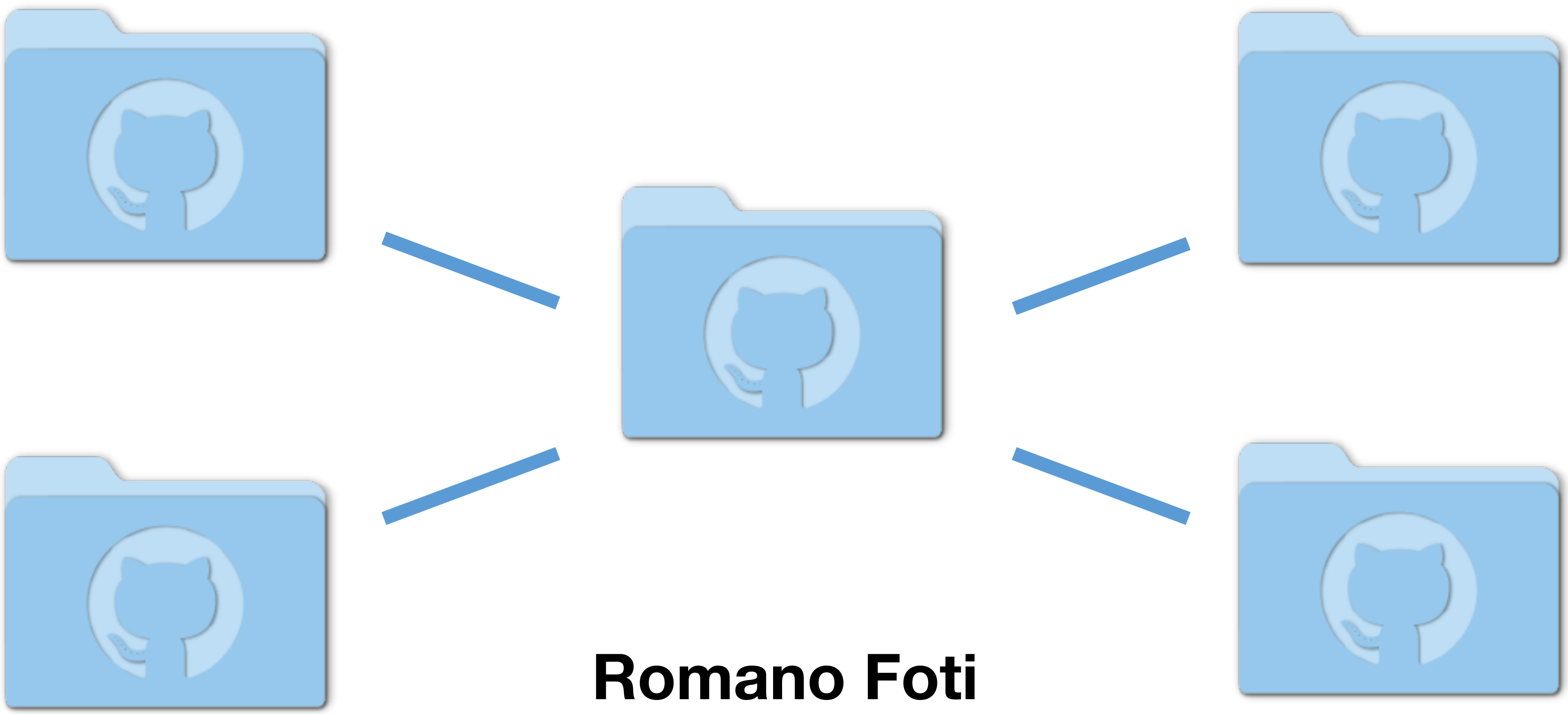
Characterization of GitHub Repositories: A Natural Language Processing Approach



Romano Foti

2019 ODSC Europe
London, November 2019

Problem

Why is repository characterization important?

- Contain the vast majority of the code hosted at GitHub
- Primary GitHub resource for developers, contributors and students
- Facilitates content discoverability
- Promotes collaboration

Challenges

- Almost 150M count
- Highly variable content
- Highly variable quality
- Meta-information not always informative or predictive
- Majority of content (code) is not written in a Natural Language

Problem

Objective

Characterizing repositories to provide personalized recommendations

Strategy

Leveraging NLP without looking at repository content

Background

The meaning of words

How do people learn to speak, read, or write?

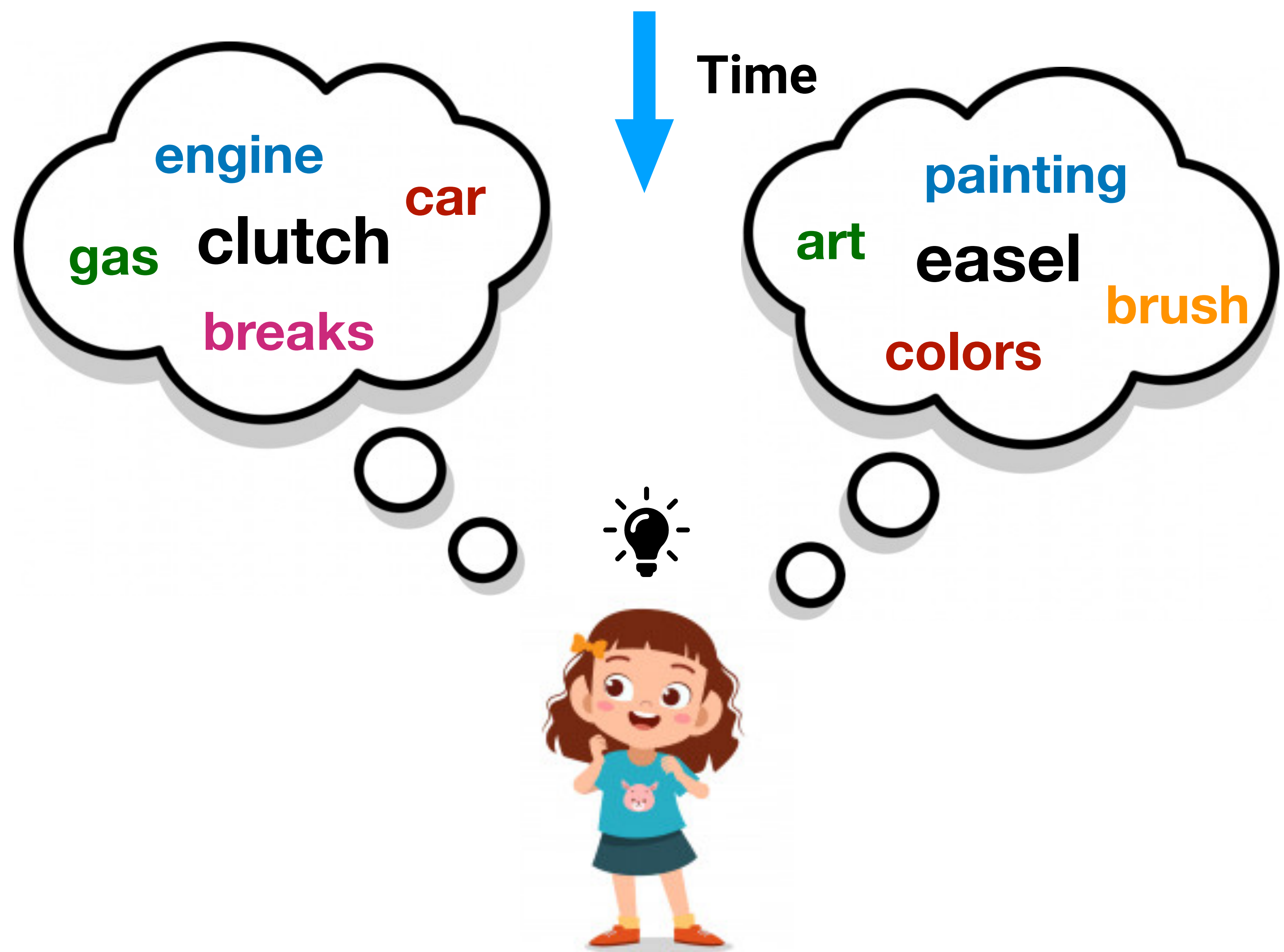
- No previous knowledge of the meaning of words
- No idea of how to build sentences with them

Background

The meaning of words

How do people learn to speak, read, or write?

