Introduction to POS Tagging

Outline

- Problem Introduction
- Simple Examples
- Code Implementation

Problem Introduction

❖ Definition

Part of Speech (POS) tagging: assign each word in a sentence with an appropriate POS

- **Input**: a string of words + a tagset
- **Output**: a best tag for each word

Words often have more than one POS:

- The back door (adjective)
- On my back (noun)
- Win the voters back (particle)
- Promised to back the bill (verb)

➢ Due to ambiguity (and unknown words), we cannot rely on a dictionary to look up the correct POS tags.

Problem Introduction

❖ Why POS tagging?

- POS tagging is one of the first steps in the NLP pipeline (right after tokenization, segmentation).

- POS tagging is traditionally viewed as a prerequisite for further analysis:
  - **Syntactic Parsing:** What words are in the sentence?
  - **Information extraction:** Finding names, dates, relations, …

Application Area	Description
Sentiment Analysis	POS tags help identify crucial sentiment indicators like adjectives.
Named Entity Recognition (NER)	Improves NER by providing context, such as identifying proper nouns as potential named entities.
Automatic Summarization	Identifies key verbs and nouns to extract main ideas for summaries.
Text Generation and Chatbots	Ensures grammatical correctness and enhances the naturalness of generated text.
Grammar Checking and Proofreading	Utilized to identify grammatical errors and suggest corrections.

Problem Introduction

❖ English word classes

Problem Introduction

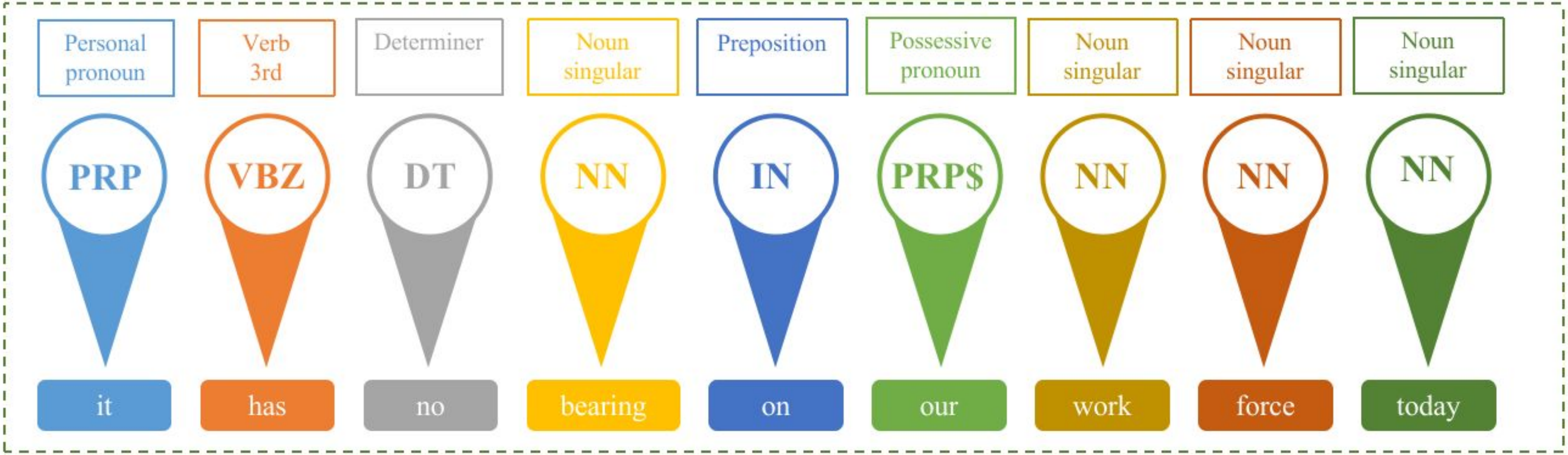
❖ English word classes

Open classes

- Words that belong to open classes are content words.
- Also called big classes.
- Consists of
  - Nouns - child, toy
  - Lexical verbs - write, sing
  - Adjectives - clever, neat
  - Adverbs - softly, rudely

Close classes

- Words that belong to close classes are structural words.
- Grammatical meaning
- Also called small classes.
- Consists of
  - Auxiliary verbs – have, will
  - Pronouns – we, he
  - Prepositions – on, under
  - Conjunctions – although and
  - Determiners – the, some
  - Enumerators – two, third
  - Interjections – alas, oh



Example from Penn Treebank

1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunctio
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	*to*
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

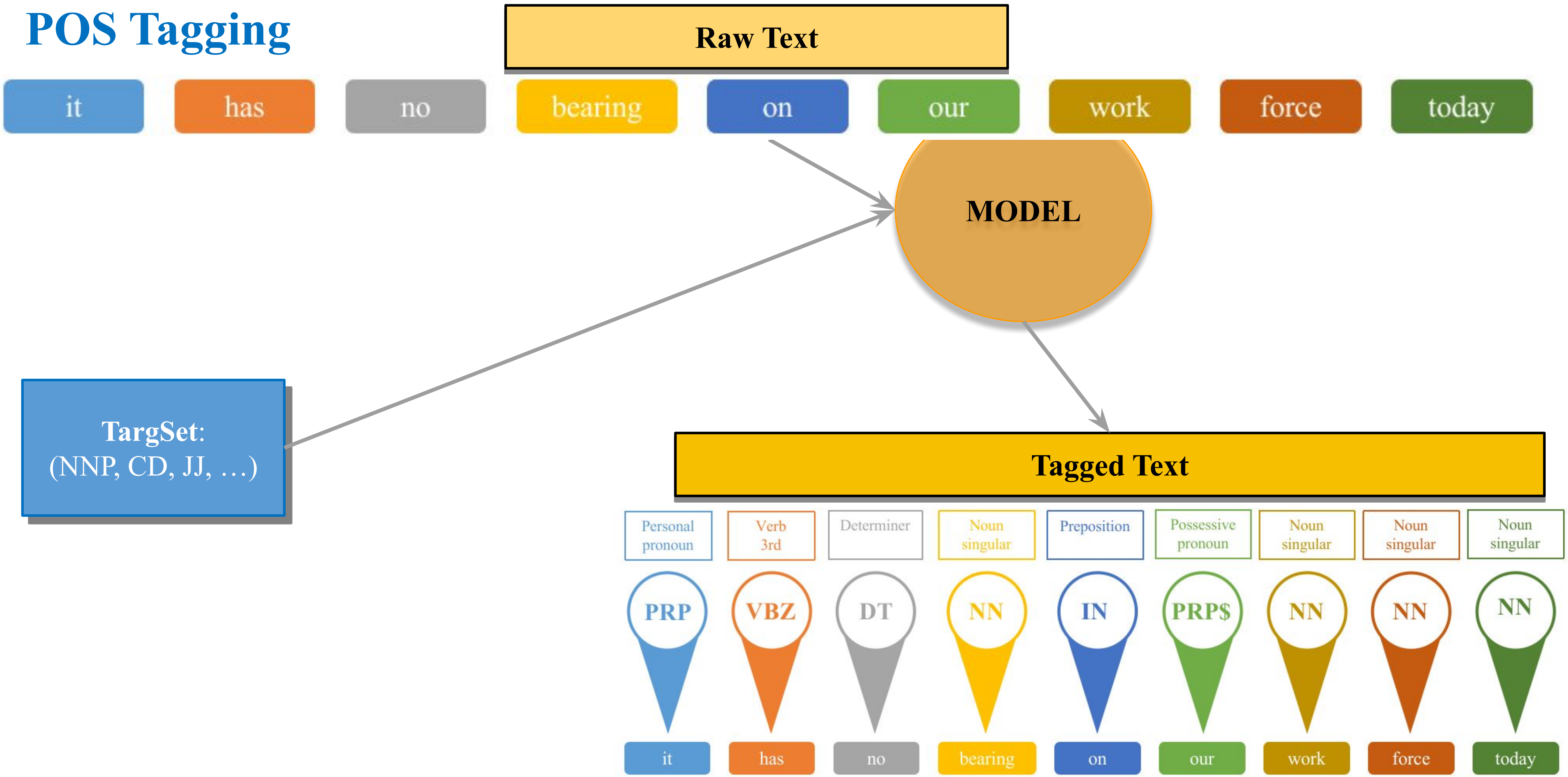
Simple Examples

❖ Creating a POS Tagger

- **Step 1**: Obtain a POS-tagged corpus
- **Step 2**: Choose a POS tagging model
- **Step 3**: Train your model on your training corpus
- **Step 4**: Evaluate your model on your test corpus

