Seq2seq & Attention

2023-2 KUBIG 방학세션
DL

0. Announcement

No real-time Class on Next Week!

⇒ Instead, Pytorch implementation of Transformer

- I would give you the youtube link on next week.
- Full course of transformer
- Refered to “Attention is All you need”
- GPU?
- After next week, we would prepare ”KUBIG Contest”

https://dacon.io/competitions/official/235747/overview/description

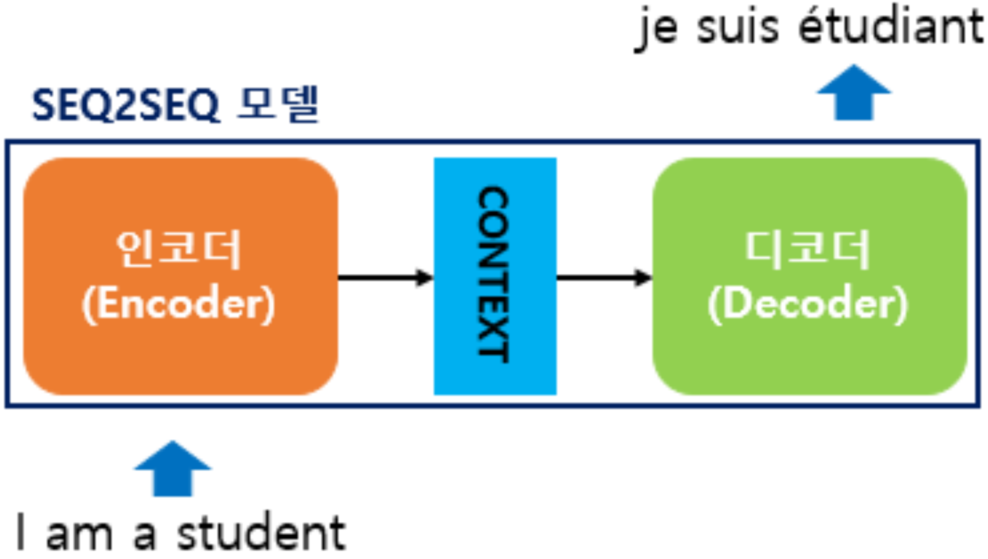
Category

1. Seq2Seq
2. Attention
3. Training Techniques
4. Q&A

1. Seq2Seq

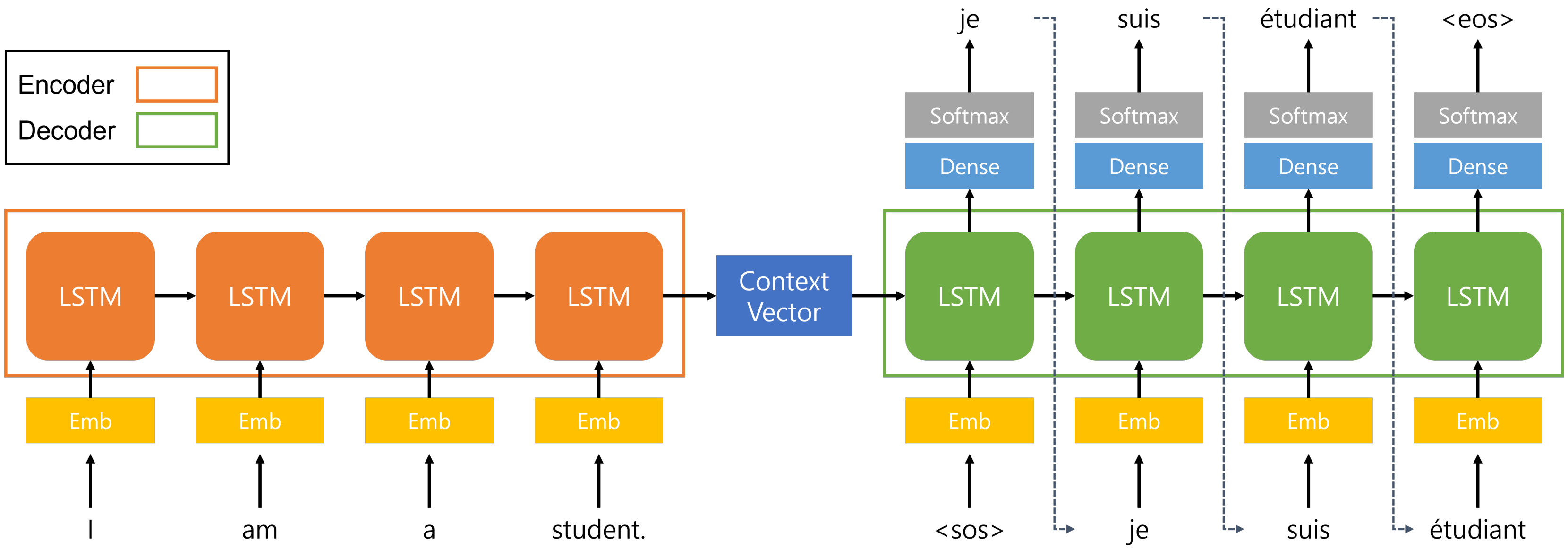
1. Seq2Seq

Many-to-Many: Machine Translation | Chatbot



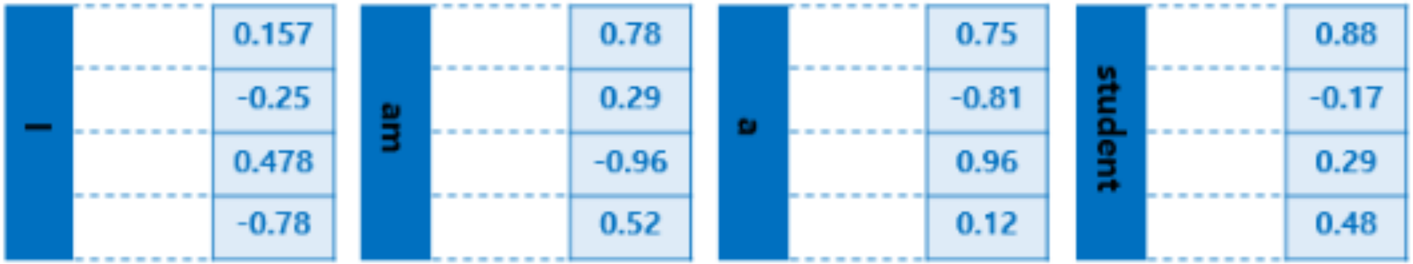
1. Seq2Seq

Many-to-Many: Machine Translation | Chatbot



1. Seq2Seq

Word Embedding: More powerful Strategy



1. Seq2Seq

1) Tokenize

"KUBIG is very good!"
-> "KUBIG" "is" "very" "good" "!"

2) Punctuation

-> "KUBIG" "is" "very" "good"
-> Removing is always not a good method!

: It's not enough to make word embeddings!

1. Seq2Seq

NLTK

1) Tokenize

```
from nltk.tokenize import word_tokenize
from nltk.tokenize import WordPunctTokenizer
```

```
['Don', "'", 't', 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', ',', 'Mr', '.', 'Jone',
"'", 's', 'Orphanage', 'is', 'as', 'cheery', 'as', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop',
'.']
```

2) Punctuation

```
from nltk.tokenize import RegexpTokenizer
```

```
['Think', 'and', 'analyze', 'analyze', 'and', 'think', 'The', 'loop']
```

1. Seq2Seq

How about this situation?

"Don't use that spoon"

1) Do / n't / use / that / spoon

2) Don' / t / use / that / spoon

1. Seq2Seq

How about this situation?

"Don't use that spoon"

1) Do / n't / use / that / spoon -> Ordinary Usage

2) Don' / t / use / that / spoon

1. Seq2Seq

Word Preprocessing

1) Case conversion
   -> Make all words into small cases
2) Cleansing noise data
   - infrequent words
   - short words('a', 'an', 'is' etc)

1. Seq2Seq

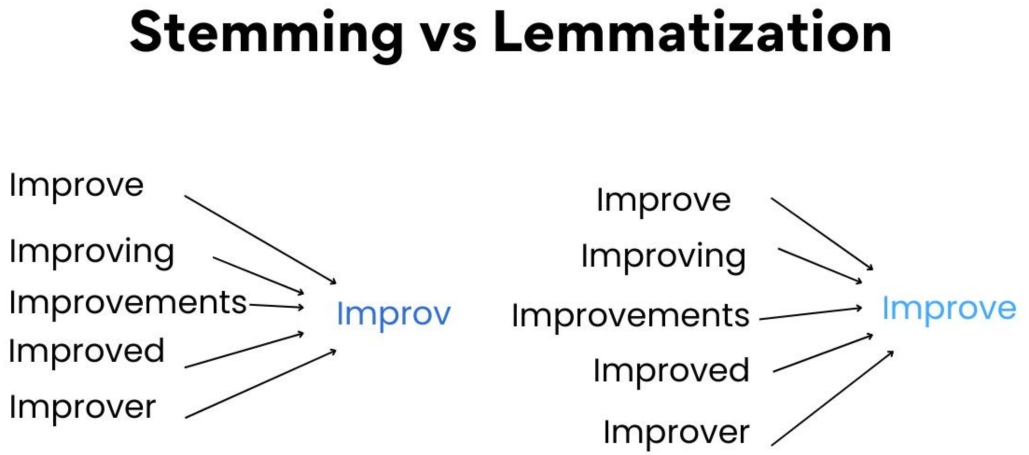
Word Preprocessing

1) Stemming

: remove few characters

2) Lemmatization

: conversion into meaningful base

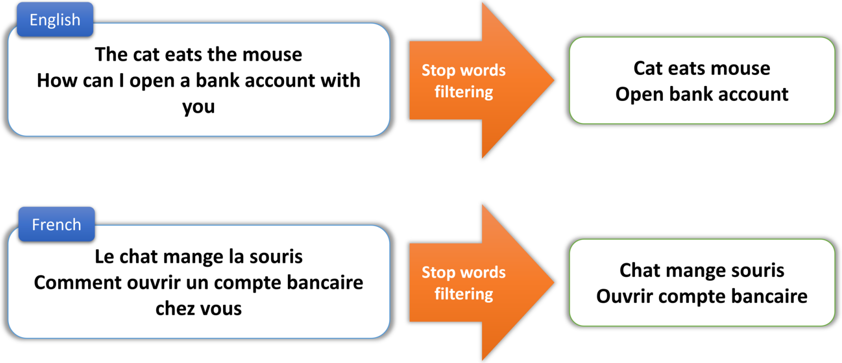


Word Preprocessing

3) Stopwords

: Actually, 'me, I, us' are not meaningful when we analyze the real natural languages

: Other unique words are more attentive!