- No previous knowledge of the meaning of words
- No idea of how to build sentences with them

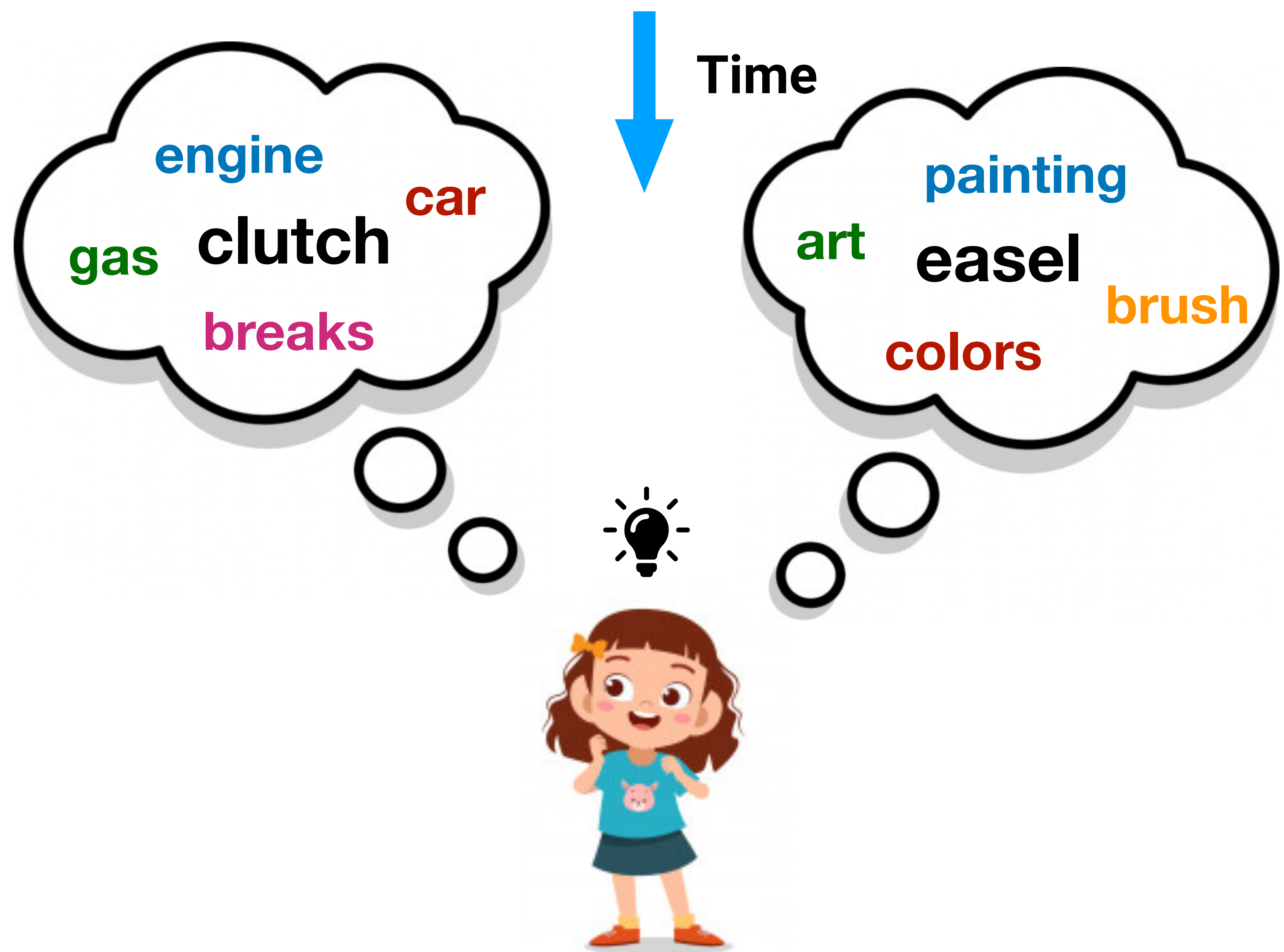


Background

The meaning of words

How do people learn to speak, read, or write?

- No previous knowledge of the meaning of words
- No idea of how to build sentences with them

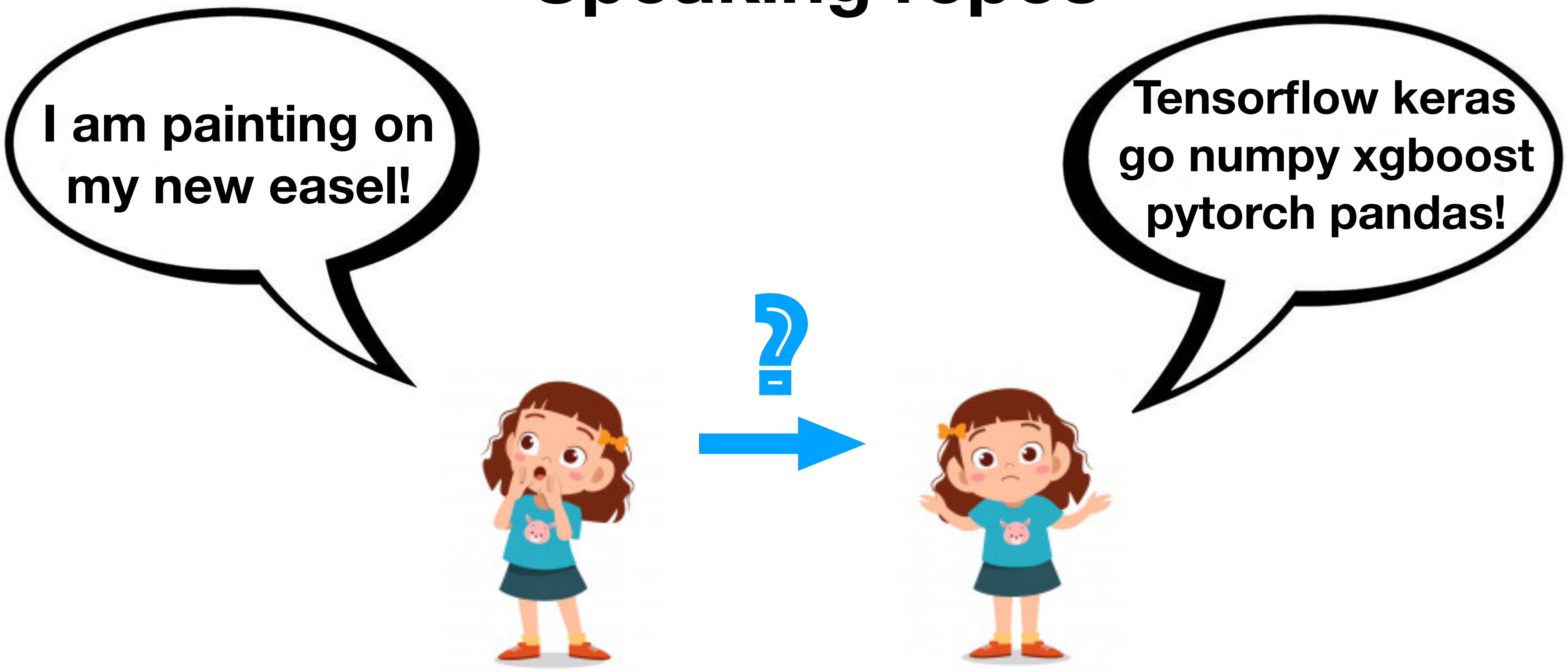


How?