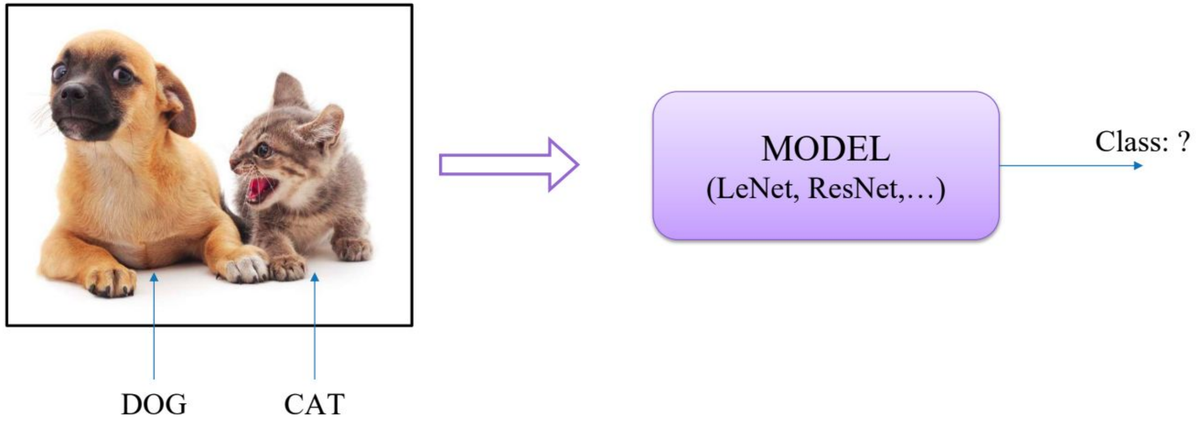
Simple Examples

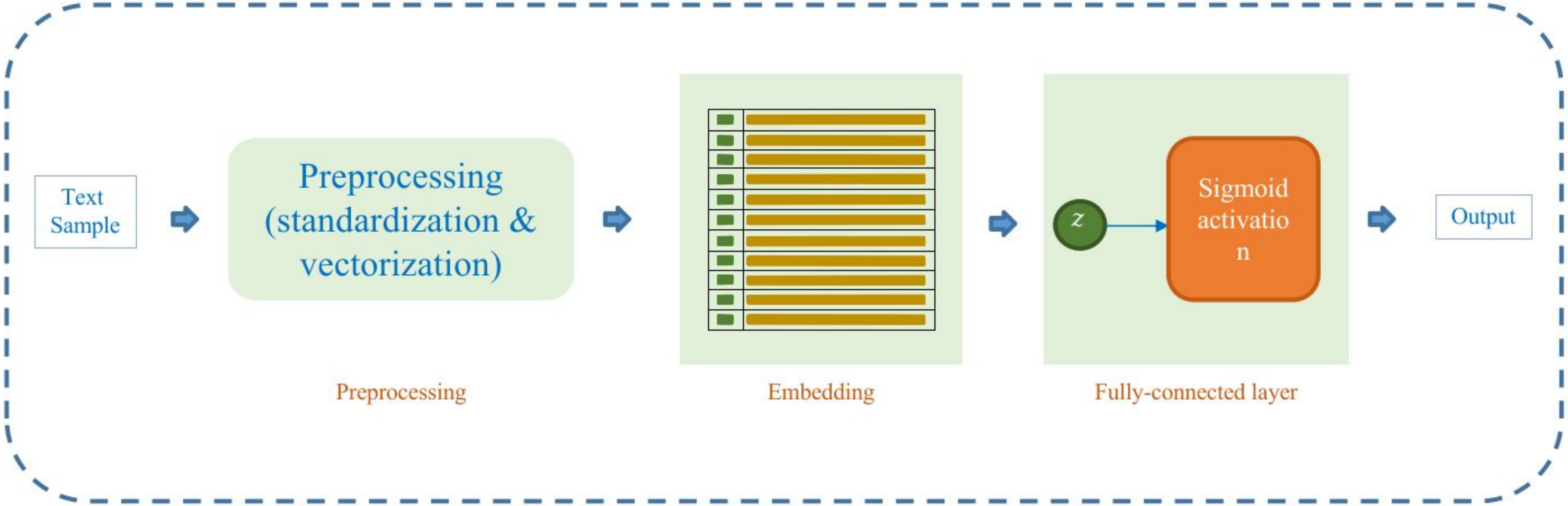
❖ POS Tagging



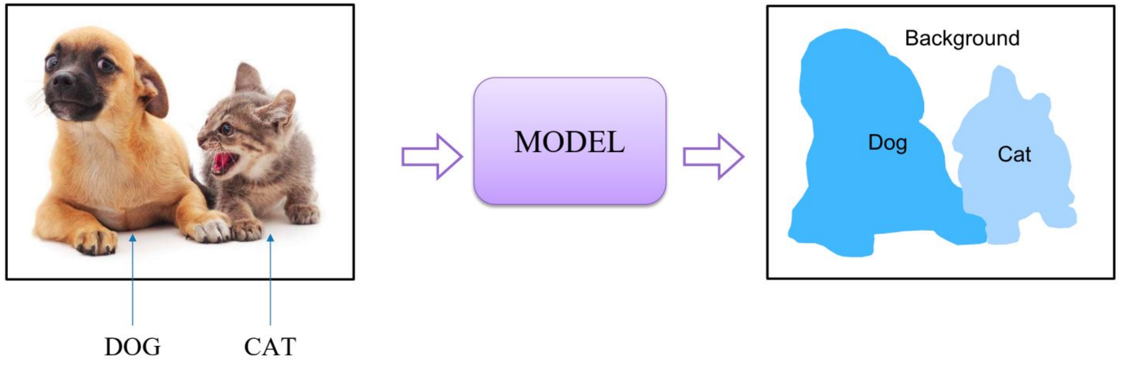
Simple Examples

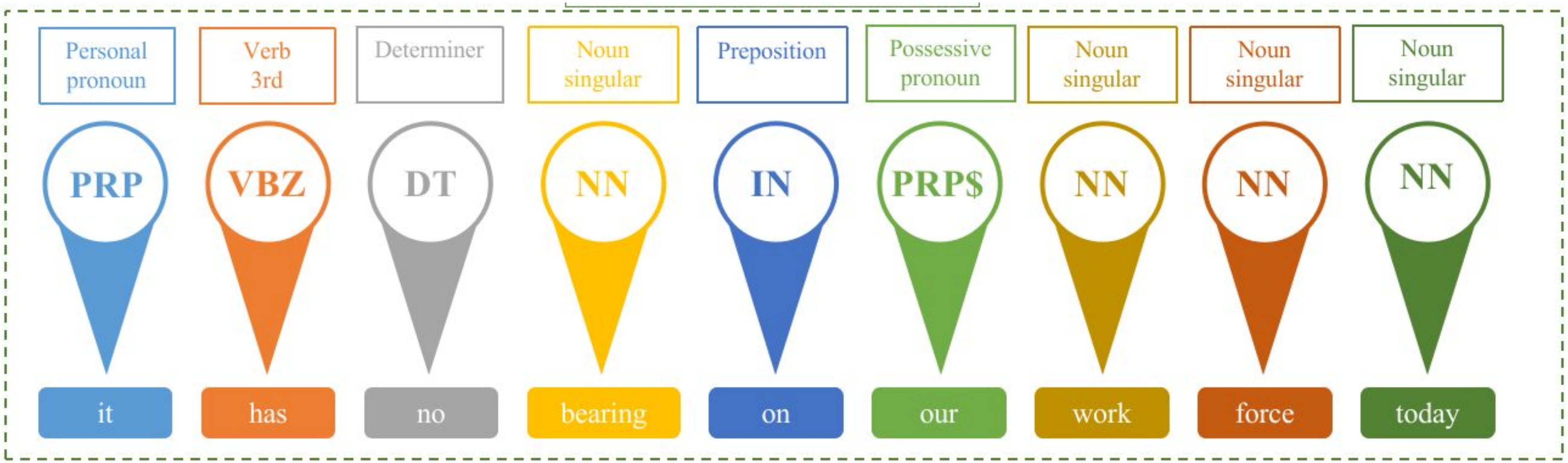
❖ Text Classification



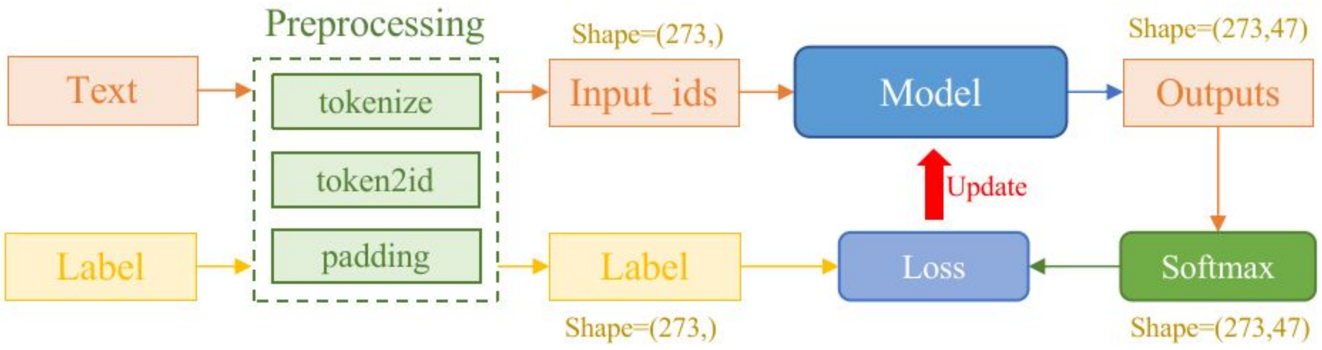


POS
