



APPLYING DEEP TRANSFER LEARNING FOR NLP

DIPANJAN (DJ) SARKAR

📅 13 - 16 November 2019

📍 Bengaluru



Slide Deck and Code

http://bit.ly/dtln_dhs19

About Me

Dipanjan Sarkar

Senior Data Scientist, Author, Google Developer Expert - ML



APPLIED
MATERIALS®



Machine Learning
GDE

Session Agenda



Transfer Learning Brief



Deep Transfer Learning for NLP



Modeling Approaches



Hands-on Session



Case-Study Review

Transfer Learning Brief



Transfer Learning

The ability to utilize knowledge learnt in prior tasks to new and novel tasks



- Know how to ride a motorbike → Learn how to ride a car
- Know how to play classic piano → Learn how to play jazz piano
- Know math and statistics → Learn machine learning

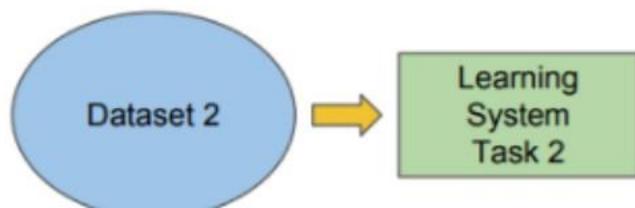
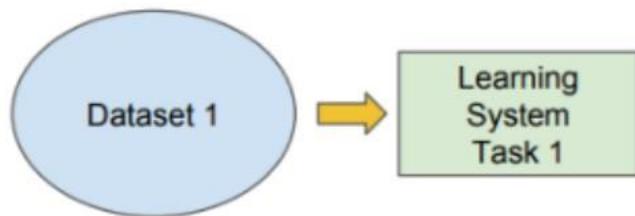
Traditional ML vs. Transfer Learning

Traditional ML

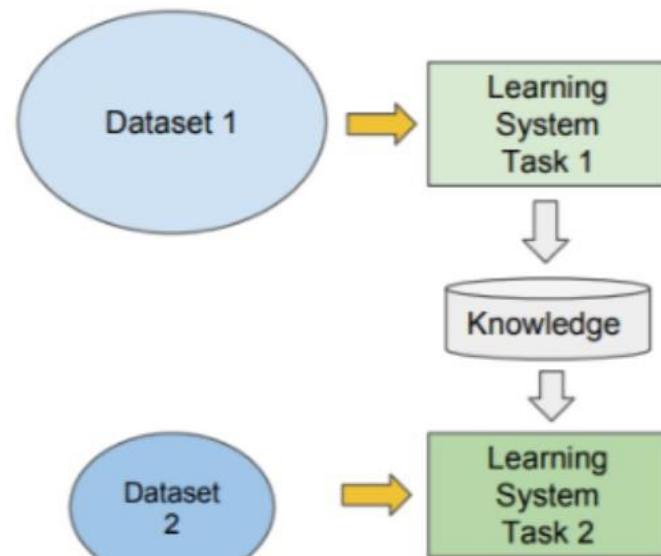
vs

Transfer Learning

- Isolated, single task learning:
 - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks



- Learning of a new tasks relies on the previous learned tasks:
 - Learning process can be faster, more accurate and/or need less training data



Defining Transfer Learning

A **Domain** consists of two components: $D = \{\mathcal{X}, P(X)\}$

- Feature space: \mathcal{X}
- Marginal distribution: $P(X)$, $X = \{x_1, \dots, x_n\}, x_i \in \mathcal{X}$

For a given domain **D**, a **Task** is defined by two components:

$$T = \{\mathcal{Y}, P(Y|X)\} = \{\mathcal{Y}, \eta\} \quad Y = \{y_1, \dots, y_n\}, y_i \in \mathcal{Y}$$

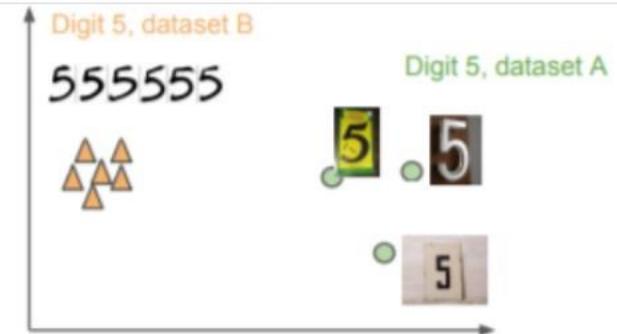
- A label space: \mathcal{Y}
- A predictive function η , learned from *feature vector/label* pairs, (x_i, y_i) , $x_i \in \mathcal{X}, y_i \in \mathcal{Y}$
- For each feature vector in the domain, η predicts its corresponding label: $\eta(x_i) = y_i$