Patterns & repetition

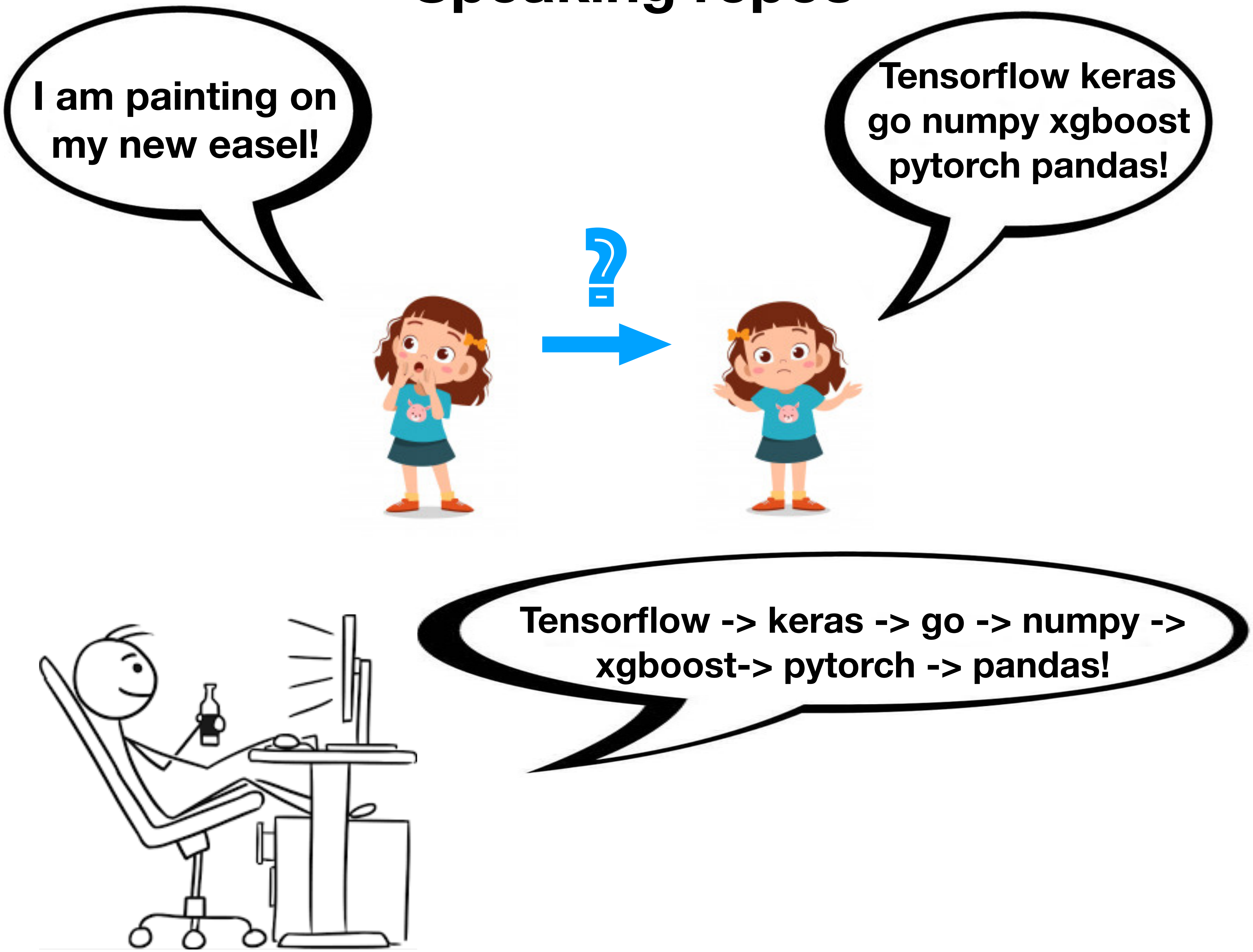
Background

Speaking repos



Background

Speaking repos



Methods

Leveraging user activity to (indirectly) extract information on repositories

Methods

Leveraging user activity to (indirectly) extract information on repositories

sentence

Tensorflow -> keras -> go -> numpy -> xgboost-> pytorch -> pandas!

word

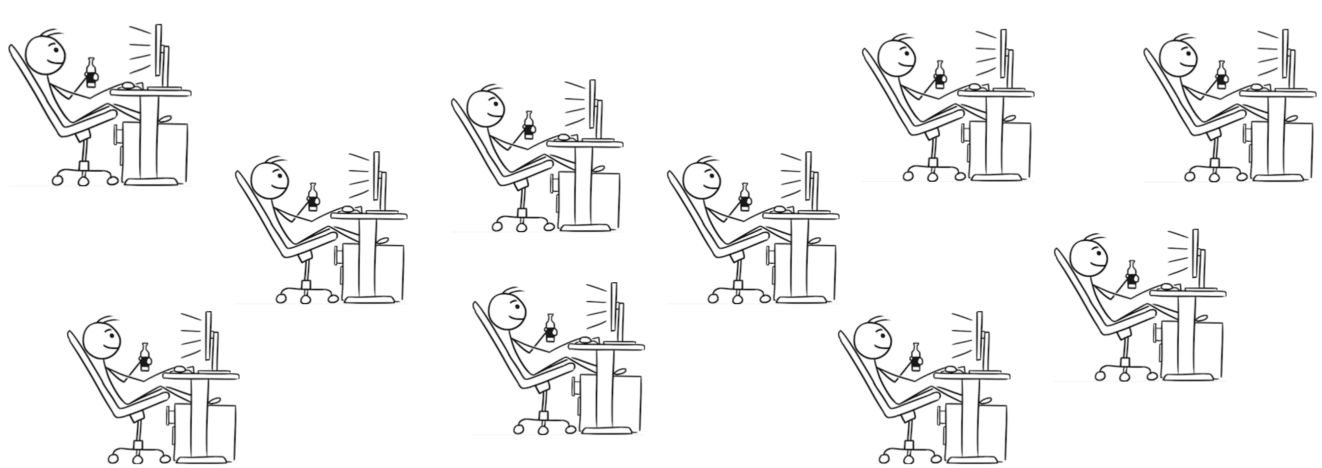
Methods

Leveraging user activity to (indirectly) extract information on repositories

sentence

Tensorflow -> keras -> go -> numpy -> xgboost-> pytorch -> pandas!

word



Patterns & repetition

Methods

Doc2vec: Learning meaning from context

Doc2vec: Learning meaning from context

Source text

Tensorflow keras go numpy xgboost pytorch pandas

Methods

Doc2vec: Learning meaning from context

Source text

Tensorflow keras go numpy xgboost pytorch pandas

Tensorflow **Seed word**

keras go **Context words**

Methods

Doc2vec: Learning meaning from context

Source text

Tensorflow | keras | go | numpy xgboost pytorch pandas

Tensorflow	**Seed word**	keras go	**Context words**

Source text → Training samples

Source text	Training samples
[**Tensorflow** keras go] numpy xgboost pytorch pandas	(tensorflow, keras) (tensorflow, go)
[Tensorflow **keras** go numpy] xgboost pytorch pandas	(keras, tensorflow) (keras, go) (keras, numpy)
[Tensorflow keras **go** numpy xgboost] pytorch pandas	(go, tensorflow) (go, keras) (go, numpy) (go, xgboost)
Tensorflow [keras go **numpy** xgboost pytorch] pandas	(numpy, keras) (numpy, go) (numpy, xgboost) (numpy, pytorch)

Doc2vec: Model architecture

Vocabulary	OneHot representation
tensorflow	(1, 0, 0, 0, 0, 0, 0)
keras	(0, 1, 0, 0, 0, 0, 0)
go	(0, 0, 1, 0, 0, 0, 0)
numpy	(0, 0, 0, 1, 0, 0, 0)
xgboost	(0, 0, 0, 0, 1, 0, 0)
pytorch	(0, 0, 0, 0, 0, 1, 0)
pandas	(0, 0, 0, 0, 0, 0, 1)

Doc2vec: Model architecture

Vocabulary	OneHot representation
tensorflow	(1, 0, 0, 0, 0, 0, 0)
keras	(0, 1, 0, 0, 0, 0, 0)
go	(0, 0, 1, 0, 0, 0, 0)
numpy	(0, 0, 0, 1, 0, 0, 0)
xgboost	(0, 0, 0, 0, 1, 0, 0)
pytorch	(0, 0, 0, 0, 0, 1, 0)
pandas	(0, 0, 0, 0, 0, 0, 1)

Training pair: (tensorflow, keras)

Input	Output
(1, 0, 0, 0, 0, 0, 0)	(0, 1, 0, 0, 0, 0, 0)

Methods

Doc2vec: Model architecture

Vocabulary	OneHot representation
tensorflow	(1, 0, 0, 0, 0, 0, 0)
keras	(0, 1, 0, 0, 0, 0, 0)
go	(0, 0, 1, 0, 0, 0, 0)
numpy	(0, 0, 0, 1, 0, 0, 0)
xgboost	(0, 0, 0, 0, 1, 0, 0)
pytorch	(0, 0, 0, 0, 0, 1, 0)
pandas	(0, 0, 0, 0, 0, 0, 1)

