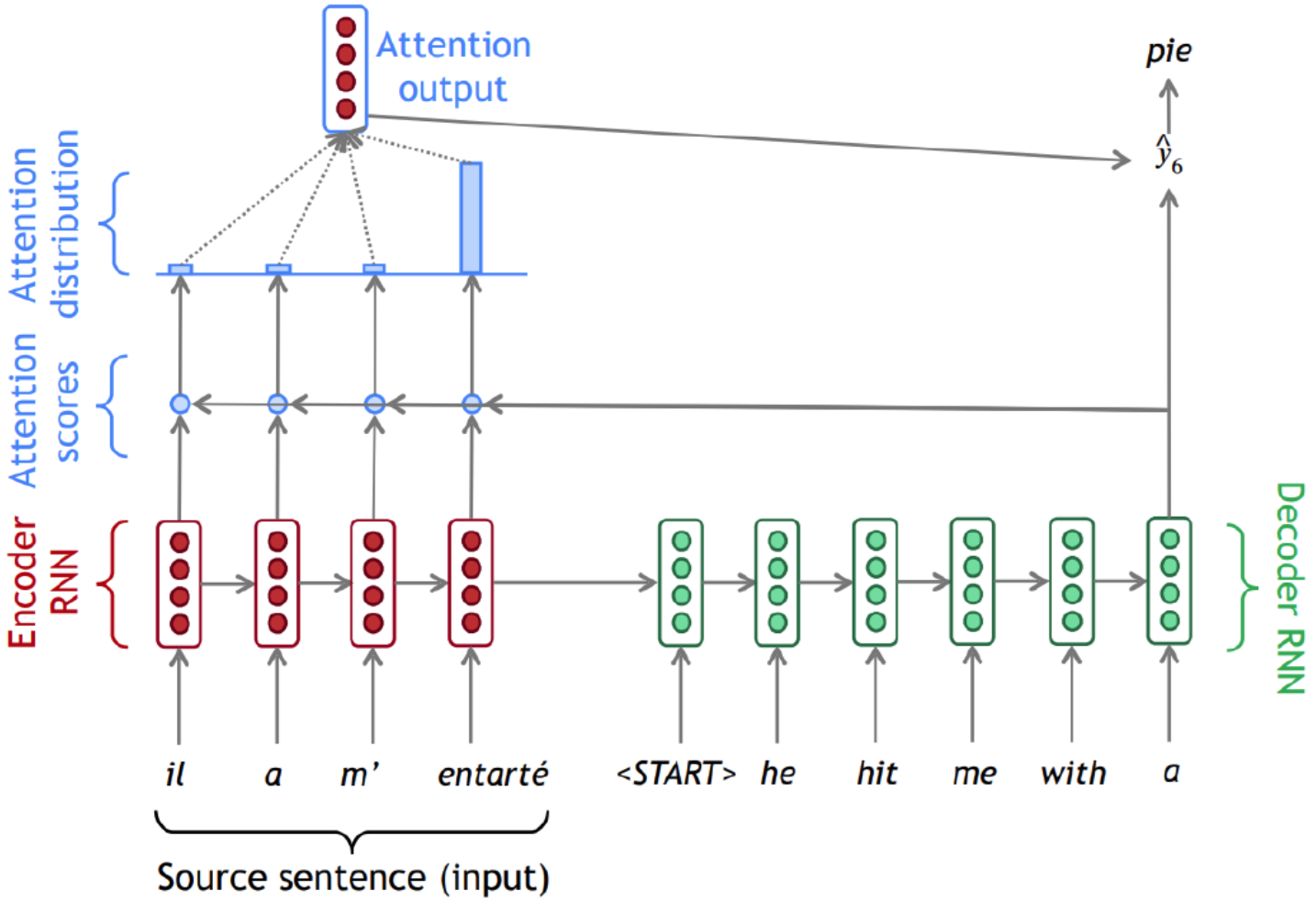
Single sentence tagging task



(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

U.N.	NNP	I-NP	I-ORG
official	NN	I-NP	O
Ekeus	NNP	I-NP	I-PER
heads	VBZ	I-VP	O
for	IN	I-PP	O
Baghdad	NNP	I-NP	I-LOC
.	.	O	O