Defining Transfer Learning

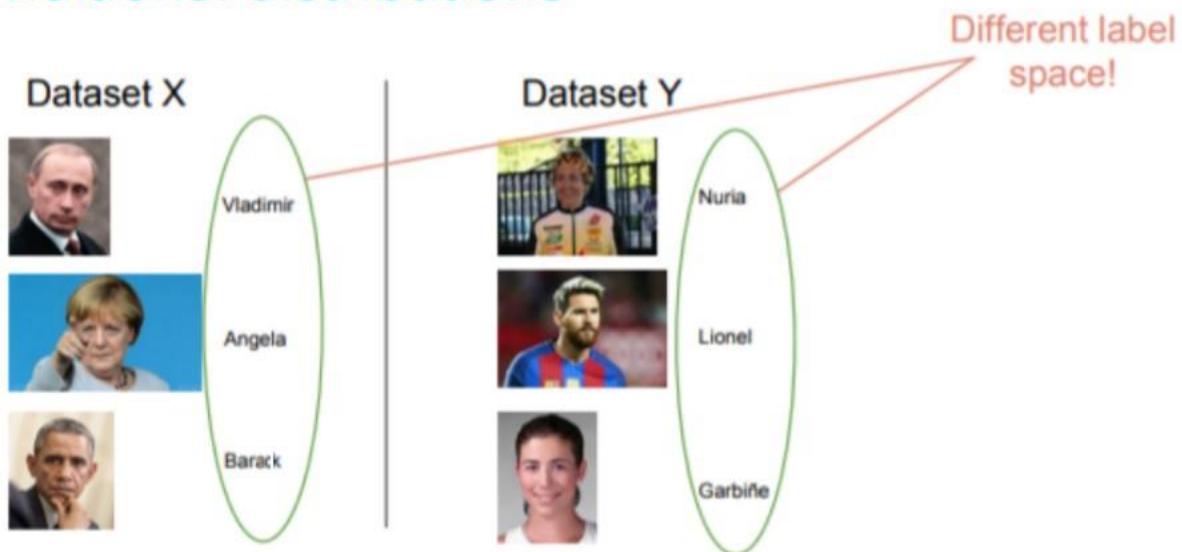
Given a source domain \mathcal{D}_S , a corresponding source task \mathcal{T}_S , as well as a target domain \mathcal{D}_T and a target task \mathcal{T}_T , the objective of transfer learning now is to enable us to learn the target conditional probability distribution $P(Y_T|X_T)$ in \mathcal{D}_T with the information gained from \mathcal{D}_S and \mathcal{T}_S where $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$. In most cases, a limited number of labeled target examples, which is exponentially smaller than the number of labeled source examples are assumed to be available.

Defining Transfer Learning

If two domains are different, they may have different **feature spaces** or different **marginal distributions**



If two tasks are different, they may have different **label spaces** or different **conditional distributions**