1. Seq2Seq

Word Preprocessing

NLTK

1) Stemming

```
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

stemmer = PorterStemmer()
```

2) Lemmatization

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
```

3) Stopwords

```
from nltk.corpus import stopwords
```

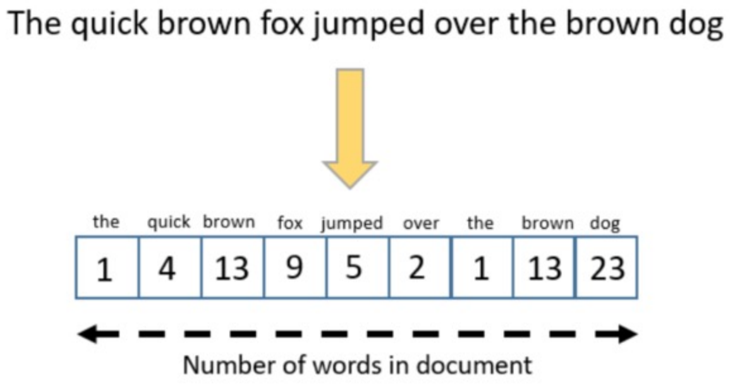
1. Seq2Seq

Word Preprocessing

=> Integer Encoding

: Computer couldn't understand our real language -> We have to convert them into number!

1) Index Encoding
2) One-hot Encoding



1. Seq2Seq

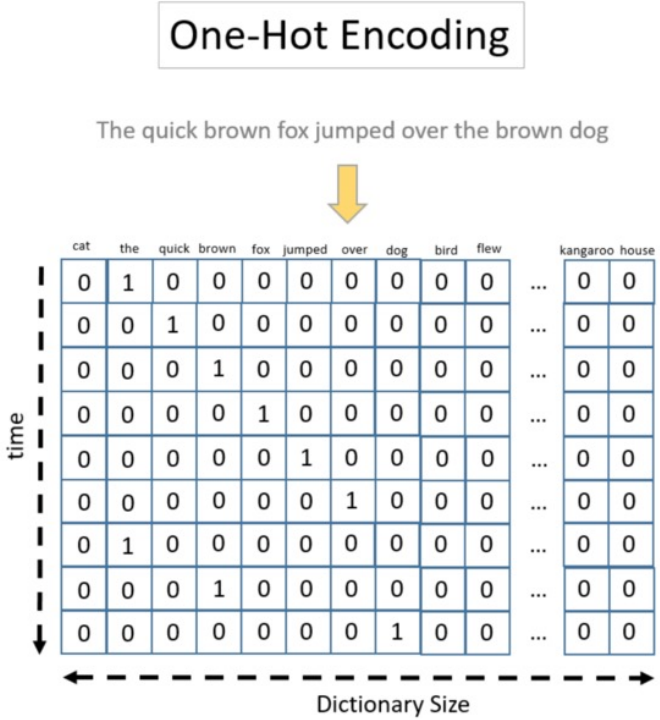
Word Preprocessing

=> Integer Encodir

: Computer couldn't understar ... ıem into number!

1) Index Encoding
2) One-hot Encoding

-> Simple Python function
could make Embeddings!



1. Seq2Seq

Word Preprocessing

+) Padding

: Fitting the dimension is required for parallel operation

: zero-pad for short sentences

```
for sentence in encoded:
    while len(sentence) < max_len:
        sentence.append(0)

padded_np = np.array(encoded)
padded_np
```

1. Seq2Seq

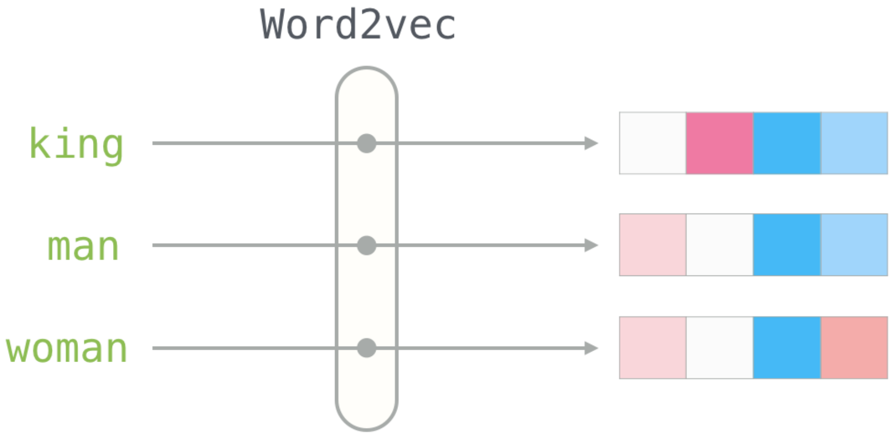
Word Preprocessing

+) Word2Vec

: Library for making Natural words to Vectors

: Training is necessary but powerful

- Feature
  : Distributed Expression ←→ sparse
  : Skip-gram
  : CBOW



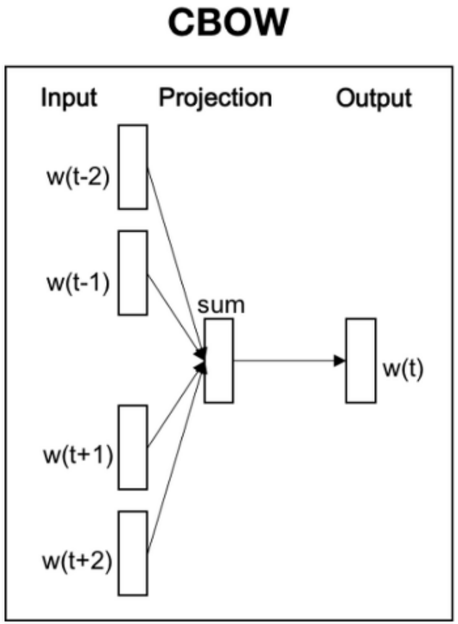
1. Seq2Seq

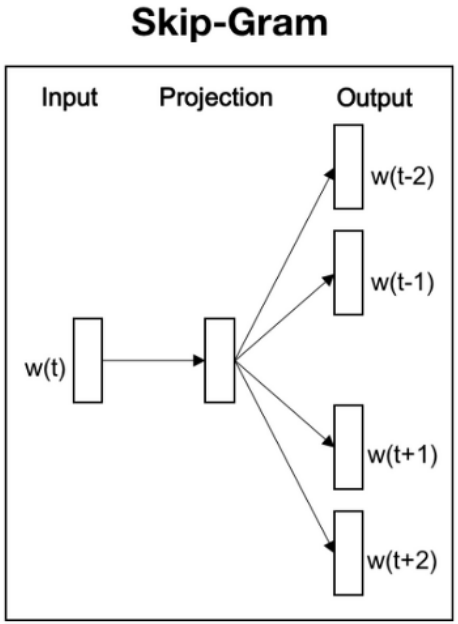
Word Preprocessing

+) Word2Vec

: Library for making Natural words to Vec
: Training is necessary but powerful

- Feature
  : Distributed Expression ←→ sparse
  : Skip-gram
  : CBOW





1. Seq2Seq