Vocabulary size **7**
Embedding size **10 (arbitrary)**

Training pair: (tensorflow, keras)

Input	Output
(1, 0, 0, 0, 0, 0, 0)	(0, 1, 0, 0, 0, 0, 0)

Methods

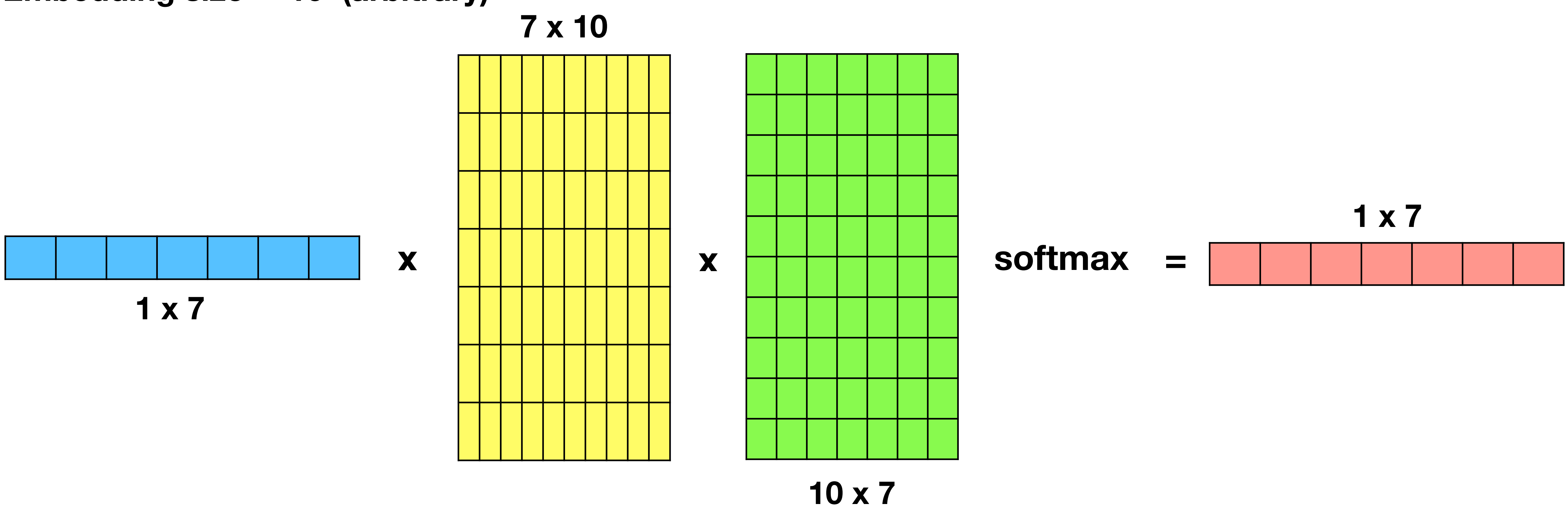
Doc2vec: Model architecture

Vocabulary	OneHot representation
tensorflow	(1, 0, 0, 0, 0, 0, 0)
keras	(0, 1, 0, 0, 0, 0, 0)
go	(0, 0, 1, 0, 0, 0, 0)
numpy	(0, 0, 0, 1, 0, 0, 0)
xgboost	(0, 0, 0, 0, 1, 0, 0)
pytorch	(0, 0, 0, 0, 0, 1, 0)
pandas	(0, 0, 0, 0, 0, 0, 1)

Training pair: (tensorflow, keras)

Input	Output
(1, 0, 0, 0, 0, 0, 0)	(0, 1, 0, 0, 0, 0, 0)

Vocabulary size 7
Embedding size 10 (arbitrary)



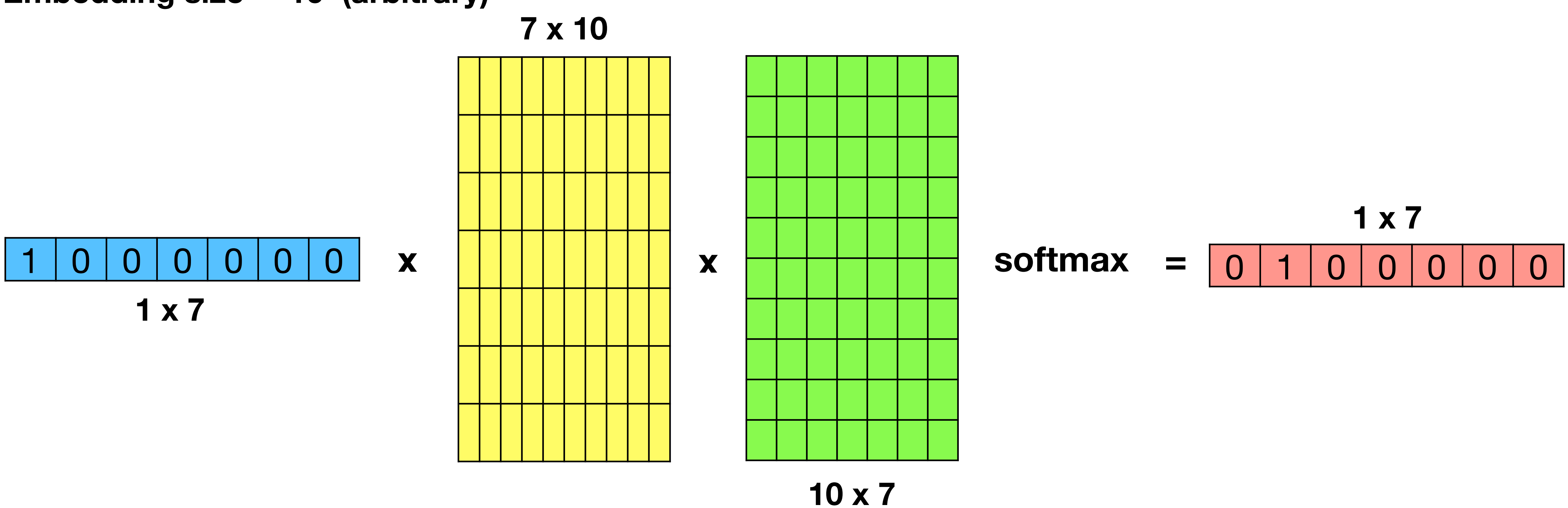
Doc2vec: Model architecture

Vocabulary	OneHot representation
tensorflow	(1, 0, 0, 0, 0, 0, 0)
keras	(0, 1, 0, 0, 0, 0, 0)
go	(0, 0, 1, 0, 0, 0, 0)
numpy	(0, 0, 0, 1, 0, 0, 0)
xgboost	(0, 0, 0, 0, 1, 0, 0)
pytorch	(0, 0, 0, 0, 0, 1, 0)
pandas	(0, 0, 0, 0, 0, 0, 1)

Training pair: (tensorflow, keras)

Input	Output
(1, 0, 0, 0, 0, 0, 0)	(0, 1, 0, 0, 0, 0, 0)

Vocabulary size 7
Embedding size 10 (arbitrary)