Part-of-speed
Tagging





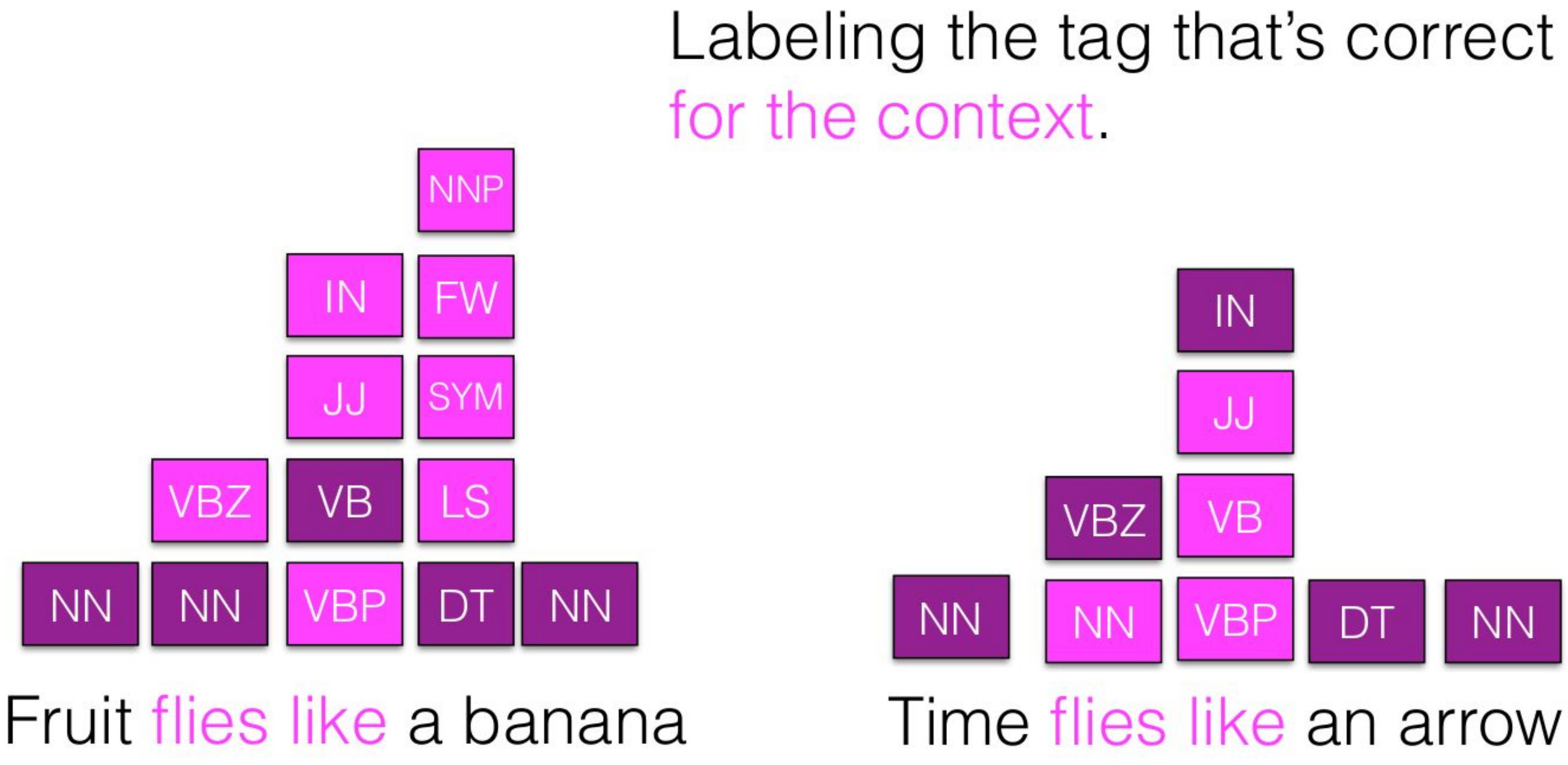
Index	Label
0	<unk>
1	NN
2	IN
3	NNP
…	…
43	LS
44	FW
45	UH
46	SYM



Simple Examples

❖ POS Part-of-speed Tagging

- **Neural sequence models (RNNs or Transformers)**
- **Large Language Models (like BERT), finetuned**