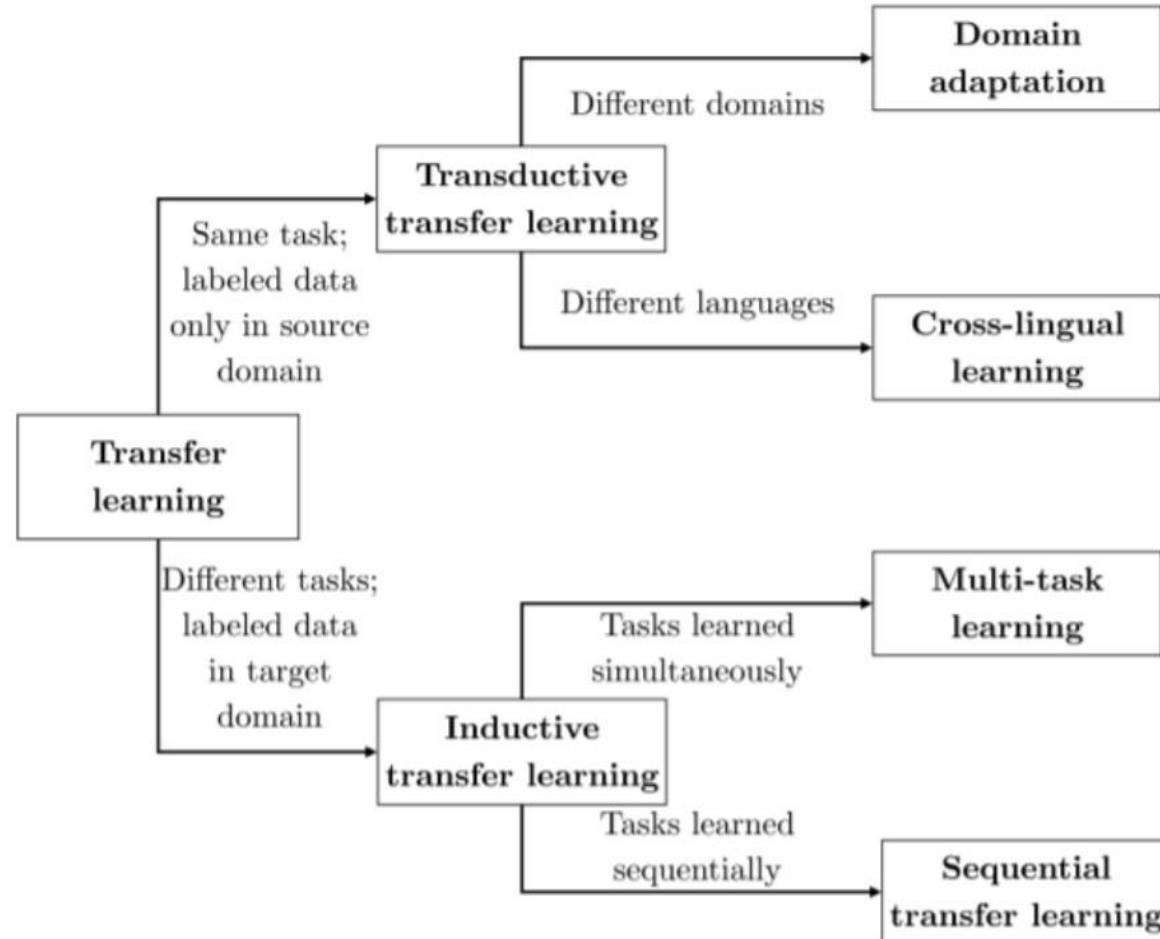


Transfer Learning Scenarios for NLP

Given source and target domains \mathcal{D}_S and \mathcal{D}_T where $\mathcal{D} = \{\mathcal{X}, P(X)\}$ and source and target tasks \mathcal{T}_S and \mathcal{T}_T where $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$ source and target conditions can vary in four ways, which we will illustrate in the following again using our document classification example:

1. $\mathcal{X}_S \neq \mathcal{X}_T$. The feature spaces of the source and target domain are different, e.g. the documents are written in two different languages. In the context of natural language processing, this is generally referred to as cross-lingual adaptation.
2. $P(X_S) \neq P(X_T)$. The marginal probability distributions of source and target domain are different, e.g. the documents discuss different topics. This scenario is generally known as domain adaptation.
3. $\mathcal{Y}_S \neq \mathcal{Y}_T$. The label spaces between the two tasks are different, e.g. documents need to be assigned different labels in the target task. In practice, this scenario usually occurs with scenario 4, as it is extremely rare for two different tasks to have different label spaces, but exactly the same conditional probability distributions.
4. $P(Y_S|X_S) \neq P(Y_T|X_T)$. The conditional probability distributions of the source and target tasks are different, e.g. source and target documents are unbalanced with regard to their classes. This scenario is quite common in practice and approaches such as over-sampling, under-sampling, or SMOTE are widely used