```python
from gensim.models import Word2Vec
model = Word2Vec(split, size=100, window=5, min_count=20, workers=4,
iter=50, sg=1)
```

Word Preprocessing

+) Word2Vec

: Library for making Natural words to Vec

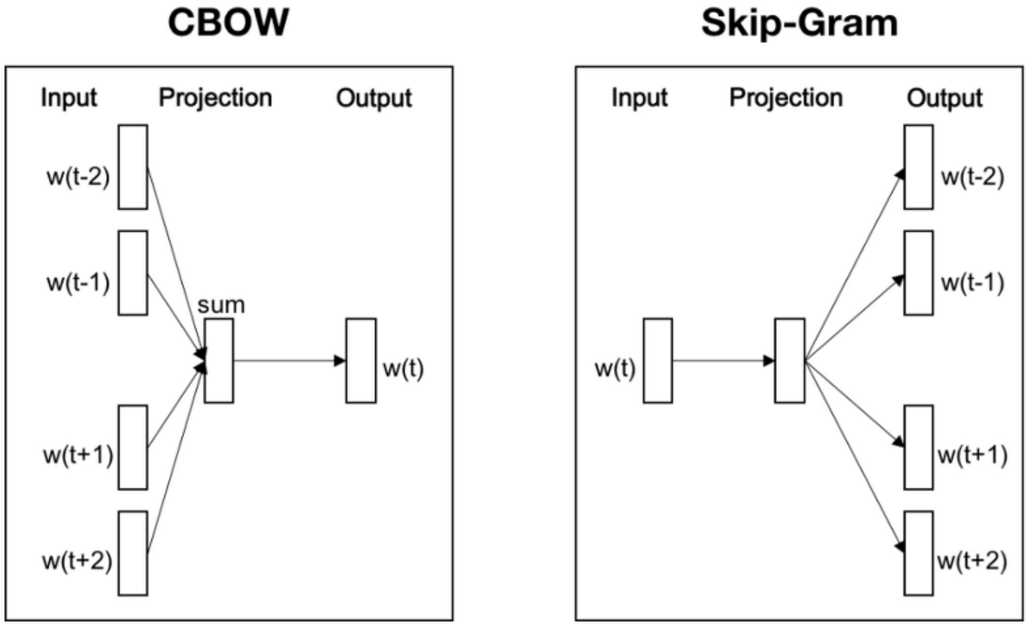
: Training is necessary but powerful

- Feature

: Distributed Expression ←→ sparse

: Skip-gram

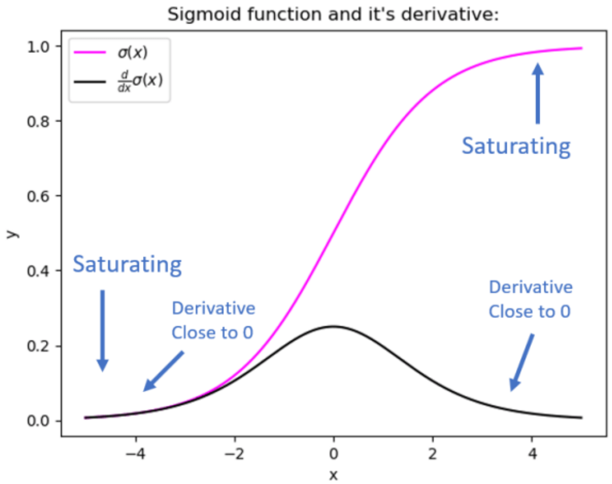
: CBOW



Seq2Seq Limitation

- Trying to compress and summarize all the information in the input sequence into one fixed-sized vector (context vector), inevitably leads to loss of information

- Gradient vanishing/exposing phenomena inevitably occur



2. Attention

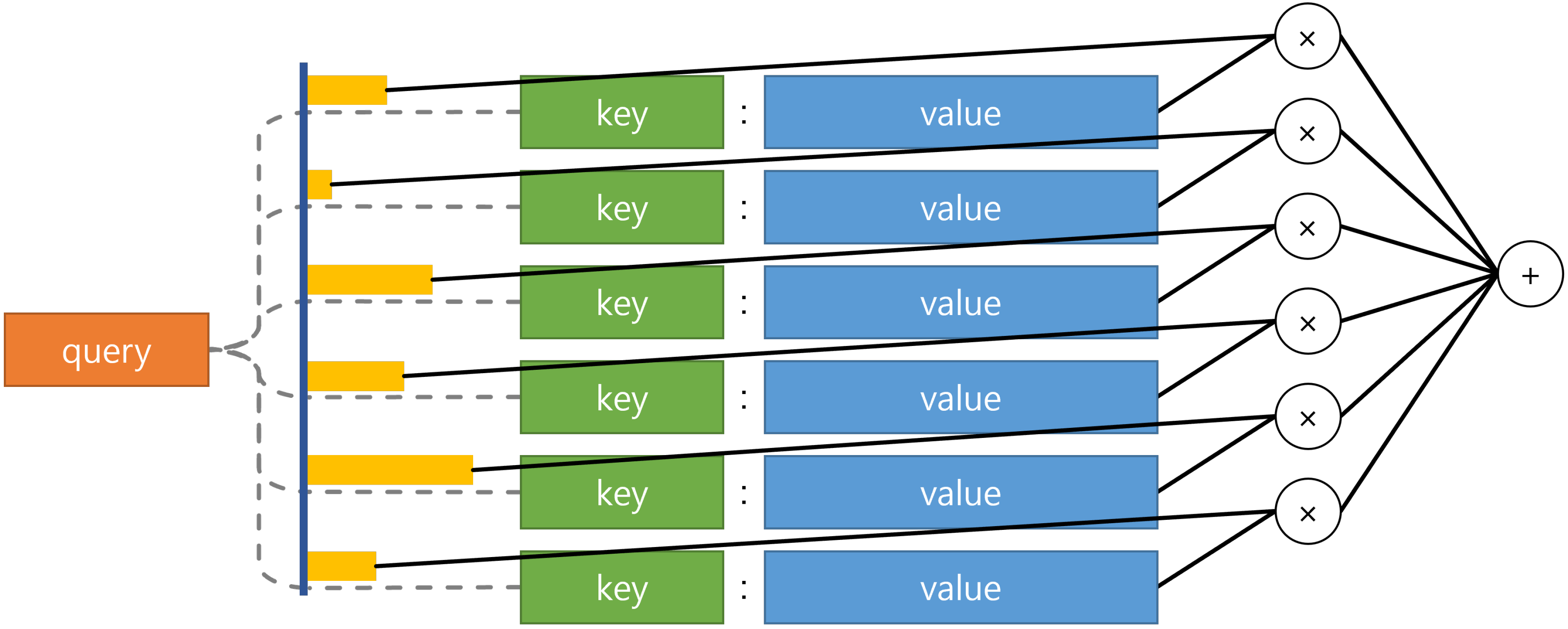
Pre-Assumption

-> The decoder hidden state immediately before the decoder outputs the word will be like the encoder hidden state immediately after the encoder reads the word deeply related to in the input sequence

*Use all hidden state onto each decoder layer!

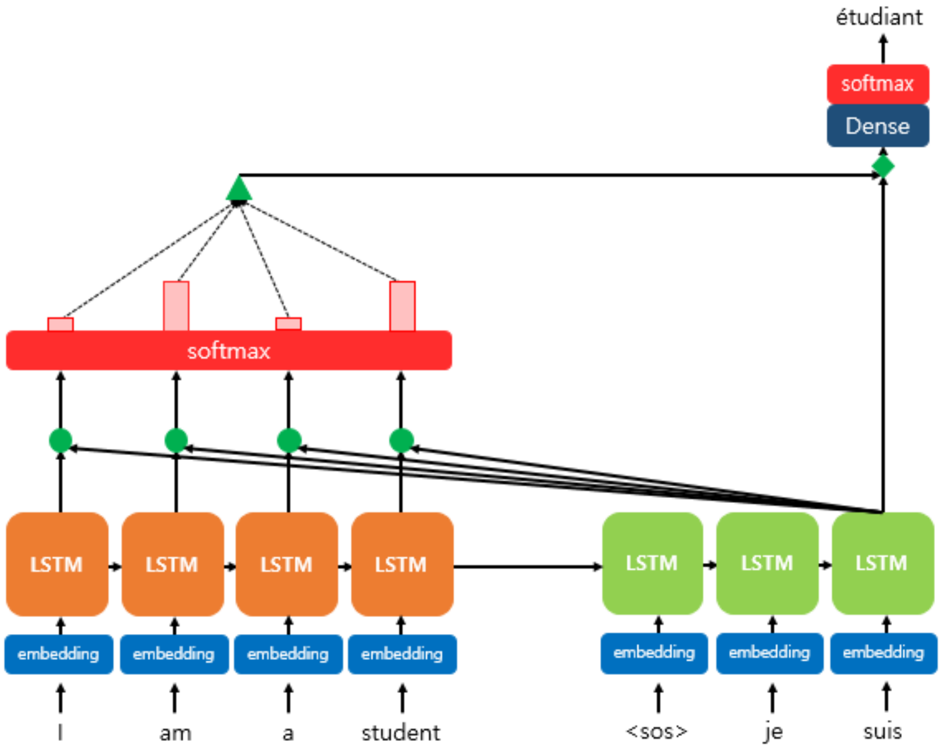
2. Attention

Attention Mechanism



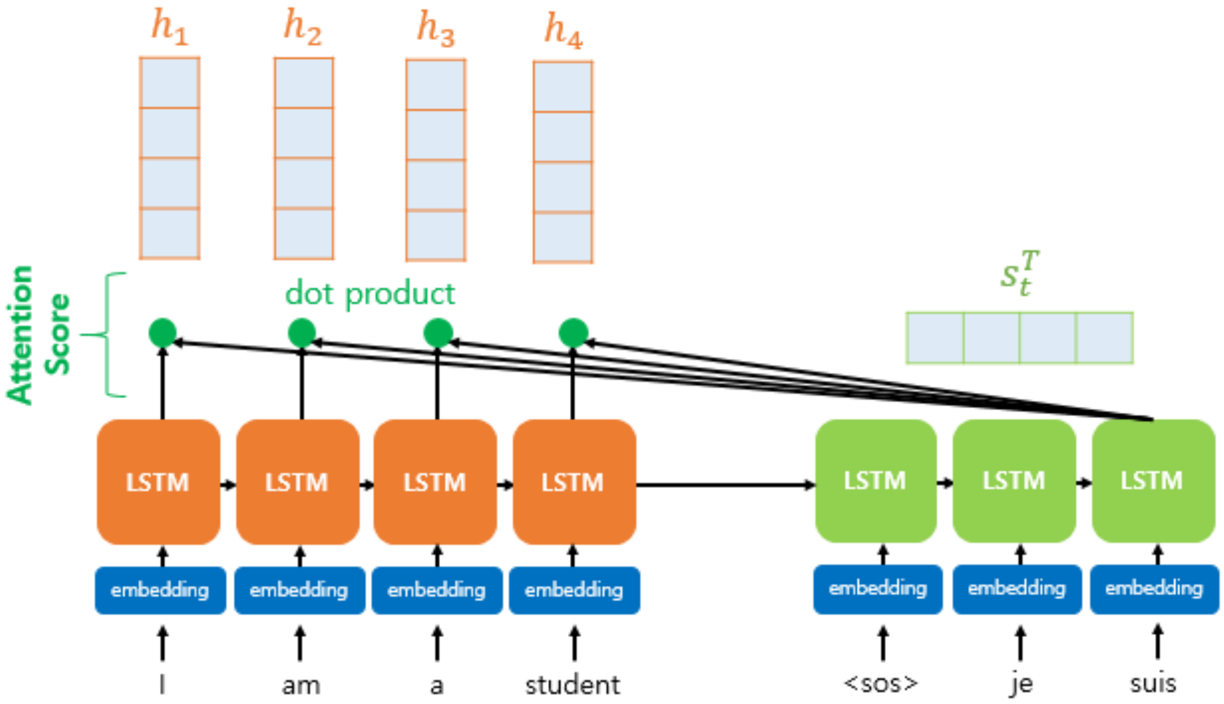
2. Attention

Attention Mechanism



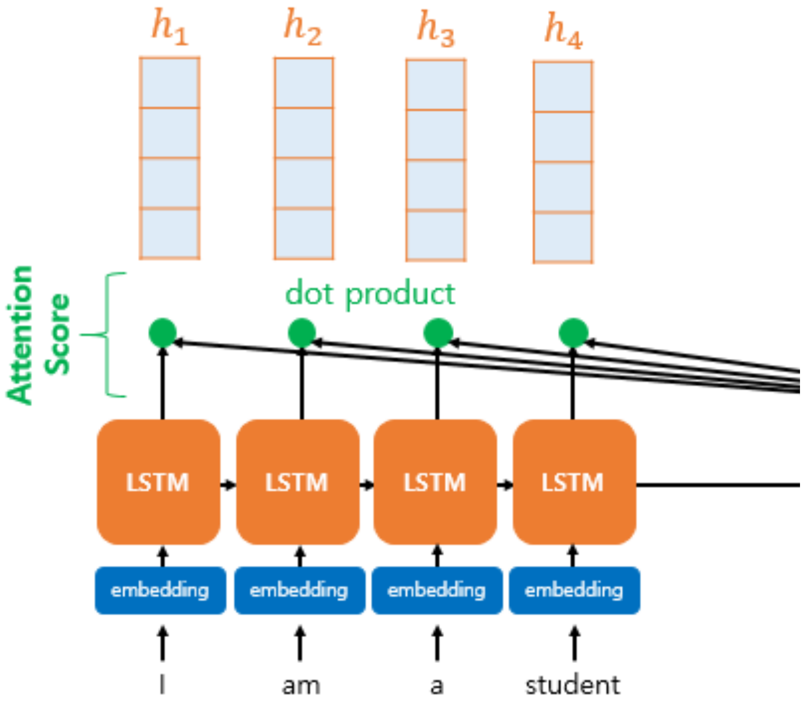
2. Attention

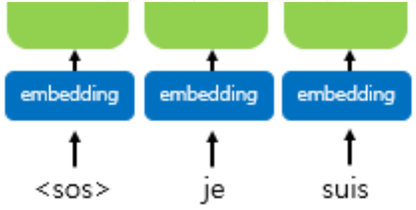
Attention Mechanism



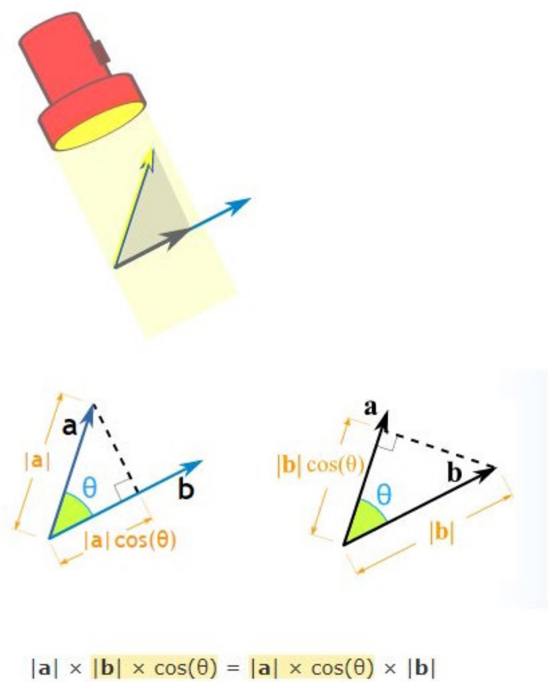
2. Attention

Attention Mechanism



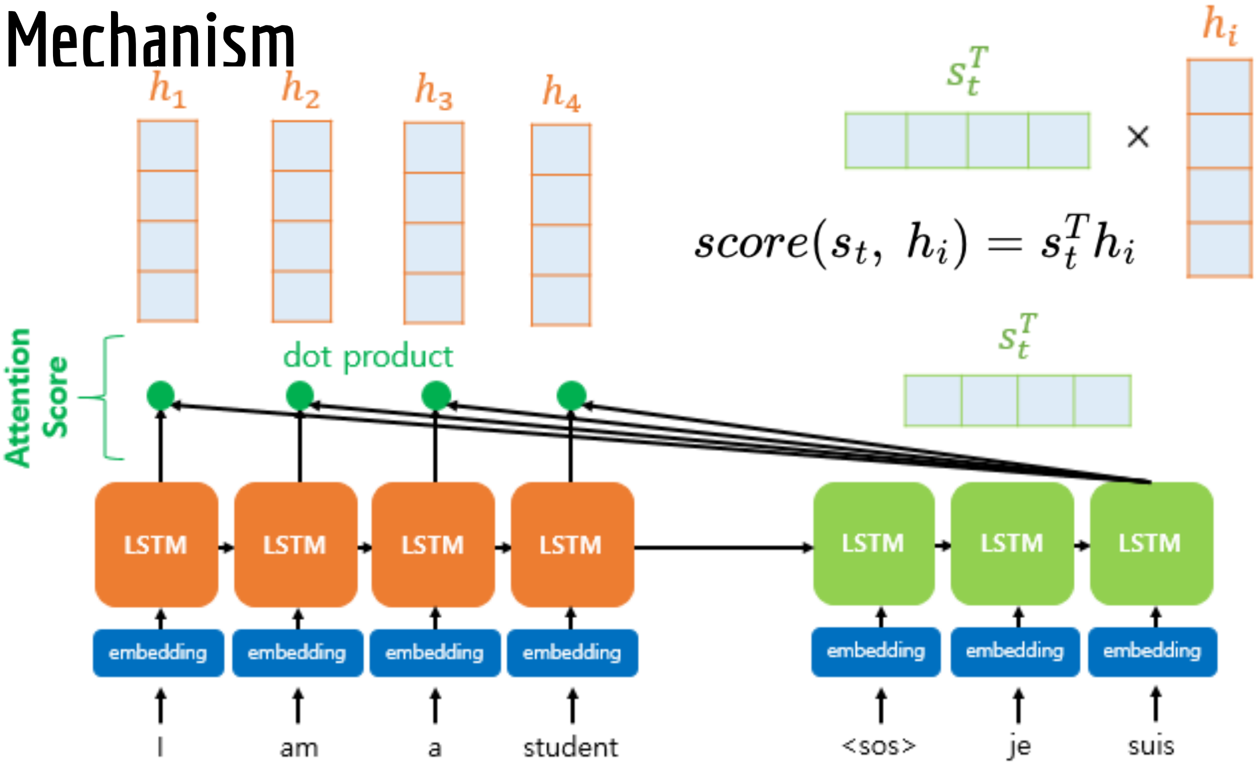


Physics of dot product



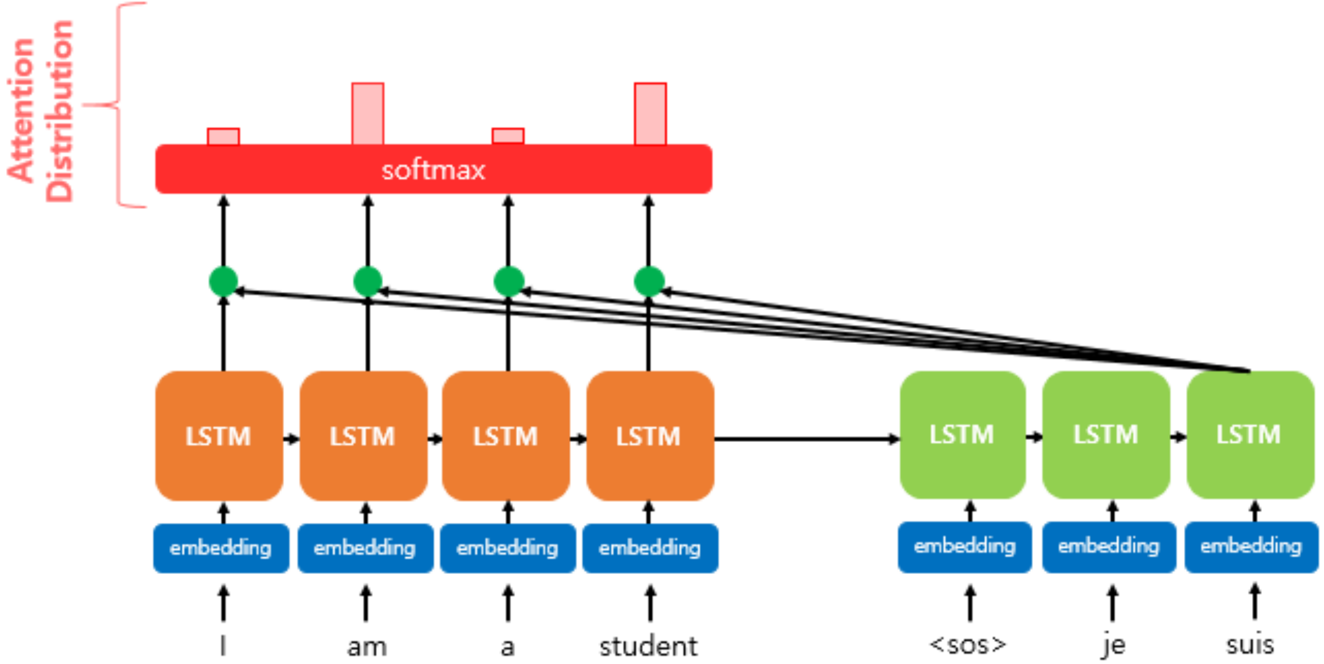
2. Attention

Attention Mechanism



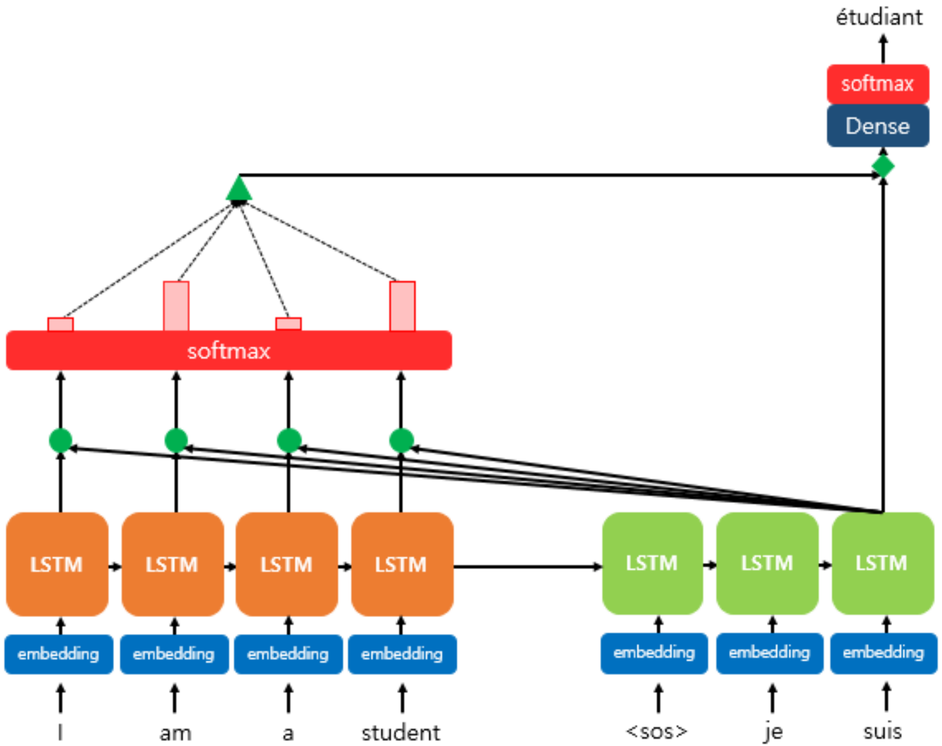
2. Attention