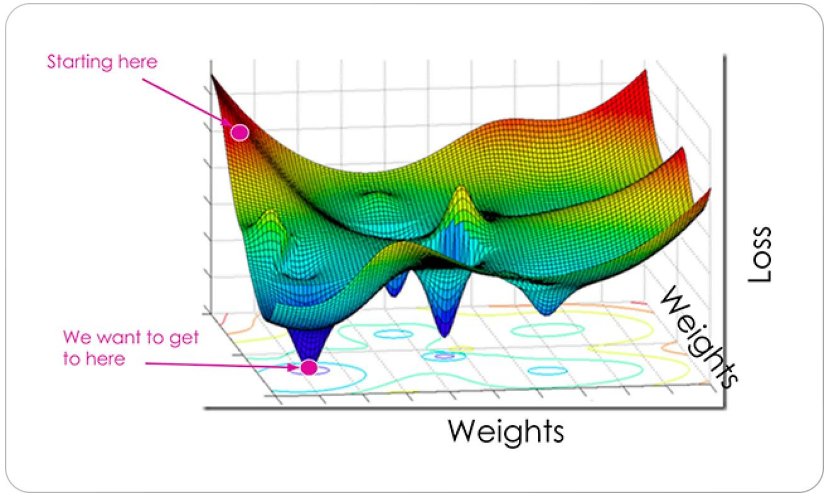


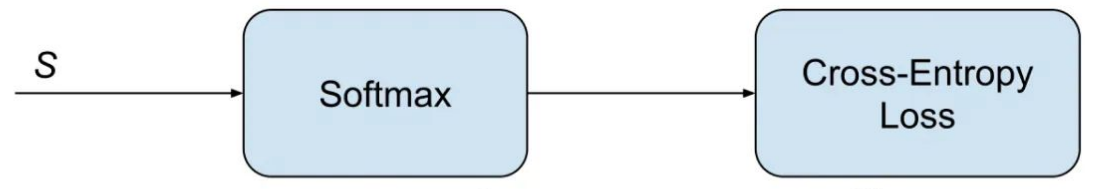
Simple Examples

❖ Evaluation Metric: Test Accuracy

- **How many words in the unseen test data can you tag correctly?**
  ⇨ How many sentences can you tag correctly?

Metric	Description	Importance
Accuracy	Proportion of words correctly tagged	Measures overall effectiveness
Precision	Correctly predicted tags for a specific POS out of all predicted for that POS	Important for understanding model's performance on each tag
Recall	Correctly predicted tags for a specific POS out of all actual instances of that POS	Helps identify if the model is missing any specific POS tags
F1 Score	Weighted average of Precision and Recall	Useful in cases of uneven distribution of POS tags





$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \qquad CE = -\sum_i^C t_i log(f(s)_i)$$

Code Implementation

❖ POS Tagging Code Overview

0: Import Library → 1: Data Preparation → 2: Modeling → 3: Train and Evaluate → 4: Plot Results

1: Data Preparation
1. Load Raw Data
2. Split Data
3. Tokenizer
4. Pad and Truncate

2: Modeling
1. Config Model
2. Build Model

3: Train and Evaluate
1. Train model
2. Evaluate model

4: Plot Results
1. Train/Test Loss
2. Train/Test Acc

Code Implementation

❖ Import Library

`import torch`: Imports PyTorch, a library for tensor computation and deep learning.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

`import torch.nn as nn`: Brings in PyTorch's neural network module, aliased as `nn`, for building network layers.

`import torch.nn.functional as F`: Imports PyTorch's functional interface, providing access to activation and loss functions.

❖ Conference on Computational Natural Language Learning (CoNLL-2003) Dataset

id string · lengths 1 5	tokens sequence	pos_tags sequence	chunk_tags sequence	ner_tags sequence
0	["EU", "rejects", "German", "call", "to", "boycott",…	[22, 42, 16, 21, 35, 37, 16, 21, 7]	[11, 21, 11, 12, 21, 22, 11, 12, 0]	[3, 0, 7, 0, 0, 0, 7, 0, 0]
1	["Peter", "Blackburn"]	[22, 22]	[11, 12]	[1, 2]
2	["BRUSSELS", "1996-08-22"]	[22, 11]	[11, 12]	[5, 0]
3	["The", "European", "Commission", "said", "on",…	[12, 22, 22, 38, 15, 22, 28, 38, 15, 16, 21,…	[11, 12, 12, 21, 13, 11, 11, 21, 13, 11, 12, 13,…	[0, 3, 4, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0,…
4	["Germany", "'s", "representative", "to", "the",…	[22, 27, 21, 35, 12, 22, 22, 27, 16, 21, 22,…	[11, 11, 12, 13, 11, 12, 12, 11, 12, 12, 12, 12,…	[5, 0, 0, 0, 0, 3, 4, 0, 0, 0, 1, 2, 0, 0, 0,…
5	["\"", "We", "do", "n't", "support", "any", "such",…	[0, 28, 41, 30, 37, 12, 16, 21, 15, 28, 41, 30,…	[0, 11, 21, 22, 22, 11, 12, 12, 17, 11, 21, 22,…	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,…
6	["He", "said", "further",	[28, 38, 16, 16, 21,	[11, 21, 11, 12, 12, 21,	[0, 0, 0, 0, 0, 0, 0,

{
"chunk_tags": [11, 12, 12, 21, 13, 11, 11, 21, 13, 11, 12, 13, 11, 21, 22, 11, 12, 17, 11, 21, 17, 11, 12, 12, 21, 22, 22, 13, 11, 0],
"id": "0",
"ner_tags": [0, 3, 4, 0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
"pos_tags": [12, 22, 22, 38, 15, 22, 28, 38, 15, 16, 21, 35, 24, 35, 37, 16, 21, 15, 24, 41, 15, 16, 21, 21, 20, 37, 40, 35, 21, 7],
"tokens": ["The", "European", "Commission", "said", "on", "Thursday", "it", "disagreed", "with", "German", "advice", "to", "consumers", "to", "shun", "British", "lamb", "until", "scientists", "determine", "whether", "mad", "cow", "disease", "can", "be", "transmitted", "to", "sheep", "."]
}

Code Implementation

❖ Data Preparation

```
from datasets import load_dataset

dataset = load_dataset("conll2003")
```

```
dataset = dataset.remove_columns(["id", "chunk_tags", "ner_tags"])
```

tokens: a list of string features.
pos_tags: a list of classification labels (int). Full tagset with indices:

{'"': 0, "''": 1, '#': 2, '$': 3, '(': 4, ')': 5, ',': 6, '.': 7, ':': 8, '``': 9, 'CC': 10, 'CD': 11, 'DT': 12, 'EX': 13, 'FW': 14, 'IN': 15, 'JJ': 16, 'JJR': 17, 'JJS': 18, 'LS': 19, 'MD': 20, 'NN': 21, 'NNP': 22, 'NNPS': 23, 'NNS': 24, 'NN|SYM': 25, 'PDT': 26, 'POS': 27, 'PRP': 28, 'PRP$': 29, 'RB': 30, 'RBR': 31, 'RBS': 32, 'RP': 33, 'SYM': 34, 'TO': 35, 'UH': 36, 'VB': 37, 'VBD': 38, 'VBG': 39, 'VBN': 40, 'VBP': 41, 'VBZ': 42, 'WDT': 43, 'WP': 44, 'WP$': 45, 'WRB': 46}