Transfer Learning Scenarios for NLP

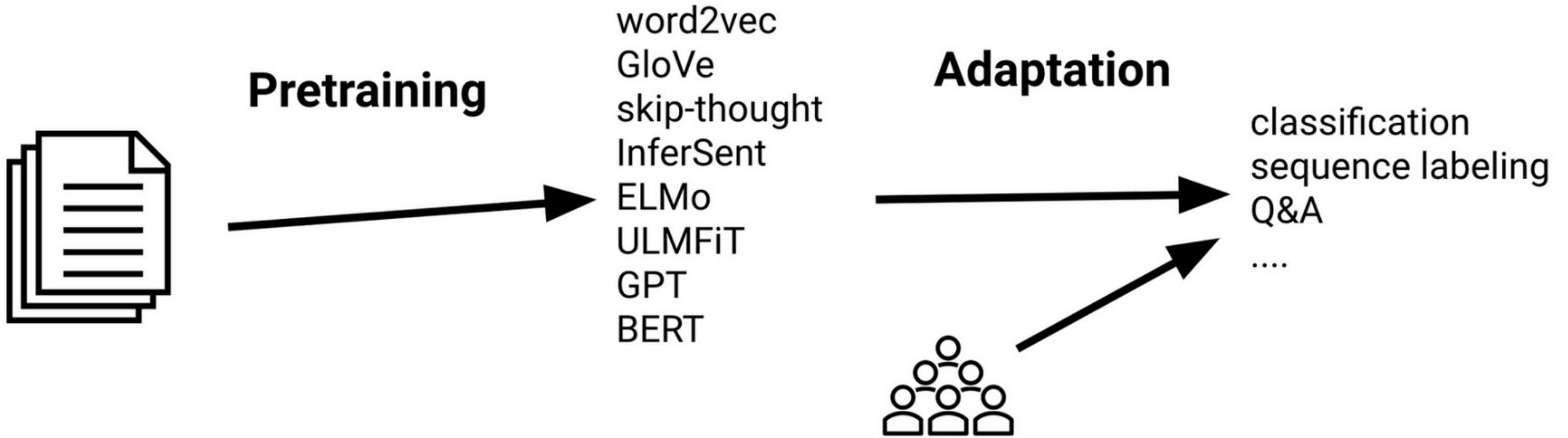


Deep Transfer Learning for NLP



Sequential Transfer Learning for NLP

Sequential transfer learning has led to the biggest improvements so far in the field of NLP