Methods

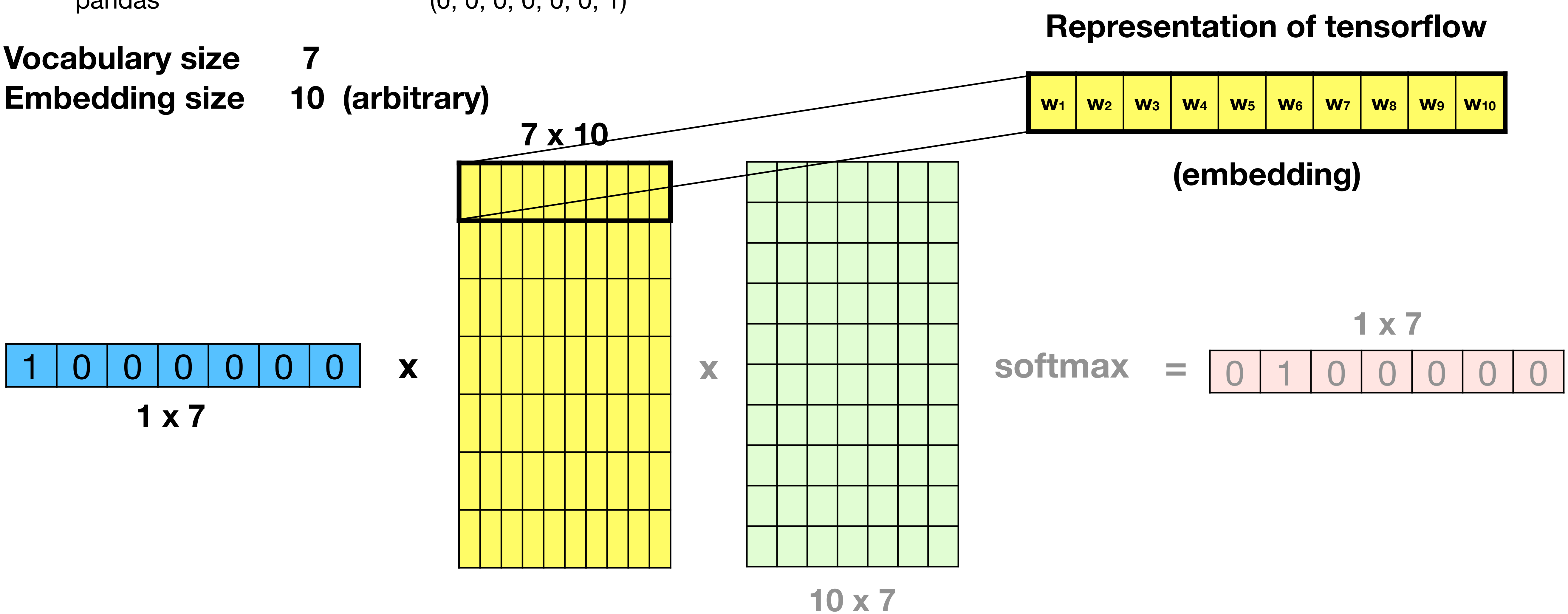
Doc2vec: Model architecture

Vocabulary	OneHot representation
tensorflow	(1, 0, 0, 0, 0, 0, 0)
keras	(0, 1, 0, 0, 0, 0, 0)
go	(0, 0, 1, 0, 0, 0, 0)
numpy	(0, 0, 0, 1, 0, 0, 0)
xgboost	(0, 0, 0, 0, 1, 0, 0)
pytorch	(0, 0, 0, 0, 0, 1, 0)
pandas	(0, 0, 0, 0, 0, 0, 1)

Training pair: (tensorflow, keras)

Input	Output
(1, 0, 0, 0, 0, 0, 0)	(0, 1, 0, 0, 0, 0, 0)

Vocabulary size 7
Embedding size 10 (arbitrary)



Methods

Workflow

Generate user view activity stream

- **user_1**: “repo_1”, “repo_72”, “repo_28”, “repo_1”, “repo_36”,…
- **user_2**: “repo_21”, “repo_41”, “repo_1”, “repo_33”, “repo_1”,…
- …
- **user_n**: “repo_40”, “repo_72”, “repo_8”, “repo_96”, “repo_31”,…
- …

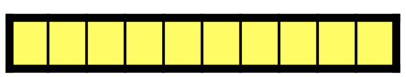
Methods

Workflow

Generate user view activity stream

↓

Apply Doc2vec and build embeddings for each repository



- **user_1**: “repo_1”, “repo_72”, “repo_28”, “repo_1”, “repo_36”,…
- **user_2**: “repo_21”, “repo_41”, “repo_1”, “repo_33”, “repo_1”,…
- …
- **user_n**: “repo_40”, “repo_72”, “repo_8”, “repo_96”, “repo_31”,…
- …

- **embedding_repo_1**: [0.021, 0.003, -0.001, -0.041, …, 0.009]
- **embedding_repo_2**: [-0.011, 0.023, -0.002, -0.019, …, 0.003]
- …
- **embedding_repo_n**: [0.039, -0.033, -0.007, -0.022, …, -0.015]
- …

Methods

Workflow

Generate user view activity stream

- **user_1**: “repo_1”, “repo_72”, “repo_28”, “repo_1”, “repo_36”,…
- **user_2**: “repo_21”, “repo_41”, “repo_1”, “repo_33”, “repo_1”,…
- …
- **user_n**: “repo_40”, “repo_72”, “repo_8”, “repo_96”, “repo_31”,…
- …

Apply Doc2vec and build embeddings for each repository

- **embedding_repo_1**: [0.021, 0.003, -0.001, -0.041, …, 0.009]
- **embedding_repo_2**: [-0.011, 0.023, -0.002, -0.019, …, 0.003]
- …
- **embedding_repo_n**: [0.039, -0.033, -0.007, -0.022, …, -0.015]
- …

Querying

$\cos(\boldsymbol{e}_i, \boldsymbol{e}_j) \sim= 1 \longrightarrow$ Repos *i* and *j* are similar

$\cos(\boldsymbol{e}_i, \boldsymbol{e}_j) \sim= 0 \longrightarrow$ Repos *i* and *j* are not similar

Results

Seed repository	Similar repositories	cosine distance
JetBrains/kotlin	JetBrains/kotlin-web-site	0.80
	Kotlin/KEEP	0.79
	JetBrains/kotlin-native	0.79
rails/rails	theSteveMitchell/after_party	0.82
	fxn/zeitwerk	0.81
	voltrb/volt	0.77
tensorflow/tensorflow	evdcush/TensorFlow-wheels	0.83
	tensorflow/autograph	0.78
	snipsco/tensorflow-build	0.76
outman123/tensorflow	ynulonger/ijcnn	0.92
	xingjianzhang1997/CatVsDog	0.92
	liush5/tensorflow-example	0.91

Results

Seed repository	Similar repositories	cosine distance
JetBrains/kotlin	JetBrains/kotlin-web-site	0.80
	Kotlin/KEEP	0.79
	JetBrains/kotlin-native	0.79
rails/rails	theSteveMitchell/after_party	0.82
	fxn/zeitwerk	0.81
	voltrb/volt	0.77
tensorflow/tensorflow	evdcush/TensorFlow-wheels	0.83
	tensorflow/autograph	0.78
	snipsco/tensorflow-build	0.76
outman123/tensorflow	ynulonger/ijcnn	0.92
	xingjianzhang1997/CatVsDog	0.92
	liush5/tensorflow-example	0.91

Validation

- **Assess the model's ability to find similar repositories**
- **Detect cheating (overestimating similarity)**

Validation

- **Assess the model's ability to find similar repositories**
- **Detect cheating (overestimating similarity)**

Building a set of similar repositories: alias strategy

• **user_1**: "**repo_1**", "repo_72", "repo_7", "**repo_1**", "repo_36",… • **user_2**: "repo_21", "**repo_1**", "repo_67", "repo_33", "**repo_1**",… • … • **user_n**: "repo_40", "repo_72", "**repo_1**", "repo_96", "repo_31",…	→	• **user_1**: "**repo_1x**", "repo_72", "repo_7", "**repo_1**", "repo_36",… • **user_2**: "repo_21", "**repo_1**", "repo_67", "repo_33", "**repo_1**",… • … • **user_n**: "repo_40", "repo_72", "**repo_1x**", "repo_96", "repo_31",…

Validation

- **Assess the model's ability to find similar repositories**
- **Detect cheating (overestimating similarity)**

Building a set of similar repositories: alias strategy

Original		Aliased
• **user_1**: "**repo_1**", "repo_72", "repo_7", "**repo_1**", "repo_36",…	→	• **user_1**: "**repo_1x**", "repo_72", "repo_7", "**repo_1**", "repo_36",…
• **user_2**: "repo_21", "**repo_1**", "repo_67", "repo_33", "**repo_1**",…		• **user_2**: "repo_21", "**repo_1**", "repo_67", "repo_33", "**repo_1**",…
• …		• …
• **user_n**: "repo_40", "repo_72", "**repo_1**", "repo_96", "repo_31",…		• **user_n**: "repo_40", "repo_72", "**repo_1x**", "repo_96", "repo_31",…