Code Implementation

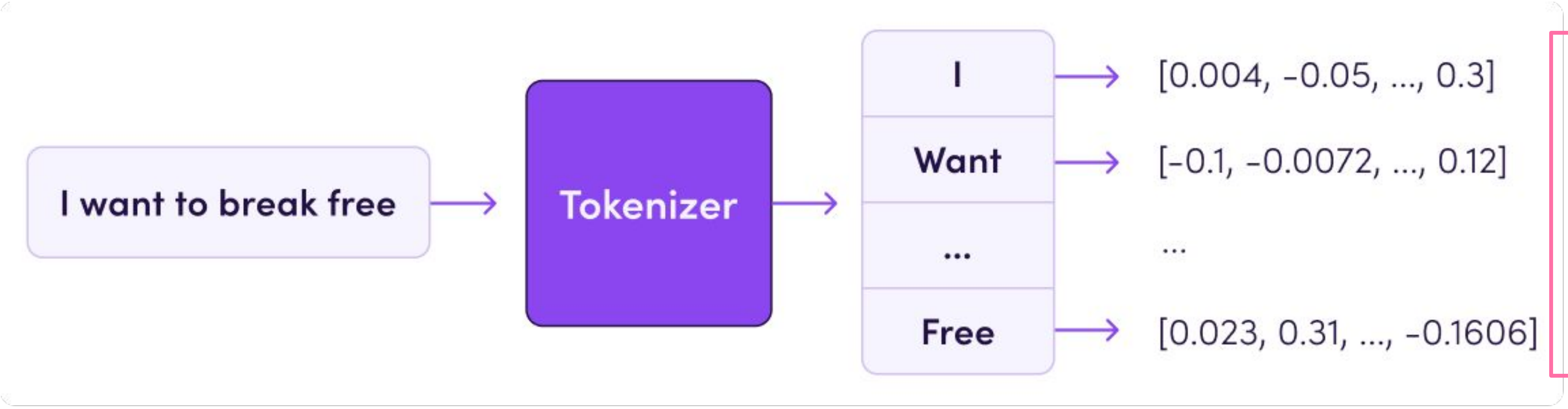
❖ Data Splits

```python
dataset_train = dataset["train"]
dataset_val = dataset["validation"]
dataset_test = dataset["test"]
```

name	train	validation	test
conll2003	14041	3250	3453

Code Implementation

❖ AutoTokenizer



Automatic Tokenizer Loading: The AutoTokenizer class can automatically detect and load the appropriate tokenizer for a given pre-trained model. This is highly convenient because different models in the Transformers library (like BERT, GPT, T5, etc.) require different tokenizers due to their distinct architectures and training setups.

```
SET tokenizer TO AutoTokenizer.from_pretrained("bert-base-uncased")
SET MAX_LEN TO 113

SET train_set TO PosTagging_Dataset(dataset_train, tokenizer)
SET val_set TO PosTagging_Dataset(dataset_val, tokenizer)
SET test_set TO PosTagging_Dataset(dataset_test, tokenizer)
```

Code Implementation

❖ PosTagging_Dataset Example

Step1: get input and label of 1 sample

Step2: tokenize

Step3:Pad and Truncate


```
num_labels = 46 + 1
MAX_LEN = 8
sample1 = ['Rare', 'Hendrix', 'song', 'draft', 'sells', 'for', 'almost', '$', '17,000']
sample2 = ['EU', 'rejects', 'call', 'to', 'boycott', 'British', 'lamb']
sentences = [sample1, sample2]
label1 = [22, 22, 21, 21, 42, 15, 30, 3, 11]
label2 = [22, 42, 21, 35, 37, 16, 21]
labels = [label1, label2]
```

Code Implementation

❖ PosTagging_Dataset

```
DEFINE CLASS PosTagging_Dataset(Dataset):

    DEFINE FUNCTION __init__(self, dataset, tokenizer):

        super().__init__()

        SET self.tokens TO dataset["tokens"]

        SET self.labels TO dataset["pos_tags"]

        SET self.tokenizer TO tokenizer

        SET self.max_len TO MAX_LEN

    DEFINE FUNCTION __len__(self):

        RETURN len(self.tokens)
```

```
    DEFINE FUNCTION __getitem__(self, idx):

        SET INPUT_token TO self.tokens[idx]

        SET label_token TO self.labels[idx]

        SET INPUT_token TO self.tokenizer.convert_tokens_to_ids(INPUT_token)

        SET INPUT_ids, labels TO self.pad_and_truncate(INPUT_token, label_token)

        RETURN INPUT_ids, labels

    DEFINE FUNCTION pad_and_truncate(self, sequence_token, sequence_label):

        SET pad_id TO self.tokenizer.pad_token_id

        IF len(sequence_token) < self.max_len:

            SET padded_sequence_token TO sequence_token + [pad_id] * (self.max_len - len(sequence_token))

            SET padded_sequence_label TO sequence_label + [47] * (self.max_len - len(sequence_label))

        ELSE:

            SET padded_sequence_token TO sequence_token[:self.max_len]

            SET padded_sequence_label TO sequence_label[:self.max_len]

        RETURN torch.tensor(padded_sequence_token), torch.tensor(padded_sequence_label)
```

Code Implementation

❖ PosTagging_Dataset Example

Step1: get input and label of 1 sample

Step2: tokenize

Step3:Pad and Truncate


```python
num_labels = 46 + 1
MAX_LEN = 8
sample1 = ['Rare', 'Hendrix', 'song', 'draft', 'sells', 'for', 'almost', '$', '17,000']
sample2 = ['EU', 'rejects', 'call', 'to', 'boycott', 'British', 'lamb']
sentences = [sample1, sample2]
label1 = [22, 22, 21, 21, 42, 15, 30, 3, 11]
label2 = [22, 42, 21, 35, 37, 16, 21]
labels = [label1, label2]
```

```python
for idx, data in enumerate(train_loader):
    x,y = data
    print('Sample ', idx, ':')
    print(x)
    print(y)
```

```
Sample  0 :
tensor([[  100,   100,  2299,  4433, 15187,  2005,  2471,  1002]])
tensor([[22, 22, 21, 21, 42, 15, 30,  3]])
Sample  1 :
tensor([[  100, 19164,  2655,  2000, 17757,   100, 12559,     0]])
tensor([[22, 42, 21, 35, 37, 16, 21, 47]])
```

Code Implementation

❖ PosTagging_Dataset Example

Step1: get input and label of 1 sample


```python
import torch
import torch.nn as nn
from torch.utils.data import Dataset
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
num_labels = 46 + 1
MAX_LEN = 8
sample1 = ['Rare', 'Hendrix', 'song', 'draft', 'sells', 'for', 'almost', '$', '17,000']
sample2 = ['EU', 'rejects', 'call', 'to', 'boycott', 'British', 'lamb']
sentences = [sample1, sample2]
label1 = [22, 22, 21, 21, 42, 15, 30, 3, 11]
label2 = [22, 42, 21, 35, 37, 16, 21]
labels = [label1, label2]
```