Sequential Transfer Learning for NLP

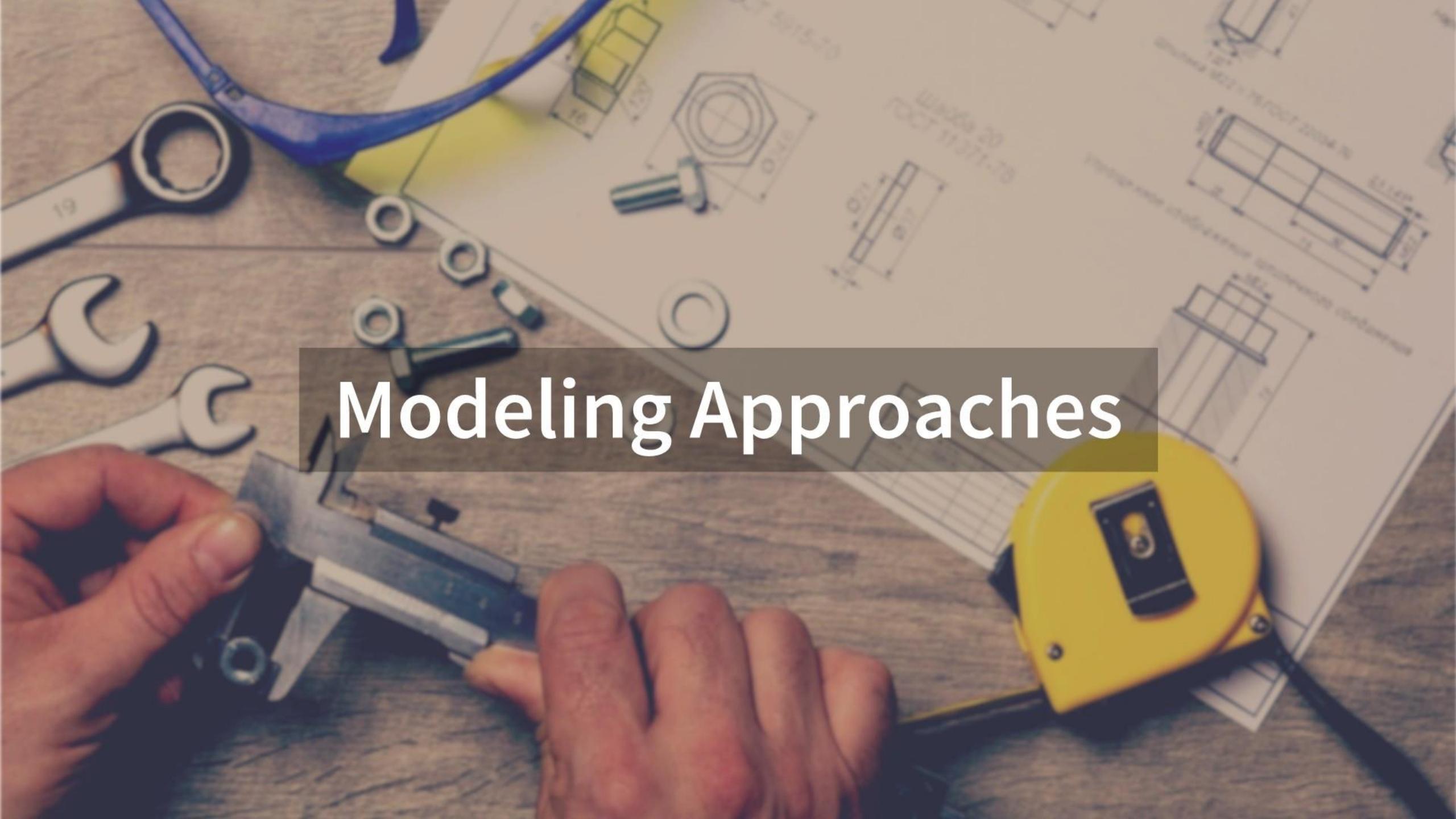
1 Pre-training

- Word and context-based representations (word2vec, fasttext, elmo)
- Simple Language Models (RNN, LSTMs, Bi-LSTMs)
- Generalized Language Modeling (ULMFiT, GPT, BERT)

2 Adaptation

- Keep the model architecture and weights unchanged
- Change and fine-tune model architecture and weights

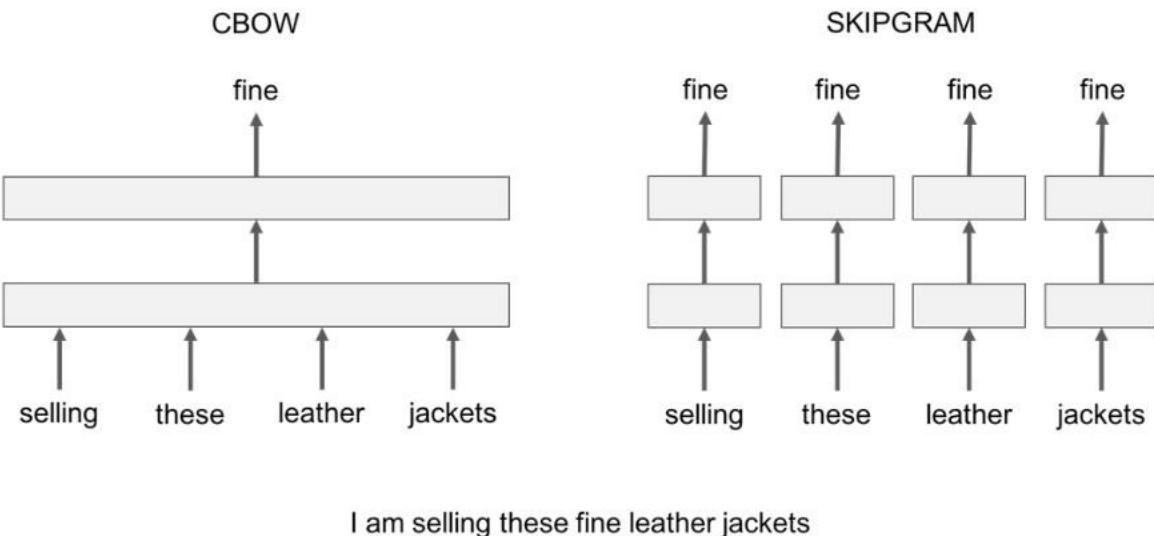
Modeling Approaches



Model Architectures

- ① Pre-trained FastText Embeddings + CNN
- ② Pre-trained FastText Embeddings + Bi-directional LSTMs + Attention
- ③ Universal Sentence Encoder
- ④ Neural Network Language Model
- ⑤ Bi-directional Encoder Representations from Transformers (BERT)
- ⑥ DistilBERT (Distillation of BERT)