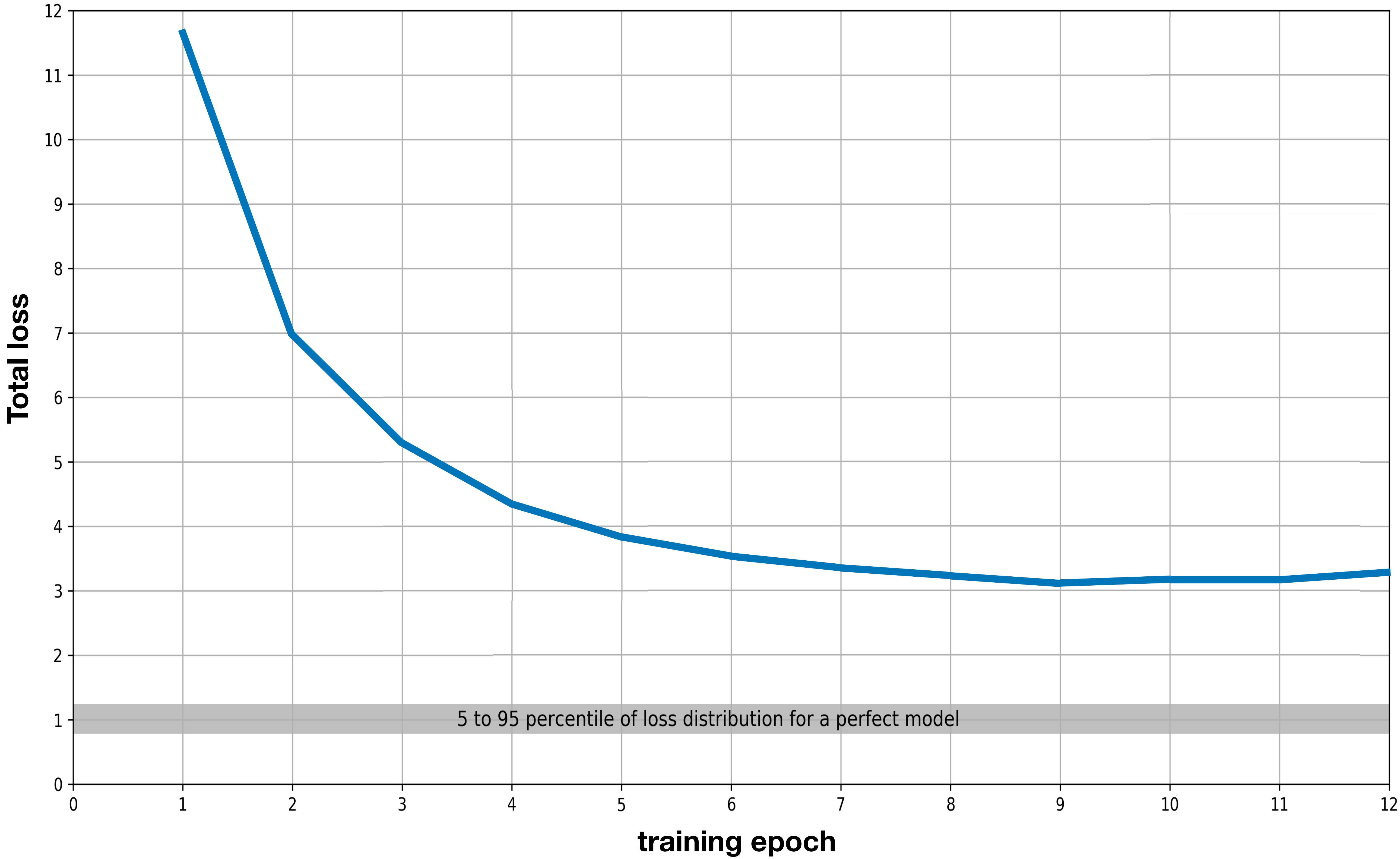
Addressing non-similar repositories: picking a set of random pairs

Total Loss = AliasRepoLoss + RandomRepoLoss

$$AliasRepoLoss = \sum_{i=1}^{N_{alias}} (1 - cos(\overrightarrow{e}_{i_{original}}, \overrightarrow{e}_{i_{alias}}))^2$$

$$RandomRepoLoss = \sum_{i=1}^{N_{random}} cos(\overrightarrow{e}_{i_1}, \overrightarrow{e}_{i_2})^2$$

Validation



Total loss
training epoch
5 to 95 percentile of loss distribution for a perfect model
0
1
2
3
4
5
6
7
8
9
10
11
12

Conclusion

- NLP techniques can successfully be used outside their natural field of application
- NLP can be applied across domains, wherever sequences of actions are performed and their underlying entities need to be characterized
- Simple implementations (e.g. Doc2Vec) can be very powerful while easy to implement
- Validation of such applications is not trivial so one needs often to get creative

Questions?

Additional resources:

- **Contact:** romanofoti@github.com
- **Appendix:** see below
- **Resource repository:** https://github.com/romanofoti/odsc_2019

Exploring incremental training

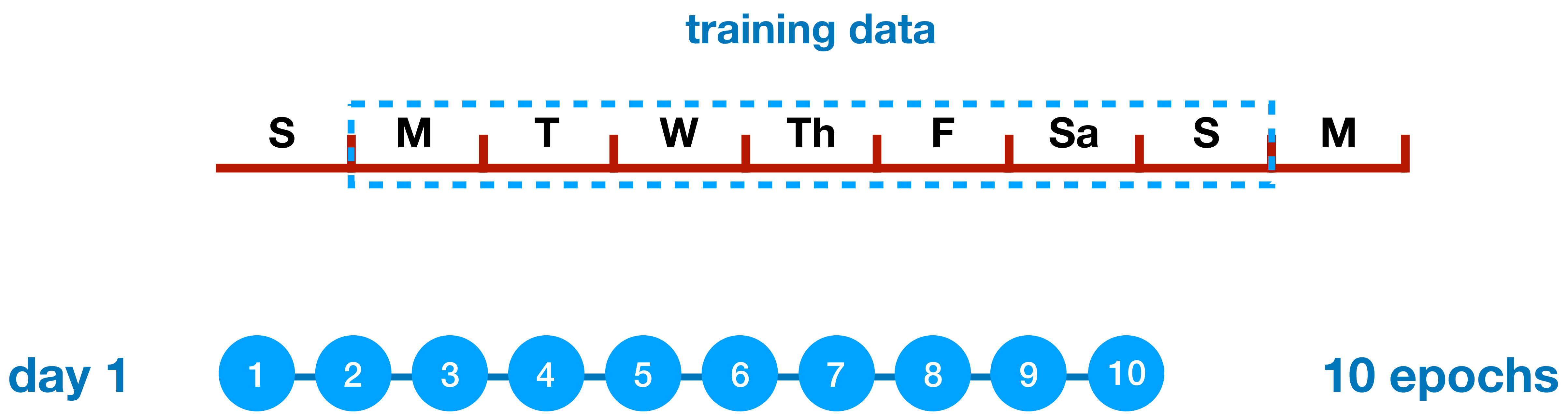
How about pre-training, saving, then resuming training?

- Constant need to update
- Time or resources constrains
- Better results

Exploring incremental training

How about pre-training, saving, then resuming training?

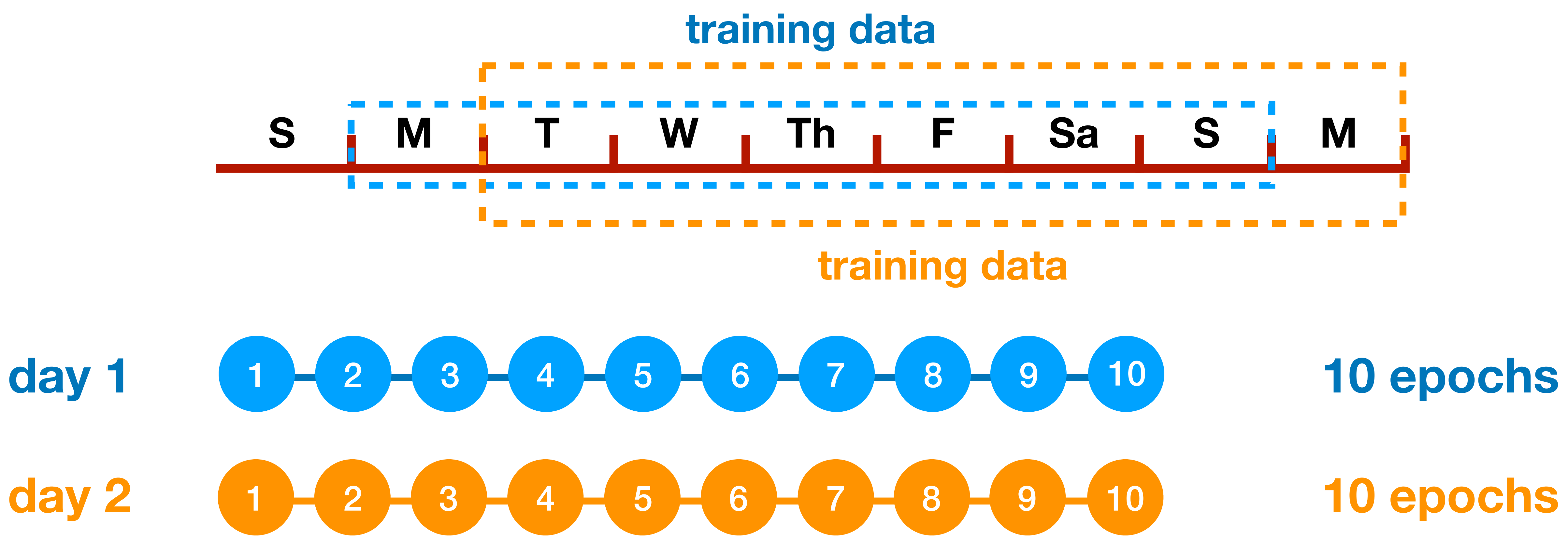
- Constant need to update
- Time or resources constrains
- Better results