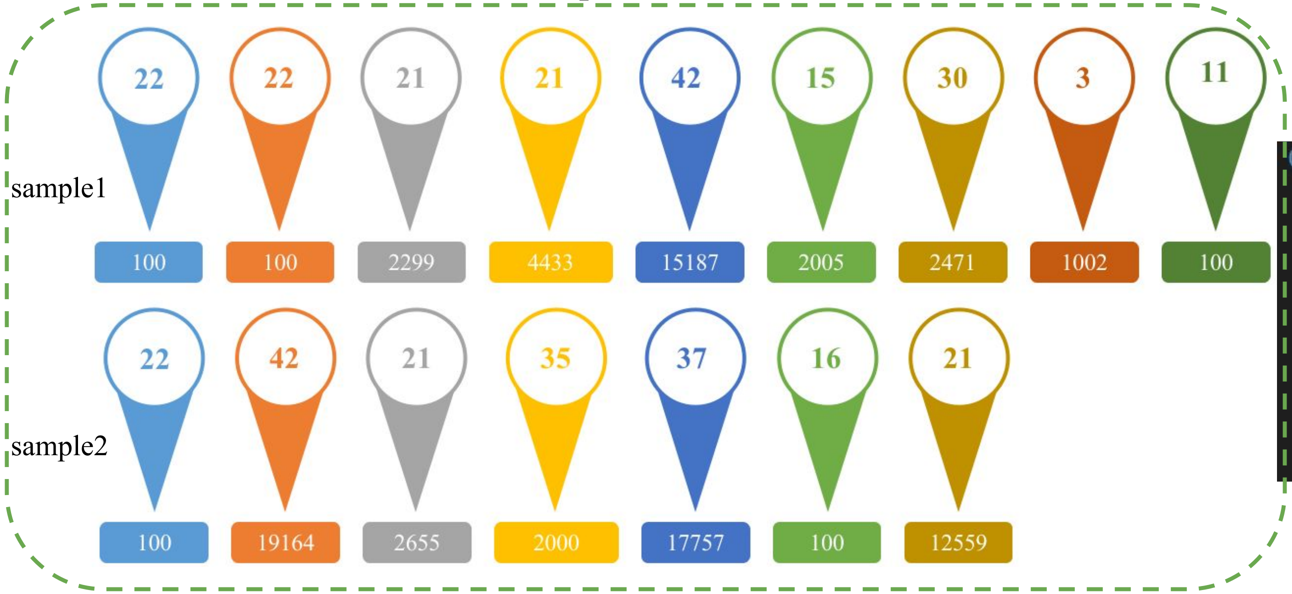
```python
from torch.utils.data import Dataset

class MyDataset(Dataset):
    def __init__(self, tokens, pos_tags, tokenizer):
        super().__init__()
        self.tokens = tokens
        self.labels = pos_tags
        self.tokenizer = tokenizer
        self.max_len = MAX_LEN

    def __len__(self):
        return len(self.tokens)
```

```python
def __getitem__(self, idx):
    input_token = self.tokens[idx]
    label_token = self.labels[idx]

    input_token = self.tokenizer.convert_tokens_to_ids(input_token)
    input_ids, labels = self.pad_and_truncate(input_token, label_token)

    return input_ids, labels
```

Code Implementation

❖ PosTagging_Dataset Example

Step2: tokenize




```python
def __getitem__(self, idx):
    input_token = self.tokens[idx]
    label_token = self.labels[idx]

    input_token = self.tokenizer.convert_tokens_to_ids(input_token)
    input_ids, labels = self.pad_and_truncate(input_token, label_token)

    return input_ids, labels
```

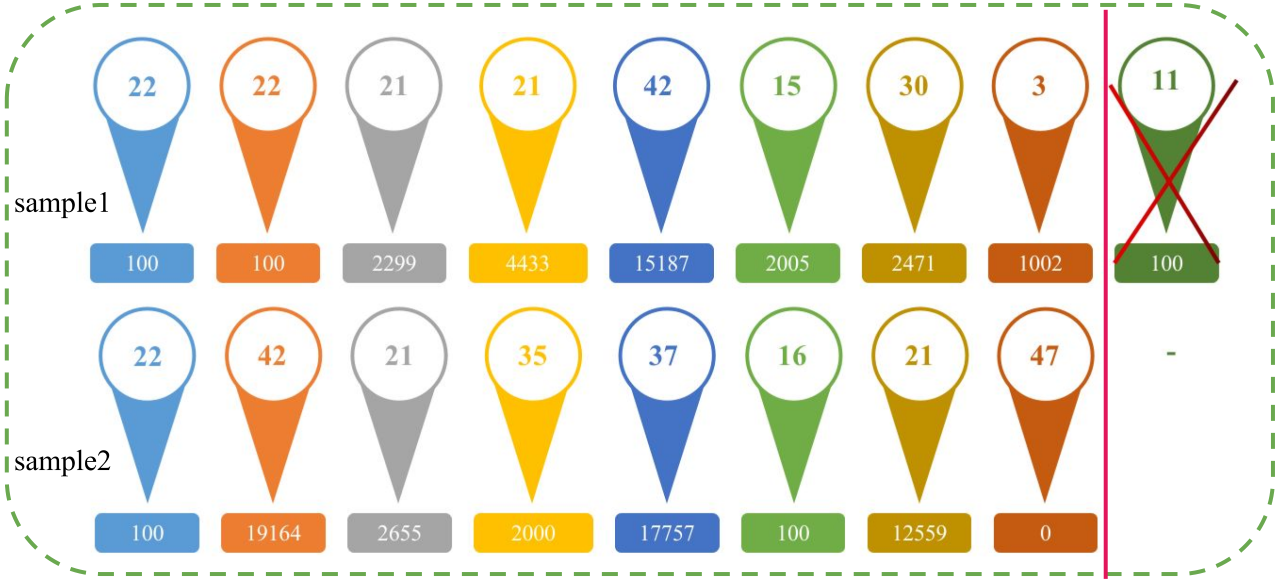
```python
import torch
import torch.nn as nn
from torch.utils.data import Dataset
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
sequence_length = 5
num_labels = 46 + 1
MAX_LEN = 8
sample1 = ['Rare', 'Hendrix', 'song', 'draft', 'sells', 'for', 'almost', '$', '17,000']
sample2 = ['EU', 'rejects', 'call', 'to', 'boycott', 'British', 'lamb']
sentences = [sample1, sample2]
label1 = [22, 22, 21, 21, 42, 15, 30, 3, 11]
label2 = [22, 42, 21, 35, 37, 16, 21]
labels = [label1, label2]
```

Code Impleme

```python
from torch.utils.data import DataLoader
train_set = MyDataset(sentences, labels, tokenizer)

batch_size = 1
train_loader = DataLoader(train_set, batch_size)
```

❖ PosTagging_Dataset Example

Step3:Pad and Truncate




```python
for idx, data in enumerate(train_loader):
    x,y = data
    print('Sample ', idx, ':')
    print(x)
    print(y)
```

```
Sample  0 :
tensor([[  100,   100,  2299,  4433, 15187,  2005,  2471,  1002]])
tensor([[22, 22, 21, 21, 42, 15, 30,  3]])
Sample  1 :
tensor([[  100, 19164,  2655,  2000, 17757,   100, 12559,     0]])
tensor([[22, 42, 21, 35, 37, 16, 21, 47]])
```

```python
def __getitem__(self, idx):
    input_token = self.tokens[idx]
    label_token = self.labels[idx]

    input_token = self.tokenizer.convert_tokens_to_ids(input_token)
    input_ids, labels = self.pad_and_truncate(input_token, label_token

    return input_ids, labels
```

```python
def pad_and_truncate(self, sequence_token, sequence_label):
    pad_id = self.tokenizer.pad_token_id
    if len(sequence_token) < self.max_len:
        padded_sequence_token = sequence_token + [pad_id] * (self.max_len - len(sequence_token))
        padded_sequence_label = sequence_label + [47] * (self.max_len - len(sequence_label))
    else:
        padded_sequence_token = sequence_token[:self.max_len]
        padded_sequence_label = sequence_label[:self.max_len]

    return torch.tensor(padded_sequence_token), torch.tensor(padded_sequence_label)
```