Attention Mechanism



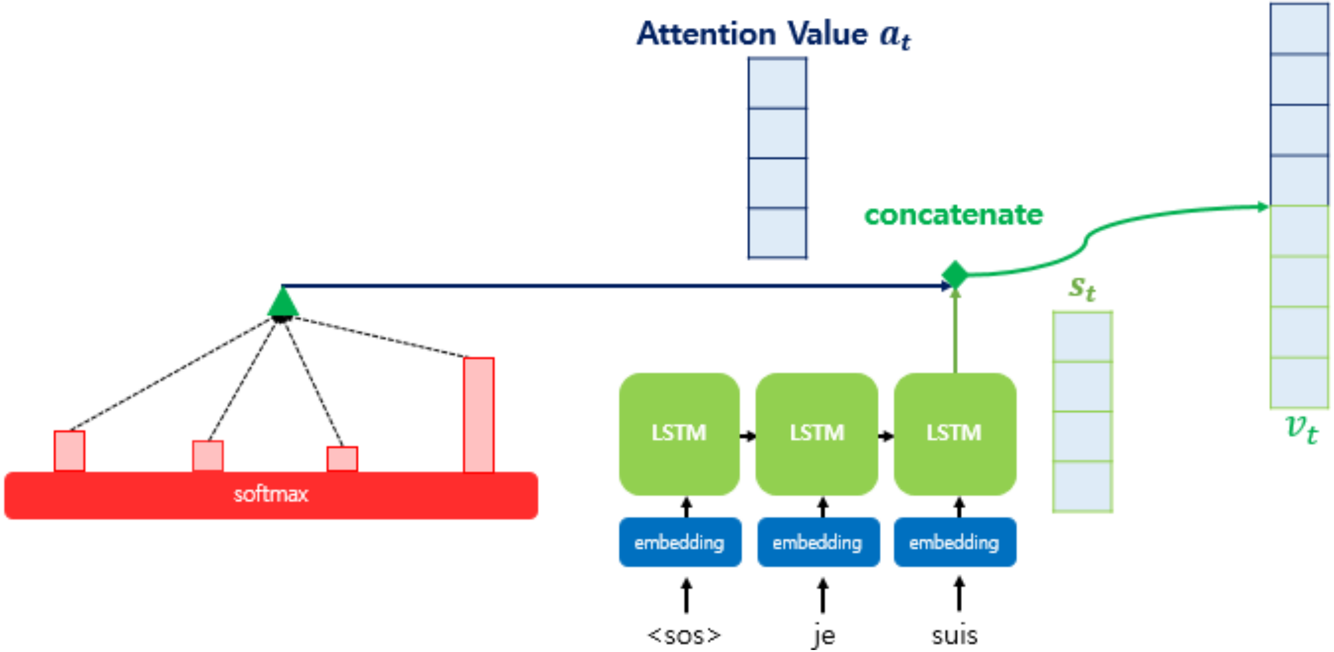
2. Attention

Attention Mechanism



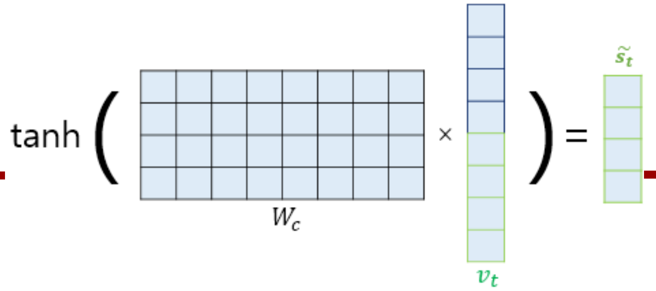
2. Attention

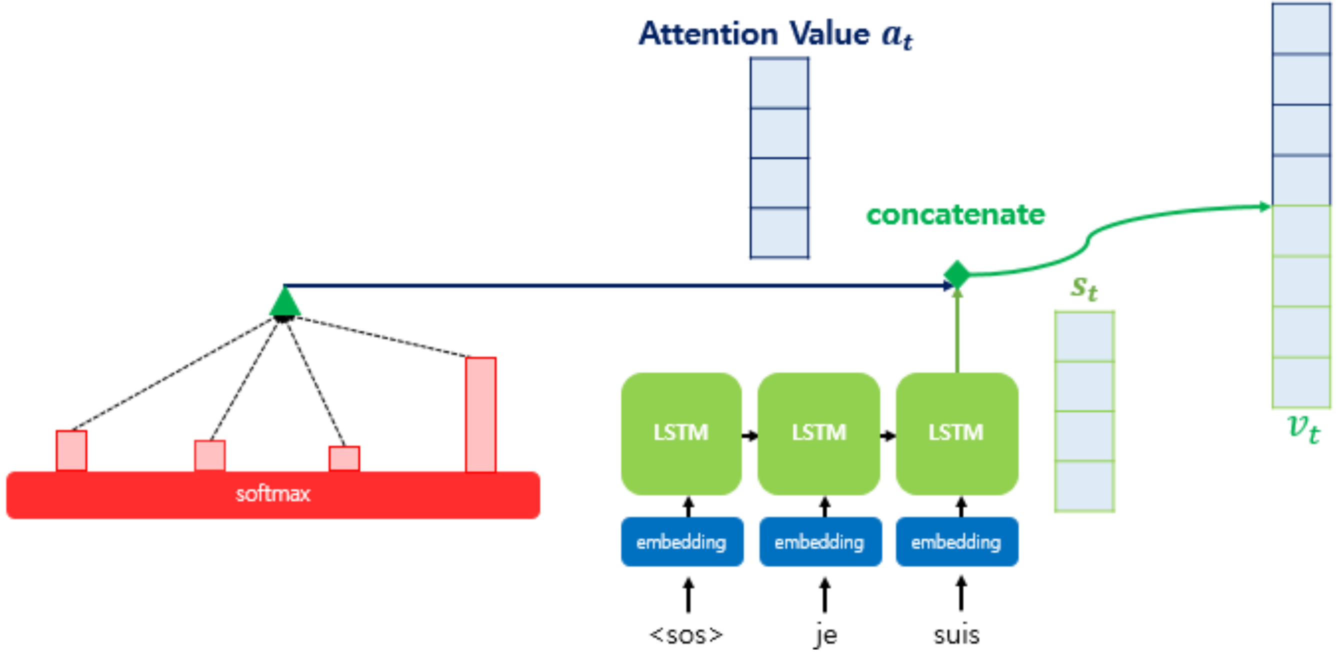
Attention Mechanism



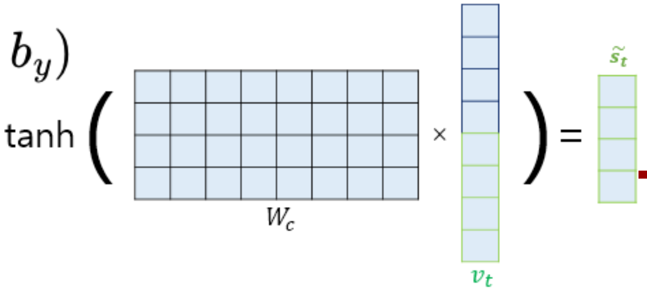
2. Attention

Attention Mechanism



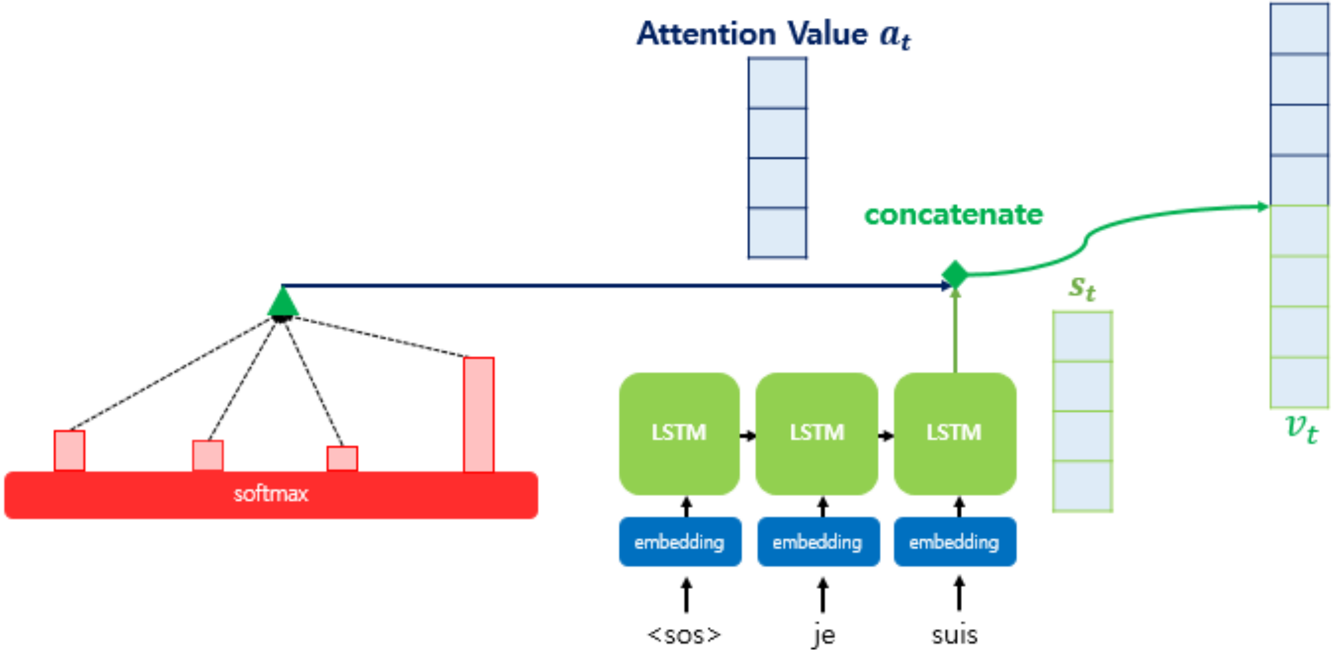


$$\hat{y}_t = \text{Softmax}\left(W_y \tilde{s}_t + b_y\right)$$



2. Attention

Attention Mechanism



2. Attention

Attention Mechanism

이름	식	출처
content-based attention	$f(s, h) = \frac{s^T h}{\|\|s\|\| \cdot \|\|h\|\|}$	Graves, 2014
additive attention (Bahdanau attention)	$f(s, h) = V^T \tanh(W_1 s + W_2 h)$ [4]	Bahdanau, 2015
dot-product attention (Loung attention)	$f(s, h) = s^T h$	Luong, 2015
scaled dot-product attention	$f(s, h) = \frac{s^T h}{\sqrt{n}}$ [5]	Vaswani, 2017

2. Attention

Attention Mechanism

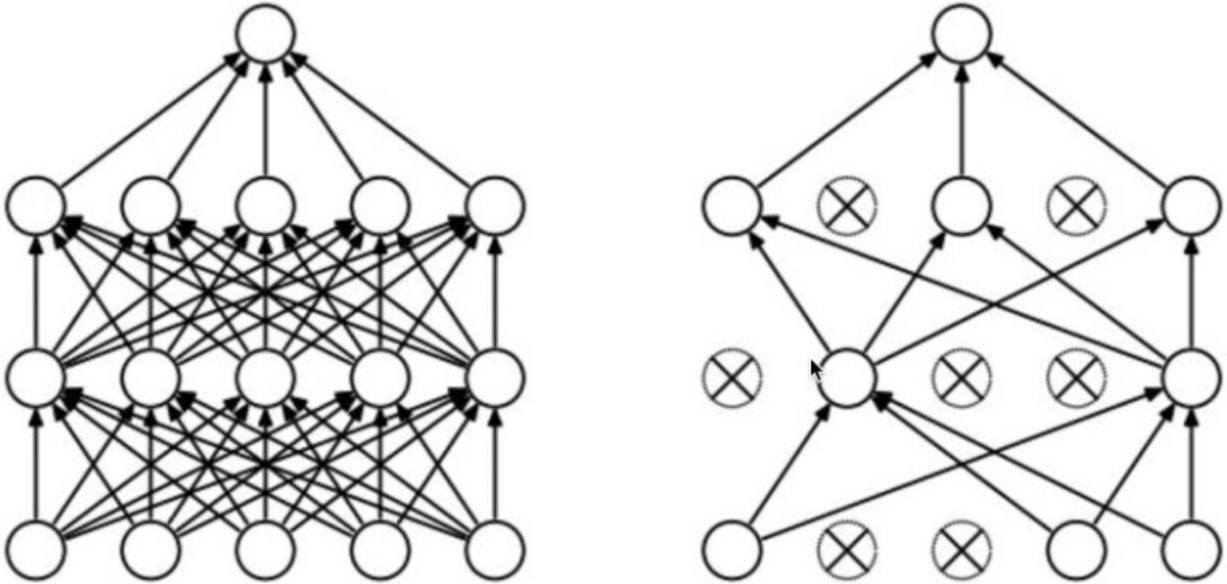
But still,,,

1) Gradient Vanishing / Exploding
2) Too slow (non-parallel operation)

=> Transformer

3. Training Techniques

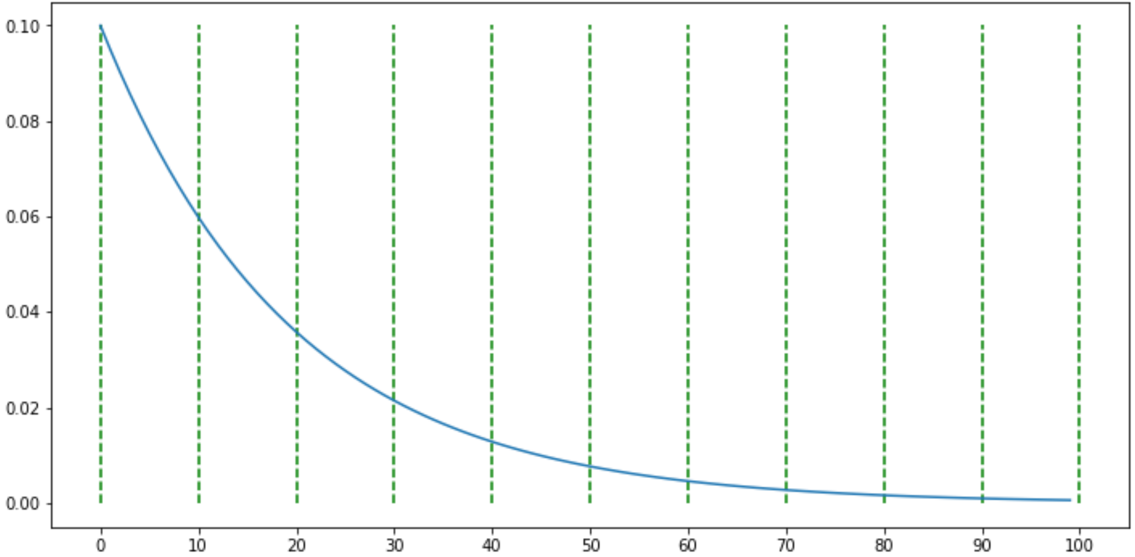
Dropout



(a) Standard Neural Net (b) After applying dropout.

LR Scheduler: Lambda LR Scheduler

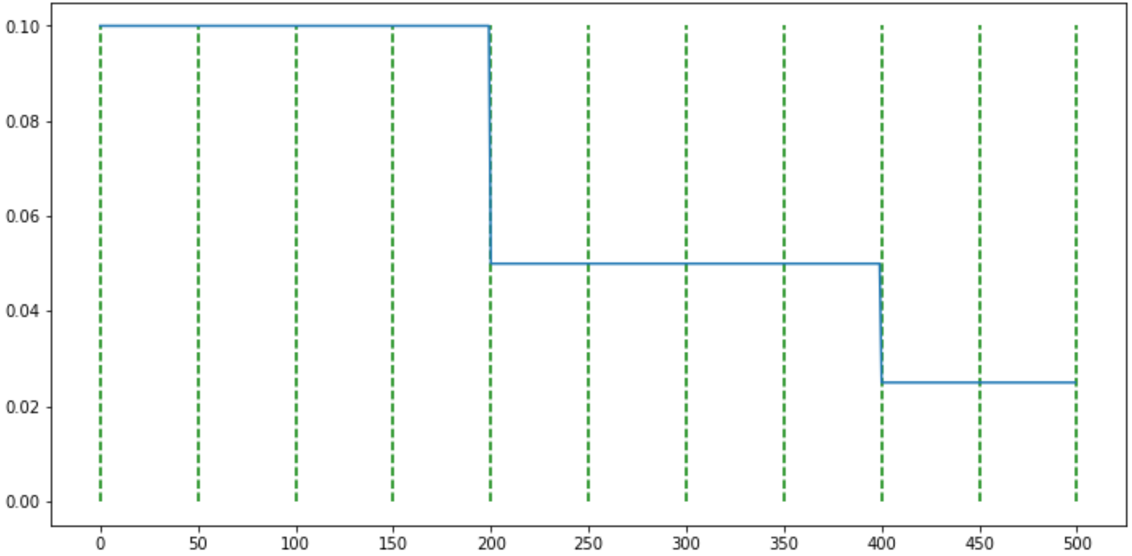
```
scheduler = LambdaLR(optimizer, lr_lambda = lambda epoch: 0.95 ** epoch)
```



PyTorch

Step LR Scheduler

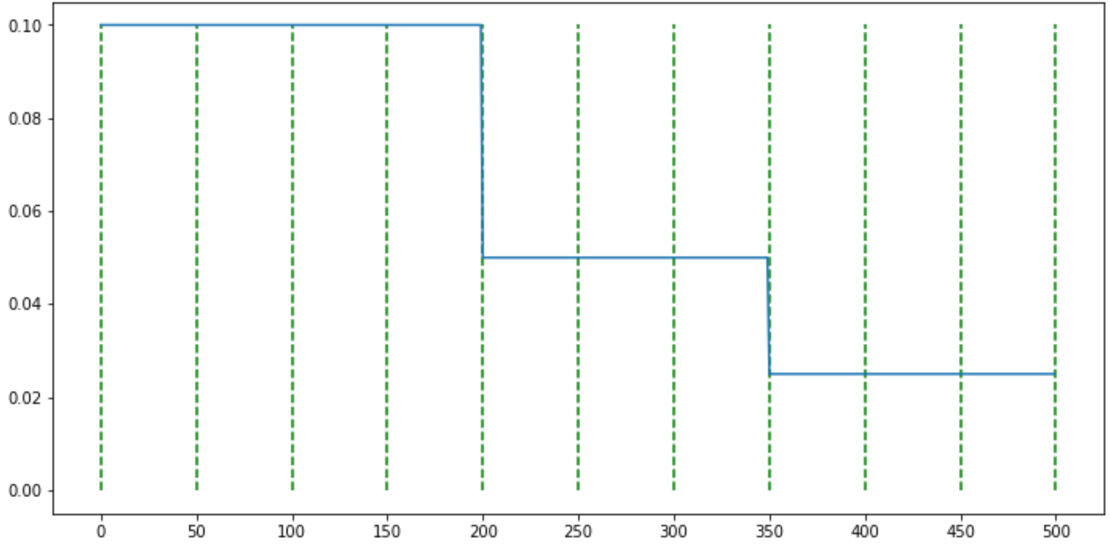
```
scheduler = StepLR(optimizer, step_size=200, gamma=0.5)
```



3. Training Techniques

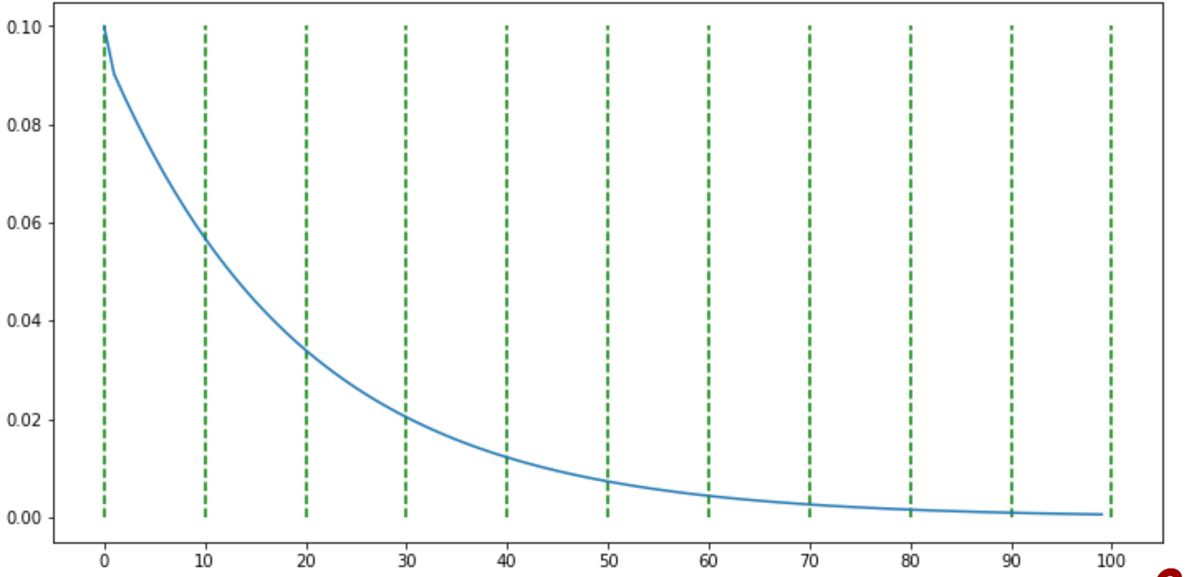
Multi-Step LR Scheduler