Exploring incremental training

How about pre-training, saving, then resuming training?

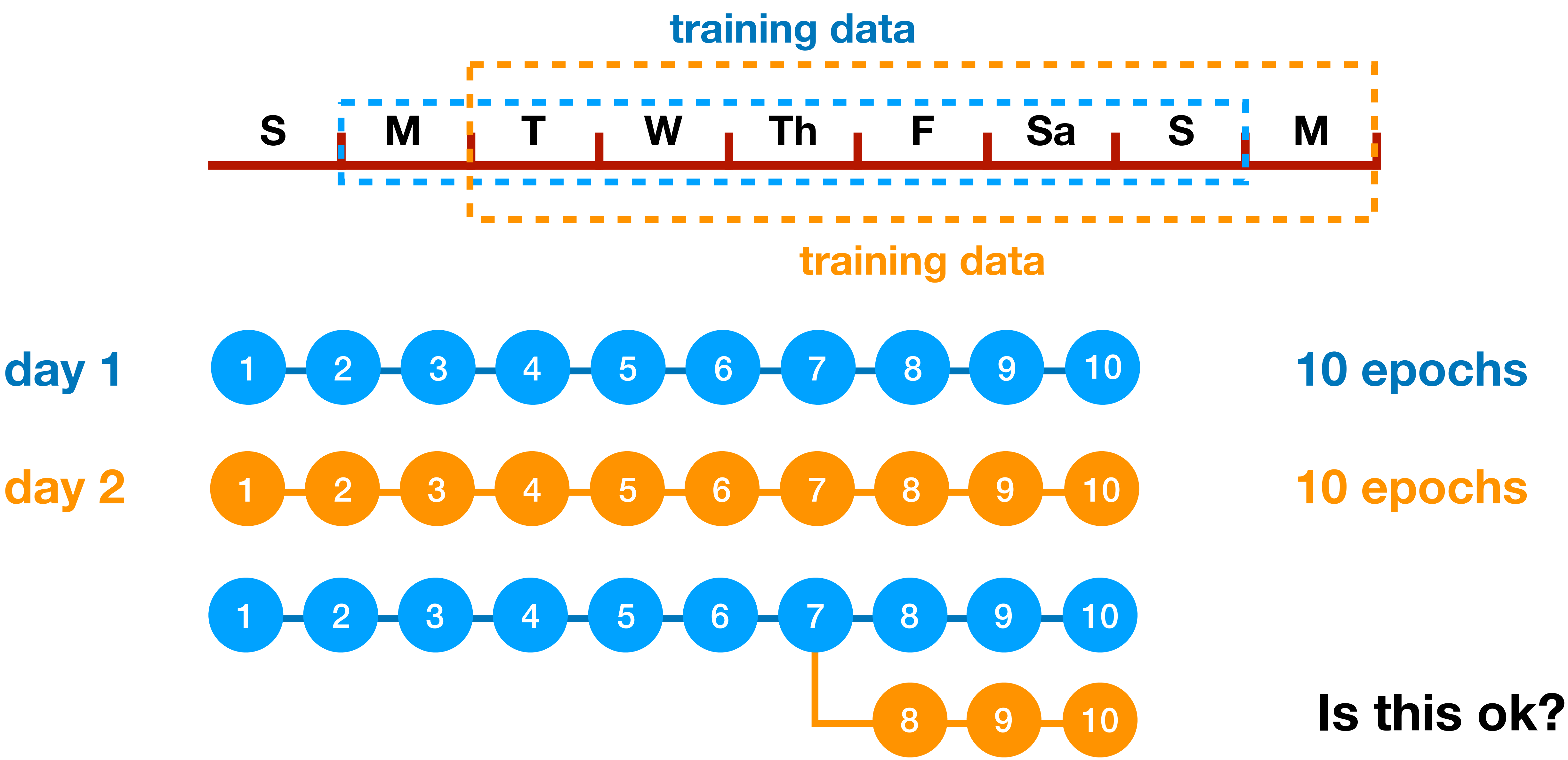
- Constant need to update
- Time or resources constrains
- Better results



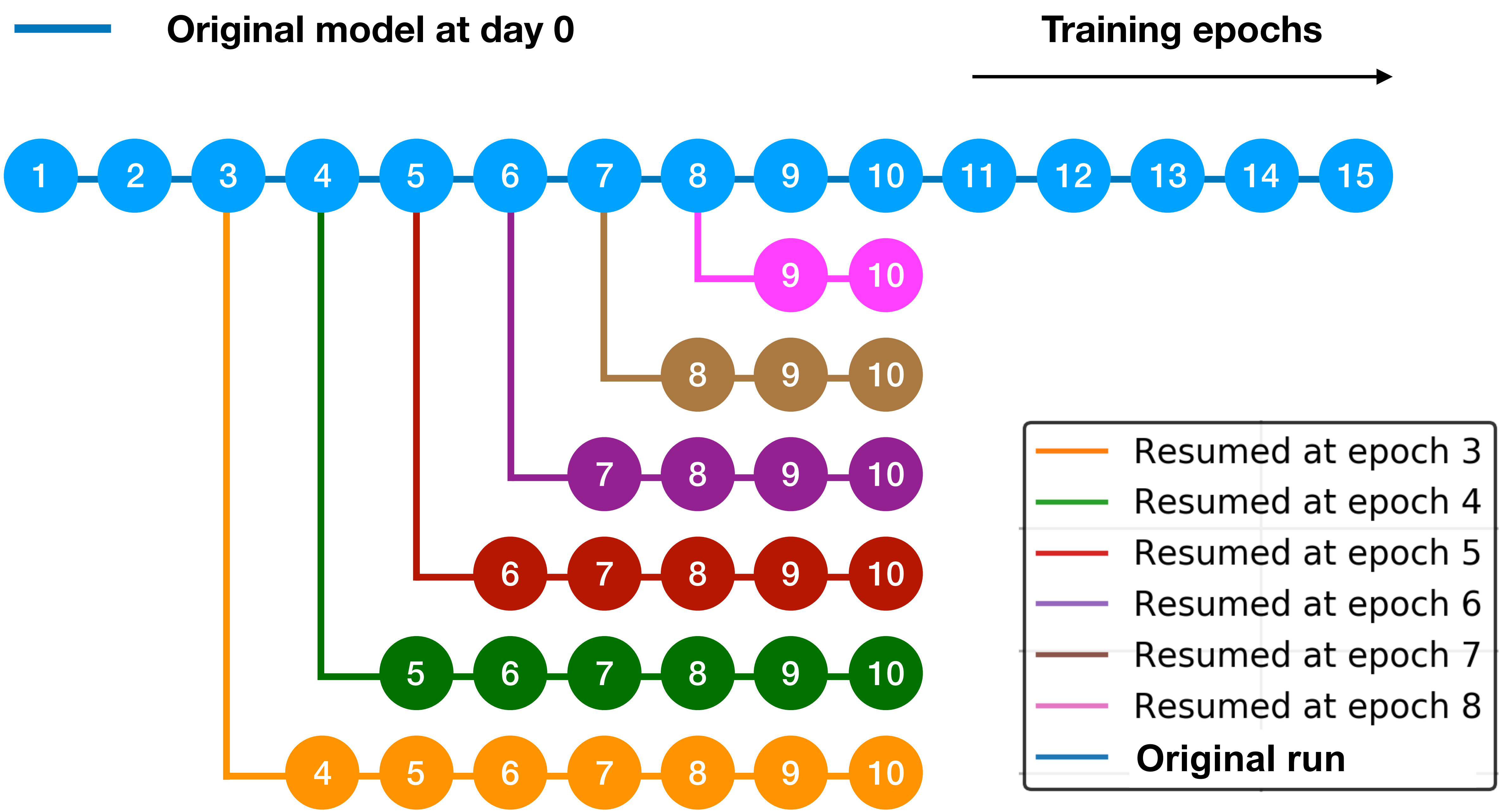
Exploring incremental training

How about pre-training, saving, then resuming training?