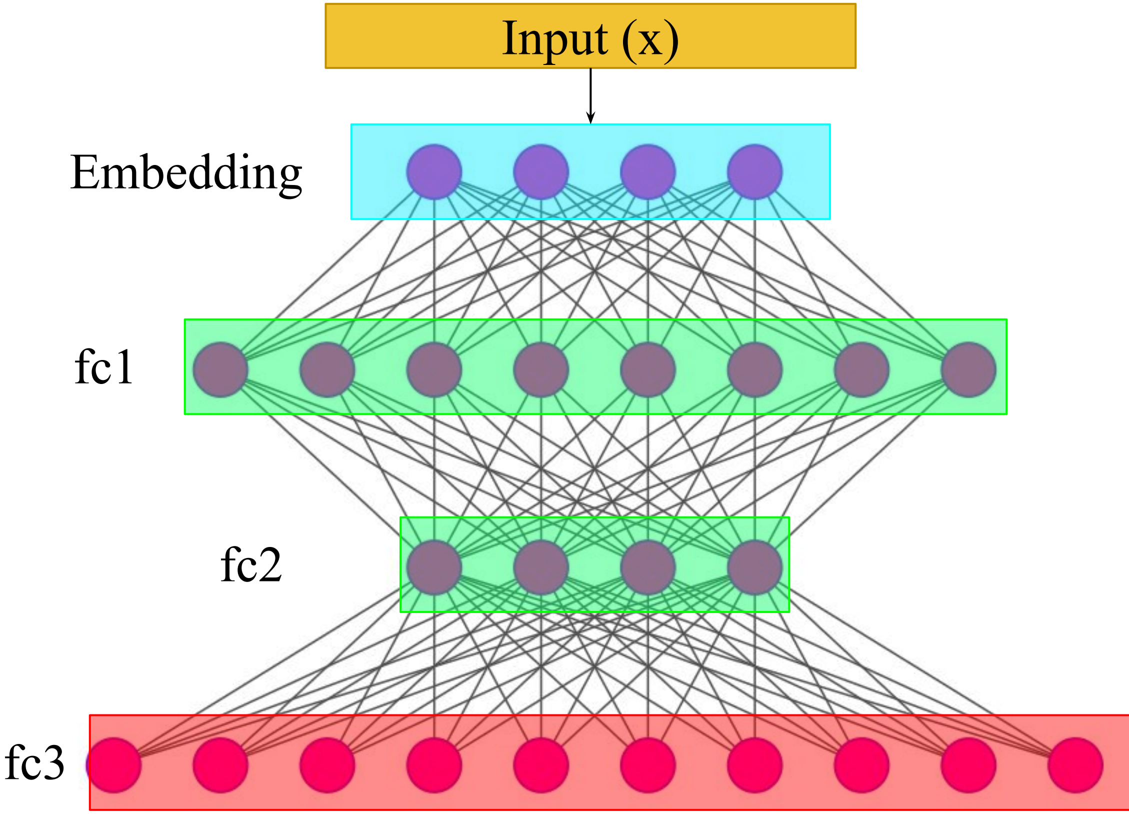
AI VIETNAM
All-In-One Course

Code Implementation

❖ POS_Model

```
DEFINE CLASS POS_Model(nn.Module):

    DEFINE FUNCTION __init__(self, vocab_size, emb_dim, hidden_size, num_classes):

        super().__init__()

        SET self.embedding TO nn.Embedding(vocab_size, emb_dim)

        SET self.fc1 TO nn.Linear(hidden_size, 2*hidden_size)

        SET self.fc2 TO nn.Linear(2*hidden_size, hidden_size)

        SET self.fc3 TO nn.Linear(hidden_size, num_classes)


    DEFINE FUNCTION forward(self, x):

        SET x TO self.embedding(x)

        SET x TO F.relu(self.fc1(x))

        SET x TO F.relu(self.fc2(x))

        SET x TO self.fc3(x)

        RETURN x.permute(0, 2, 1)
```

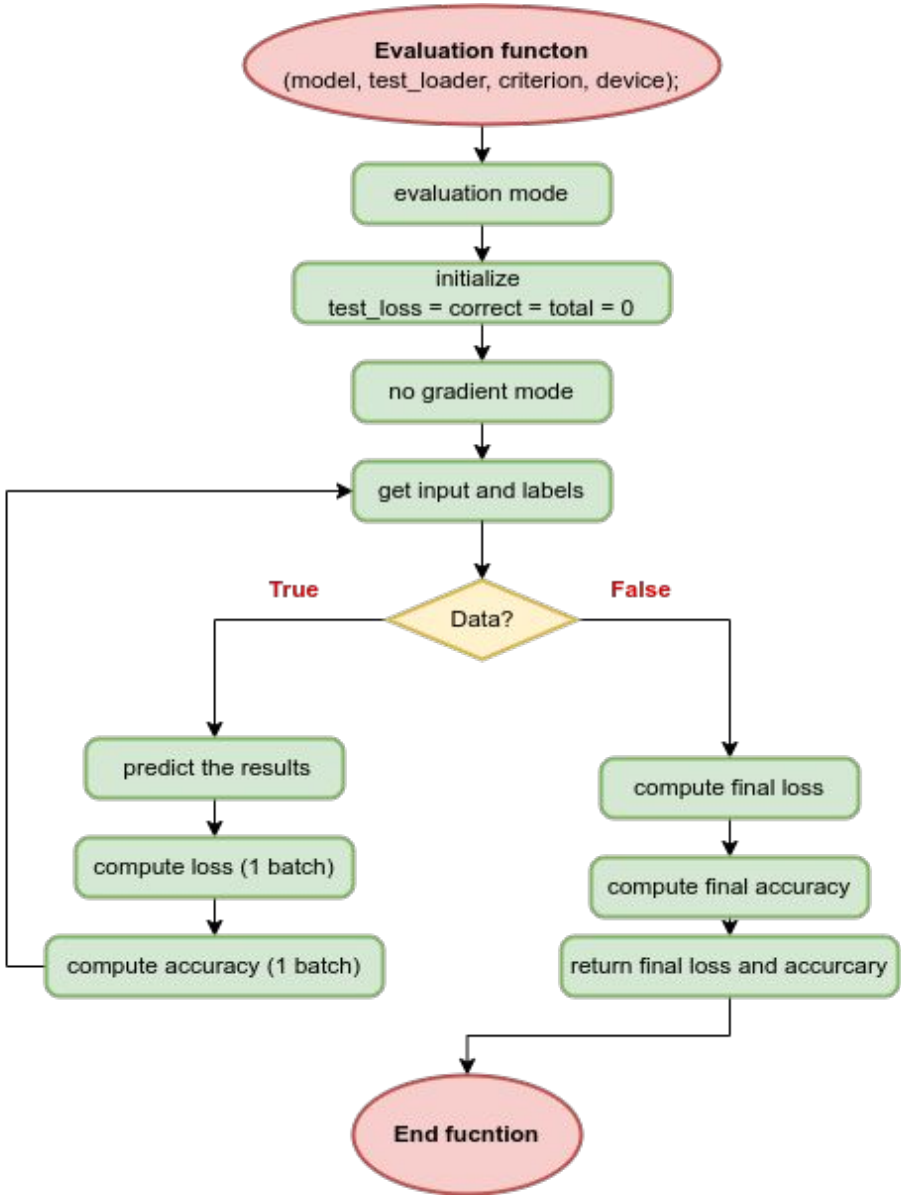
```
SET emb_dim TO 512
SET hidden_size TO 512
SET vocab_size TO len(tokenizer)
SET num_classes TO 47+1

SET model TO POS_Model(vocab_size, emb_dim,
                       hidden_size, num_classes)
```



Code Implementation

❖ Evaluate

```
DEFINE FUNCTION evaluate(model, test_loader, criterion, device):
    model.eval()
    SET test_loss TO 0.0
    SET correct TO 0
    SET total TO 0
    with torch.no_grad():
        FOR INPUTs, labels IN test_loader:
            SET INPUTs, labels TO INPUTs.to(device), labels.to(device)
            SET outputs TO model(INPUTs)
            SET loss TO criterion(outputs, labels)
            SET _, predicted TO torch.max(outputs, 1)
            total += (labels!=47).sum().item()
            test_loss += loss.item()
            correct += torch.multiply(predicted EQUALS labels, labels!=47).sum().item()
    SET test_loss TO test_loss / len(test_loader)
    SET accuracy TO 100* correct / total
    RETURN test_loss, accuracy
```