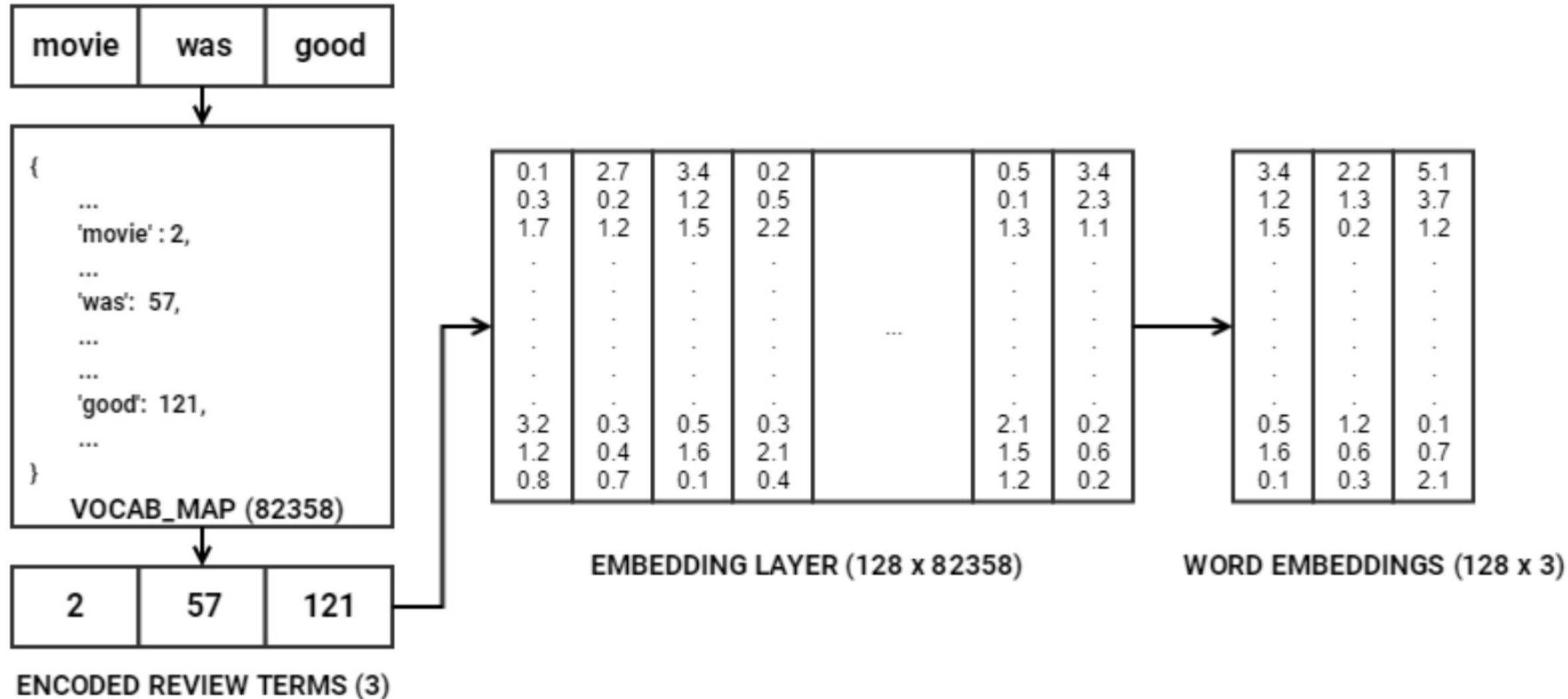
FastText Embeddings



- **FastText has similar training architectures as Word2Vec**
- **Considers each word as a bag of character n-grams**
- **Taking the word `where` and $n=3$ (tri-grams) as an example:**

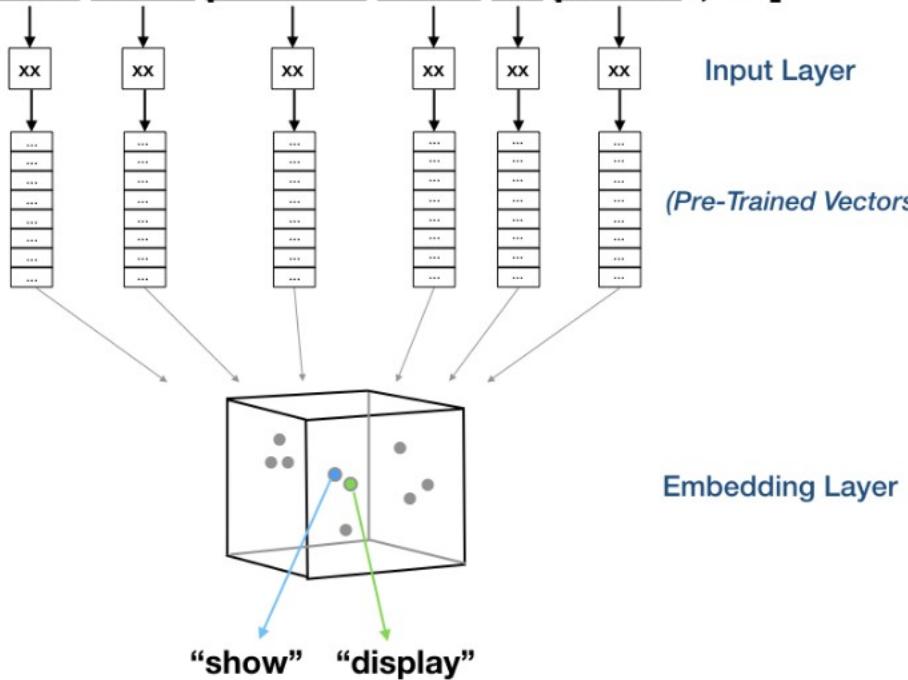
Word is represented by the character n-grams:
`<wh, whe, her, ere, re>` and the special sequence
`<where>` representing the whole word