```
scheduler = MultiStepLR(optimizer, milestones=[200, 350], gamma=0.5)
```



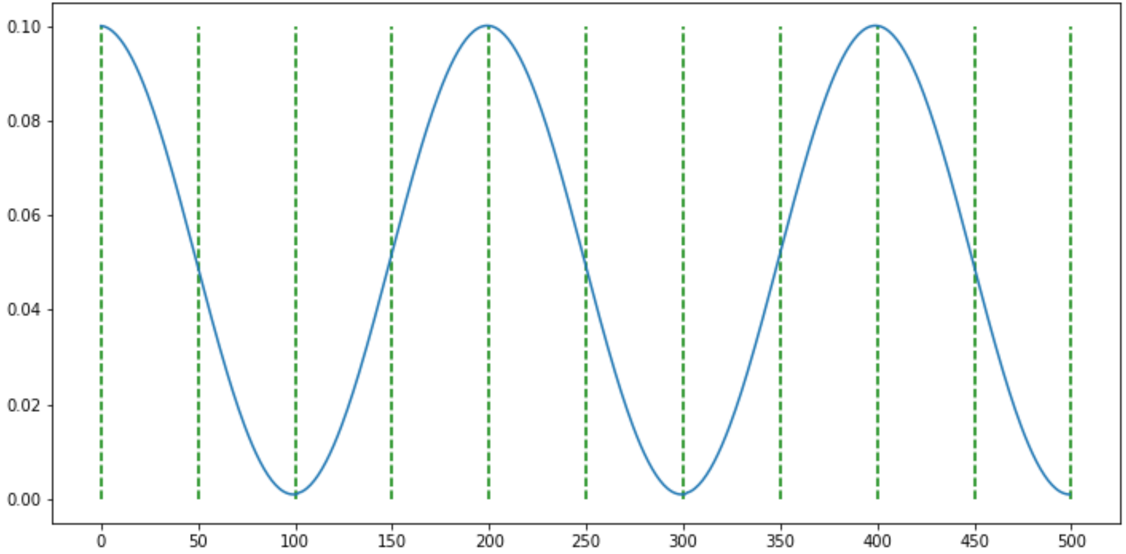
Exponential LR Scheduler