Code Implementation

❖ Train

```
SET device TO 'cuda' IF torch.cuda.is_available() else 'cpu'
SET EPOCHS TO 50
SET LR TO 1e-3
SET criterion TO nn.CrossEntropyLoss(ignore_index=47)
```

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \qquad \text{Loss} = -\sum_{i=1}^{N} y_i \cdot \log \hat{y}_i$$

Example1: N=1, C=4

z (N,C)

1.2	3.6	0.1	-2.5

y (N,)

3

softmax(z) =

0	1	2	3
0.0808	0.8903	0.0269	**0.0020**

Loss = -log(0.0020) ≃ 6.2146

Example2: N=1, C=3, d=2

z (N,C, d)

1.2	3.6
0.1	-2.5
0.1	-0.2

z (N,C, d)

1.2	3.6	0.1
3.6	-2.5	-0.2

y (N,d)

1	0

softmax(z) =

0	1	2
0.6003	**0.1998**	0.1998
0.9760	0.0022	0.0218

Loss = -[(log(0.1998) + log(0.9760)]/2 ≃ 0.8173

Code Implementation

❖ Train

```
SET device TO 'cuda' IF torch.cuda.is_available() else 'cpu'
SET EPOCHS TO 50
SET LR TO 1e-3
SET criterion TO nn.CrossEntropyLoss(ignore_index=47)
```

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

$$\text{Loss} = -\sum_{i=1}^{N} y_i \cdot \log \hat{y}_i$$

Example3: N=1, C=3, d=2, ignore_index=1

z (N,C, d)

1.2	3.6
0.1	-2.5
0.1	-0.2

z (N,C, d)

1.2	3.6	0.1
3.6	-2.5	-0.2

y (N,d)

1	0

softmax(z) =

0	1	2
0.6003	**0.1998**	0.1998
0.9760	0.0022	0.0218

softmax(z) =

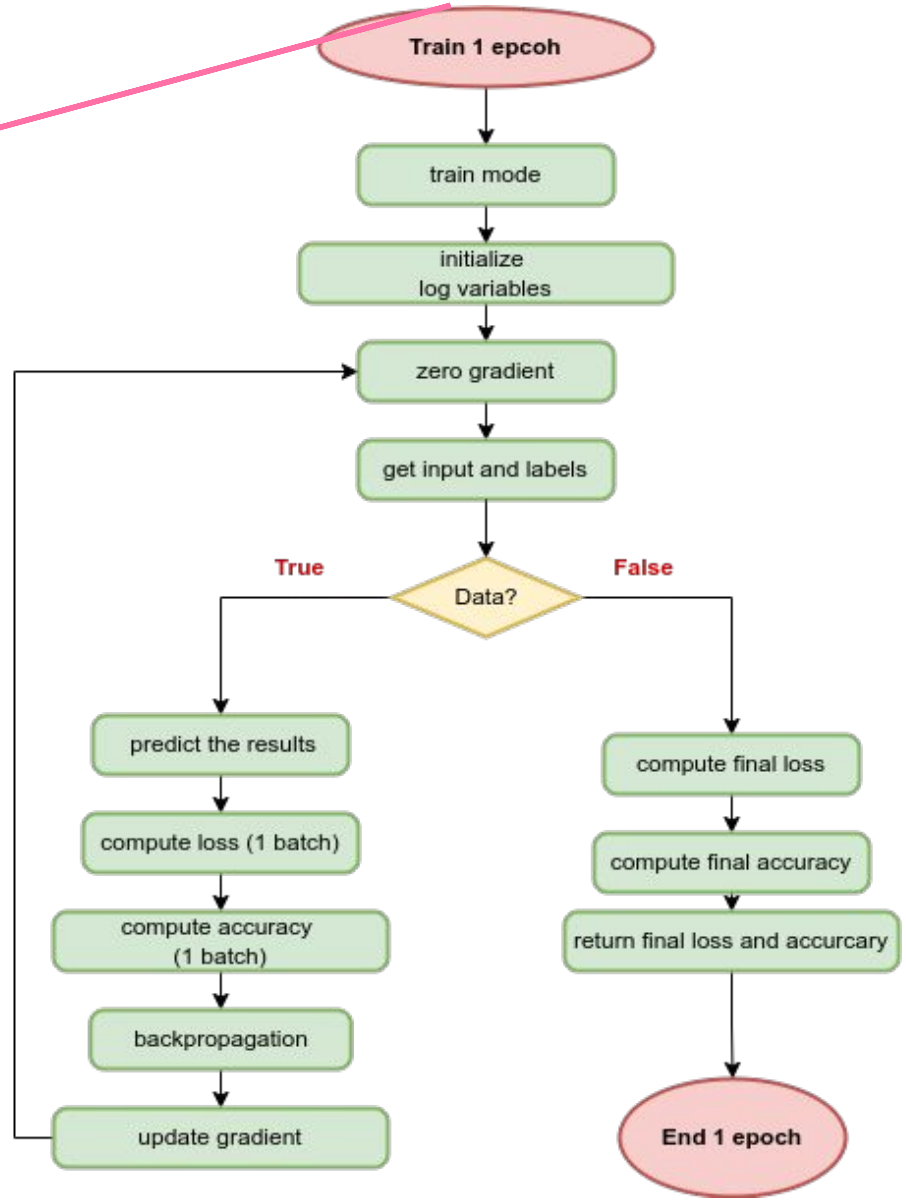
0	1	2
~~0.6003~~	~~**0.1998**~~	~~0.1998~~
0.9760	0.0022	0.0218

Loss = -log(0.9760) ≃ 0.0243

```
DEFINE FUNCTION train_model(model):

    SET hist TO {
        "train_loss": [],
        "train_accuracy": [],
        "val_loss": [],
        "val_accuracy": []
    }

    SET optimizer TO torch.optim.Adam(model.parameters(), lr=LR)

    model.to(device)

    FOR epoch IN range(EPOCHS):
        model.train()
        SET running_loss TO 0.0
        SET running_correct TO 0
        SET total TO 0

        FOR INPUTs, labels IN train_loader:
            SET INPUTs, labels TO INPUTs.to(device), labels.to(device)
            optimizer.zero_grad()

            SET outputs TO model(INPUTs)

            SET loss TO criterion(outputs, labels)

            running_loss += loss.item()
            SET _, predicted TO torch.max(outputs, 1)
            total += (labels!=47).sum().item()
            running_correct += torch.multiply(predicted EQUALS labels, labels!=47).sum().item()

            loss.backward()

            optimizer.step()

        SET epoch_loss TO  running_loss / len(train_loader)

        SET epoch_accuracy TO 100* running_correct / total

        SET val_loss, val_accuracy TO evaluate(model, val_loader, criterion, device)


        hist["train_loss"].append(epoch_loss)
        hist["train_accuracy"].append(epoch_accuracy)
        hist["val_loss"].append(val_loss)
        hist["val_accuracy"].append(val_accuracy)

    RETURN hist
```

tation

❖ Train



Code Implementation

❖ Plot Results

```
plot(hist['train_loss'], label='Train')
plot(hist['val_loss'], label='Val')

title('model loss')
ylabel('loss')
xlabel('epoch')
legend(['Train', 'Val' ], loc='upper right')
show()
```