NMT with attention

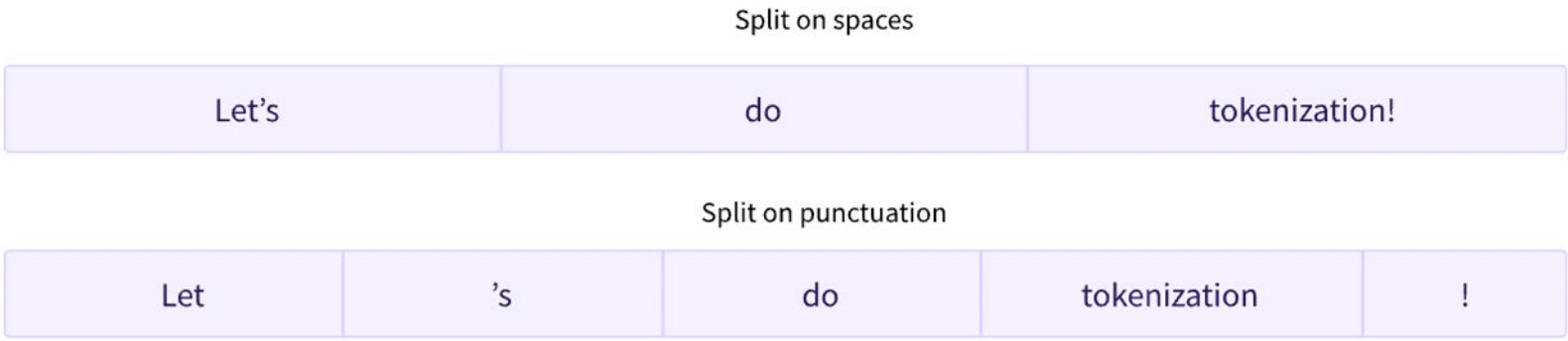


HuggingFace Tutorial

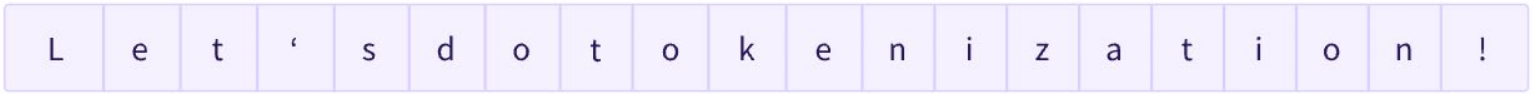
Tokenizer

Models can only process numbers.

so, tokenizers need to convert our text inputs to numerical data.



Word-based



Character-based

Subword tokenization

Let's </w>	do</w>	token	ization</w>	!</w>

Subword tokenization algorithms rely on the principle that frequently used words should not be split into smaller subwords, but rare words should be decomposed into meaningful subwords.

Byte-level BPE, as used in GPT-2

WordPiece, as used in BERT

SentencePiece or Unigram, as used in several multilingual models

```
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained("bert-base-cased")
```

```
tokenizer("Using a Transformer network is simple")
```

```
{'input_ids': [101, 7993, 170, 11303, 1200, 2443, 1110, 3014, 102],
 'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0, 0],
 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

Encoding

Translating text to numbers is known as encoding.

Encoding is done in a two-step process: the tokenization, followed by the conversion to input IDs

Step 1 : Tokenization

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")

sequence = "Using a Transformer network is simple"
tokens = tokenizer.tokenize(sequence)

print(tokens)
```

The output of this method is a list of strings, or tokens:

```
['Using', 'a', 'transform', '##er', 'network', 'is', 'simple']
```

Step 2 :From tokens to input IDs

```
ids = tokenizer.convert_tokens_to_ids(tokens)

print(ids)
```

```
[7993, 170, 11303, 1200, 2443, 1110, 3014]
```

Decoding

```
decoded_string = tokenizer.decode([7993, 170, 11303, 1200, 2443, 1110, 3014])
print(decoded_string)
```

```
'Using a Transformer network is simple'
```

Decoding is going the other way around: from vocabulary indices, we want to get a string.

Let’s code