```
scheduler = ExponentialLR(optimizer, gamma=0.95)
```



Cosine Annealing LR Scheduler

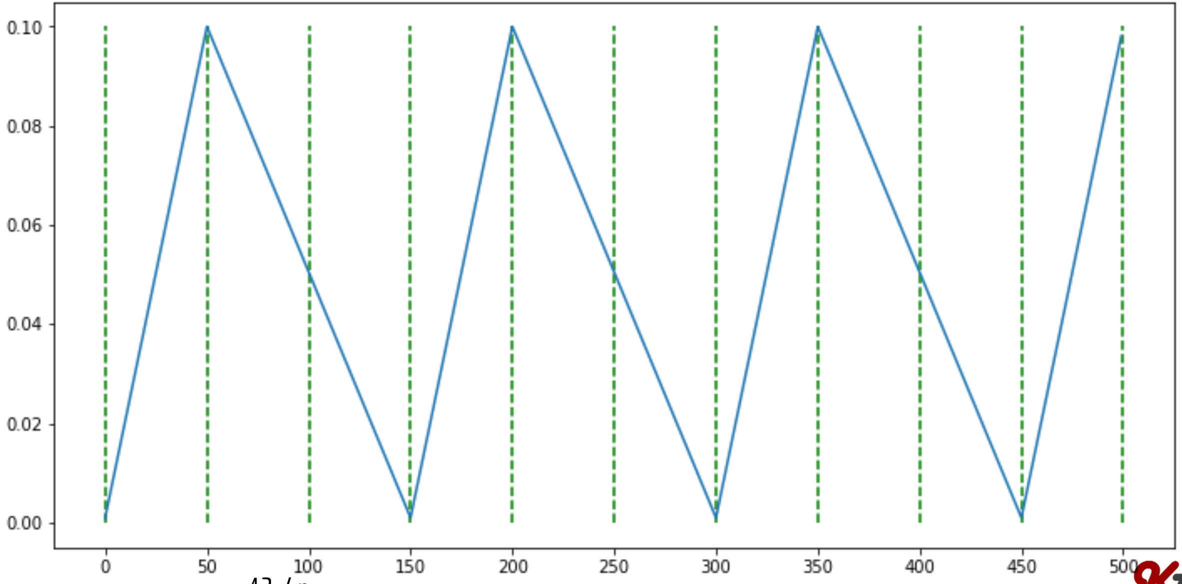
```
scheduler = CosineAnnealingLR(optimizer, T_max=100, eta_min=0.001)
```



3. Training Techniques

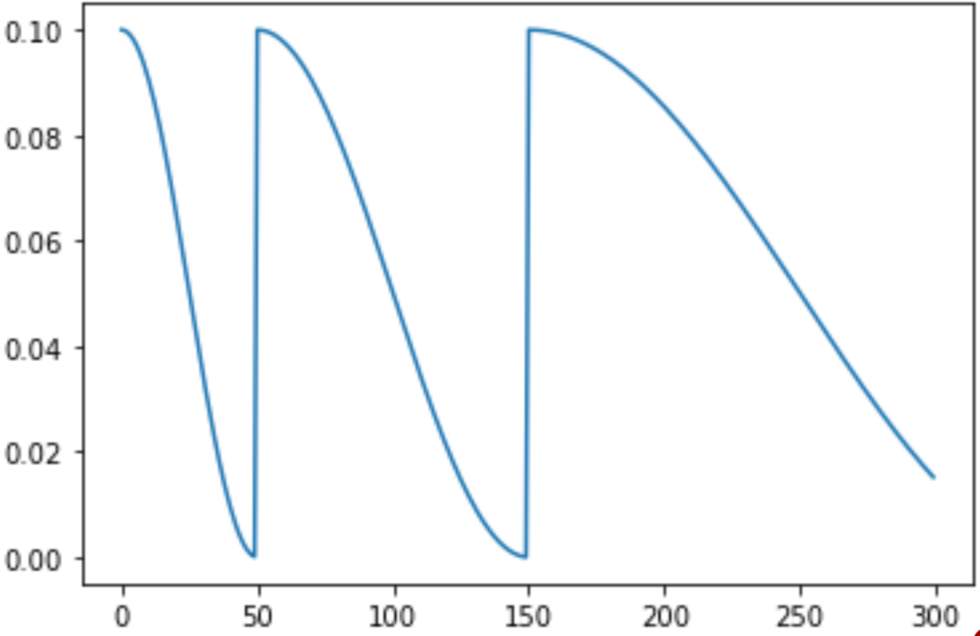
Cycle LR Scheduler