- Constant need to update
- Time or resources constrains
- Better results

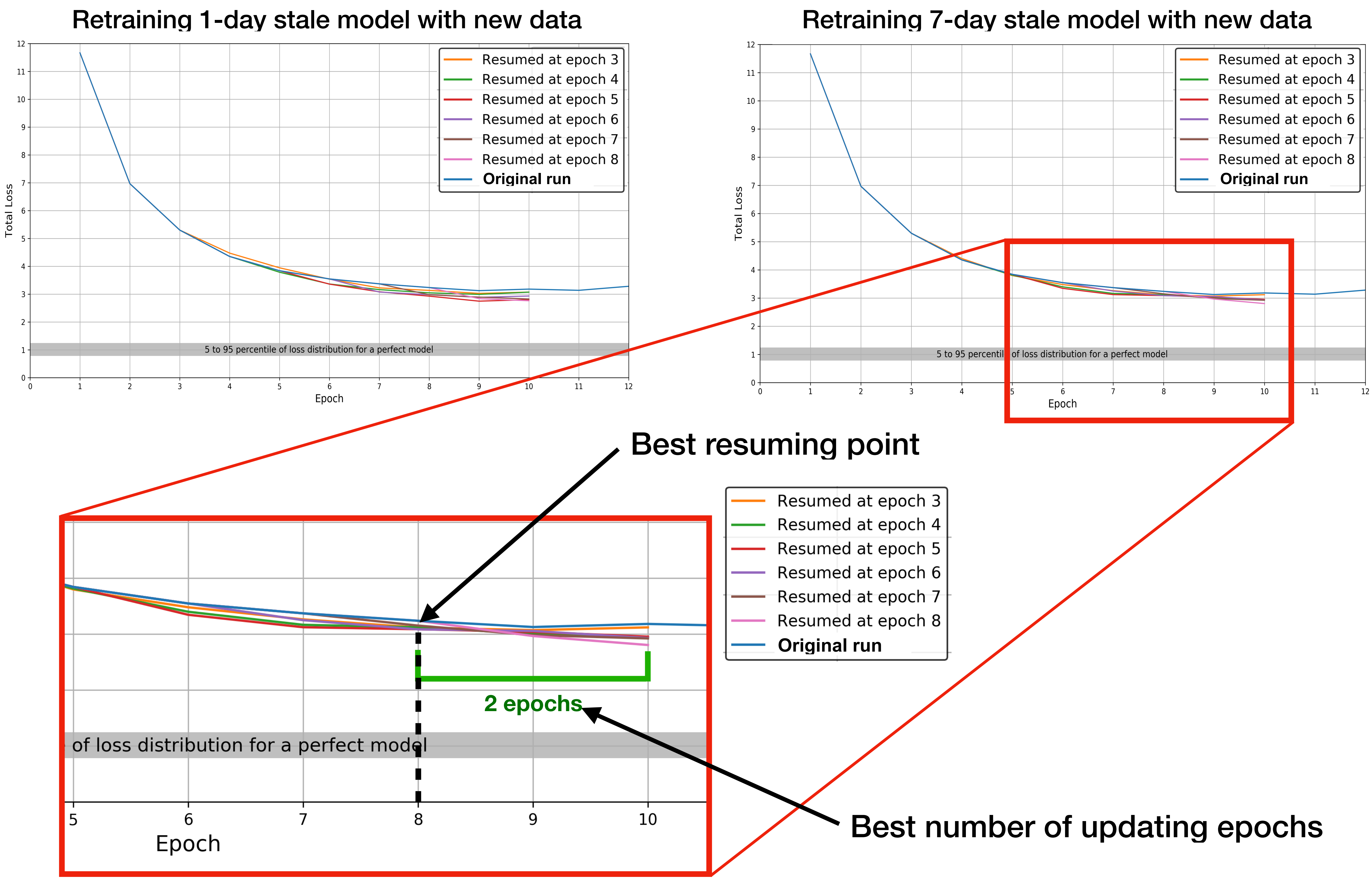


Exploring incremental training



The above branching for training updates is repeated for days 1 through 7

Results


Retraining 1-day stale model with new data
Retraining 7-day stale model with new data
Total Loss
Epoch
Resumed at epoch 3
Resumed at epoch 4
Resumed at epoch 5
Resumed at epoch 6
Resumed at epoch 7
Resumed at epoch 8
Original run
5 to 95 percentile of loss distribution for a perfect model
Best resuming point
2 epochs
of loss distribution for a perfect model
Best number of updating epochs

Doc2vec demo

Imports

In [1]:
```
import pandas as pd
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
```

Loading data

In [2]:
```
file_path = '../data/sample_data.csv'
df = pd.read_csv(file_path, names=['tag', 'stream'])
```

Manipulating input dataset

In [3]:
```
# Casting integer to string
df['tag'] = df['tag'].apply(lambda tag: str(tag))
# Loading input string and casting each element as string
df['stream'] = df['stream'].apply(lambda stream: [str(el) for el in eval(stream)])
```

In [4]:
```
# Visualizing input dataframe
df.head(3)
```

Out[4]:

	tag	stream
0	677994	[219345042, 172703514, 184153266, 56192185, 52...
1	767275	[96570421, 26516210, 50903853, 26516210, 27729...
2	786423	[187228547, 2791348, 35155700, 2791348, 351557...

Preparing training data

In [5]:
```
# Tagging each stream
tagged_streams = [TaggedDocument(stream, [tag])
                  for tag, stream in zip(list(df['tag']), list(df['stream']))]
```

Defining the model

In [6]:
```
# Initializing the model
d2v_model = Doc2Vec(min_count=2, window=5, vector_size=20, negative=10)
```

In [7]:
```
# Inizializing the vocabulary
d2v_model.build_vocab(tagged_streams)
```

Doc2vec demo

Training

In [8]:
```
d2v_model.train(tagged_streams, total_examples=d2v_model.corpus_count, epochs=10)
```

Extracting output

In [9]:
```
# Extracting list of items
item_ls = list(d2v_model.wv.vocab.keys())

# Extracting list of tags
tag_ls = d2v_model.docvecs.offset2doctag
```

In [10]:
```
# Retrieving embedding for each item
item_vector_ls = [d2v_model[item] for item in item_ls]

# Retrieving embedding for each tag
tag_vector_ls = [d2v_model.docvecs[tag_vect] for tag_vect in tag_ls]
```

In [11]:
```
# Printing a sample item and its embedding
print(item_ls[0])
print(item_vector_ls[0])
```

```
219345042
[-0.16691333  0.01391616  0.08182272  0.22406493 -0.13018888 -0.08907364
 -0.06540466  0.13903038  0.00235399 -0.0282561  -0.07210001  0.18075605
 -0.06638259  0.04872734 -0.30388135  0.17403638 -0.03945316  0.07126762
  0.03873265 -0.07725172]
```

In [12]:
```
# Printing a sample tag and its embedding
print(tag_ls[0])
print(tag_vector_ls[0])
```

```
677994
[-0.03200712 -0.06808201  0.09429213  0.20017318 -0.1556576   0.08129843
  0.01421014  0.02463668 -0.09386549  0.03395402  0.13053413  0.16646215
 -0.01707764  0.06892715 -0.35897225  0.20359874 -0.12954955  0.11253071
  0.01763733  0.10113616]
```