Embedding Layer



Embedding Layer

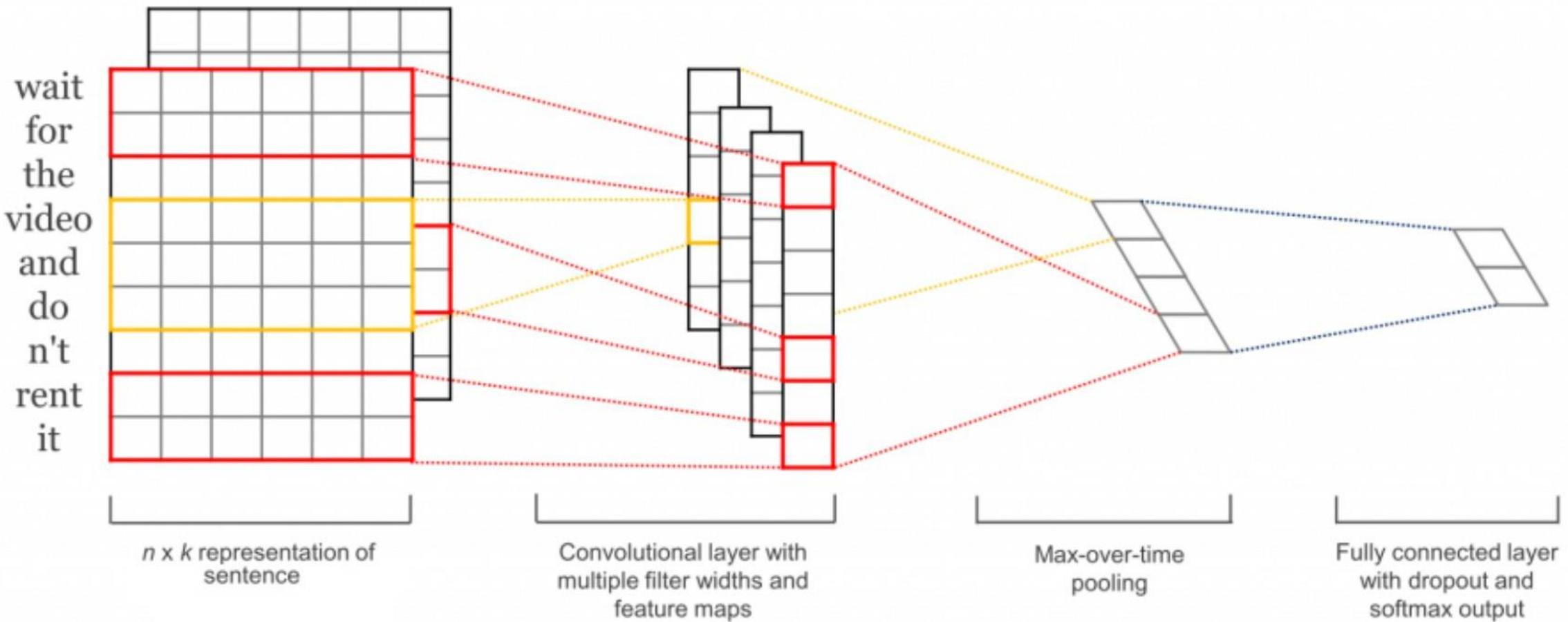
[“I want to search for blood pressure result history”,
“Show blood pressure result for patient”, ...]