```
scheduler = CyclicLR(optimizer, base_lr=0.001, max_lr=0.1, step_size_up=50, step_size_down=100, mode='triangular')
```



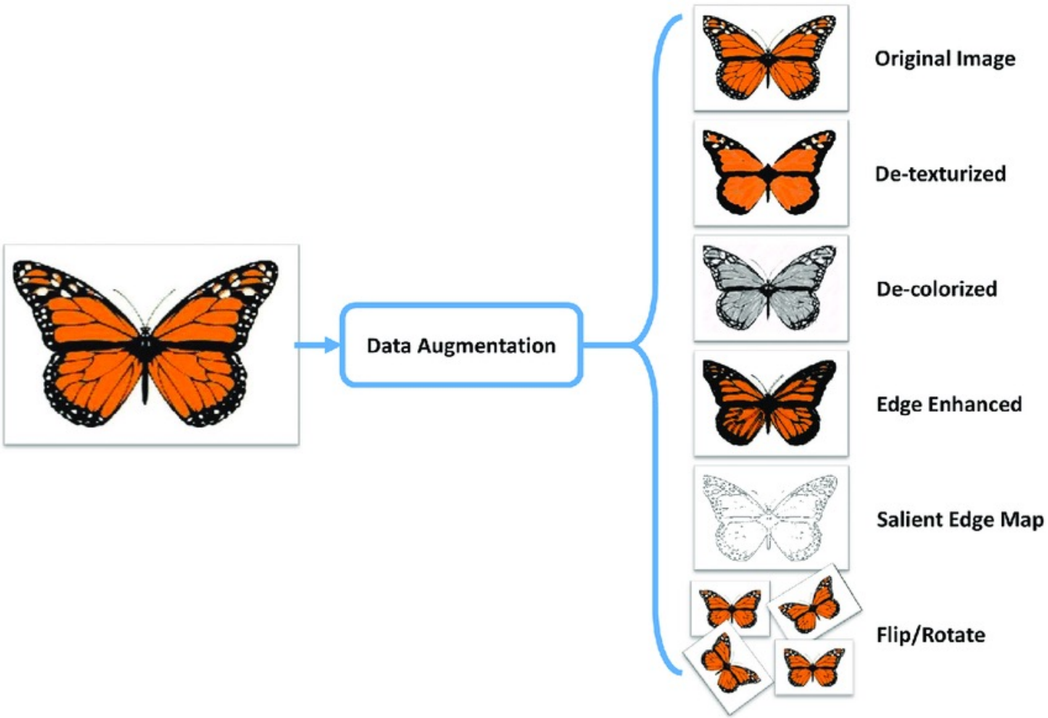
Cosine Annealing LR Scheduler with Warm Restarts

```
scheduler = optim.lr_scheduler.CosineAnnealingWarmRestarts(optimizer, T_0=50, T_mult=2, eta_min=0)
```



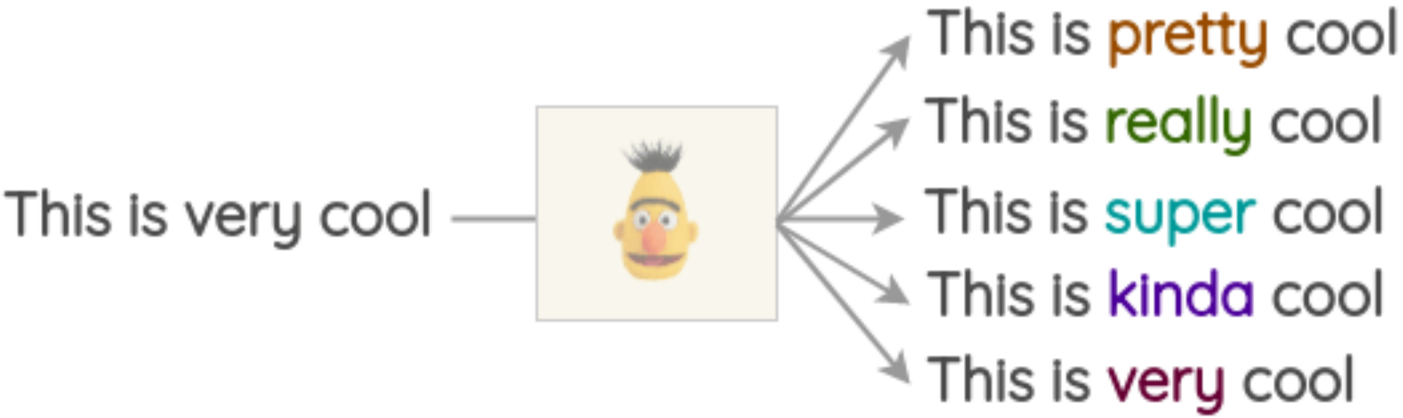
3. Training Techniques

Data Augmentation



3. Training Techniques

Data Augmentation



Does it seem efficient?

Q&A

Have a nice week🥰