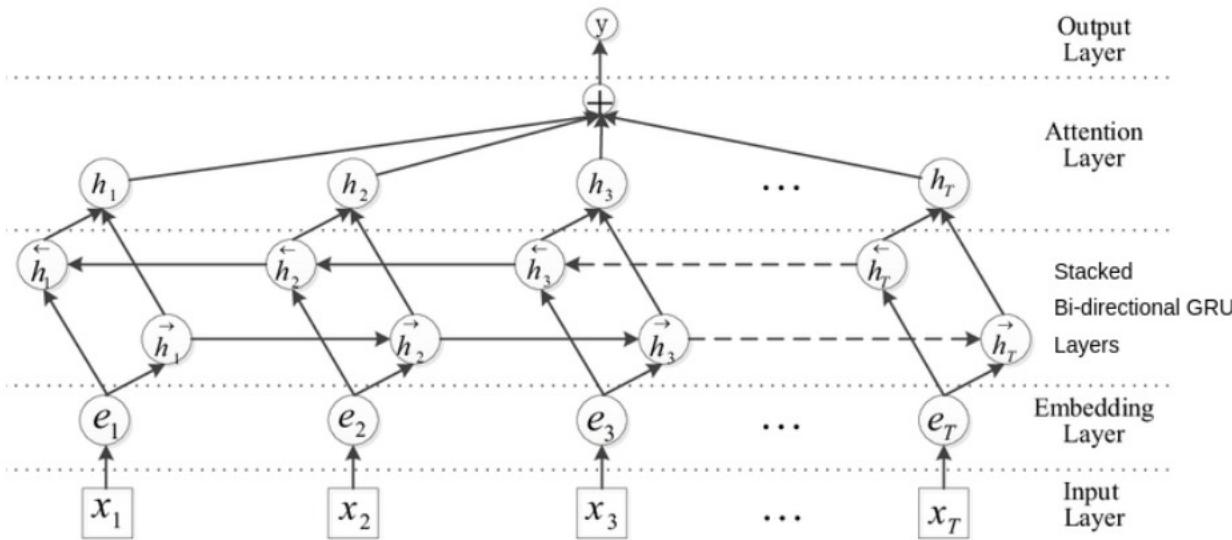
a	1
am	2
as	3
act	4
all	5
...	6
...	7
...	8
...	9
...	10
...	11
...	12
...	...
...	100
...	...
...	1000
...	...
...	10000
...	...
...	...

- **Don't initialize layer with random weights**
- **Initialize embedding layer with pre-trained embeddings**
- **Use pre-trained embeddings like FastText, Word2Vec, GloVe**
- **Model may perform better than random initialization**

Convolutional Neural Networks (CNN)

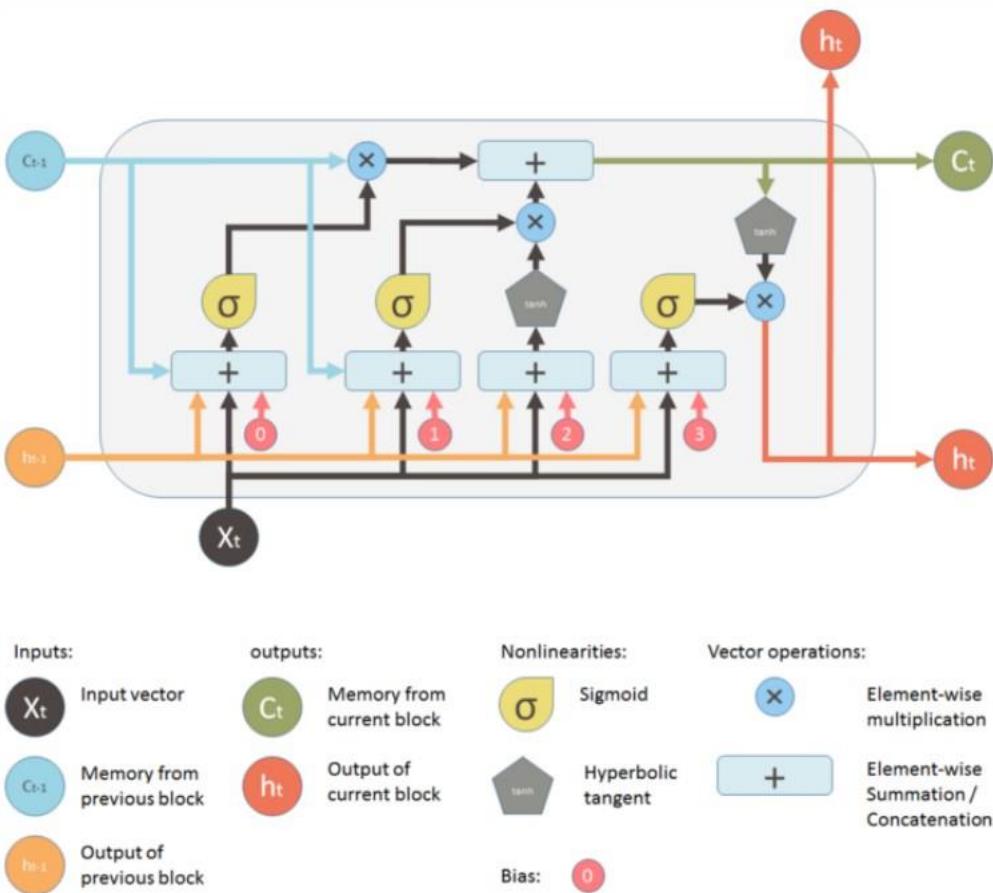


Bi-directional LSTMs + Attention



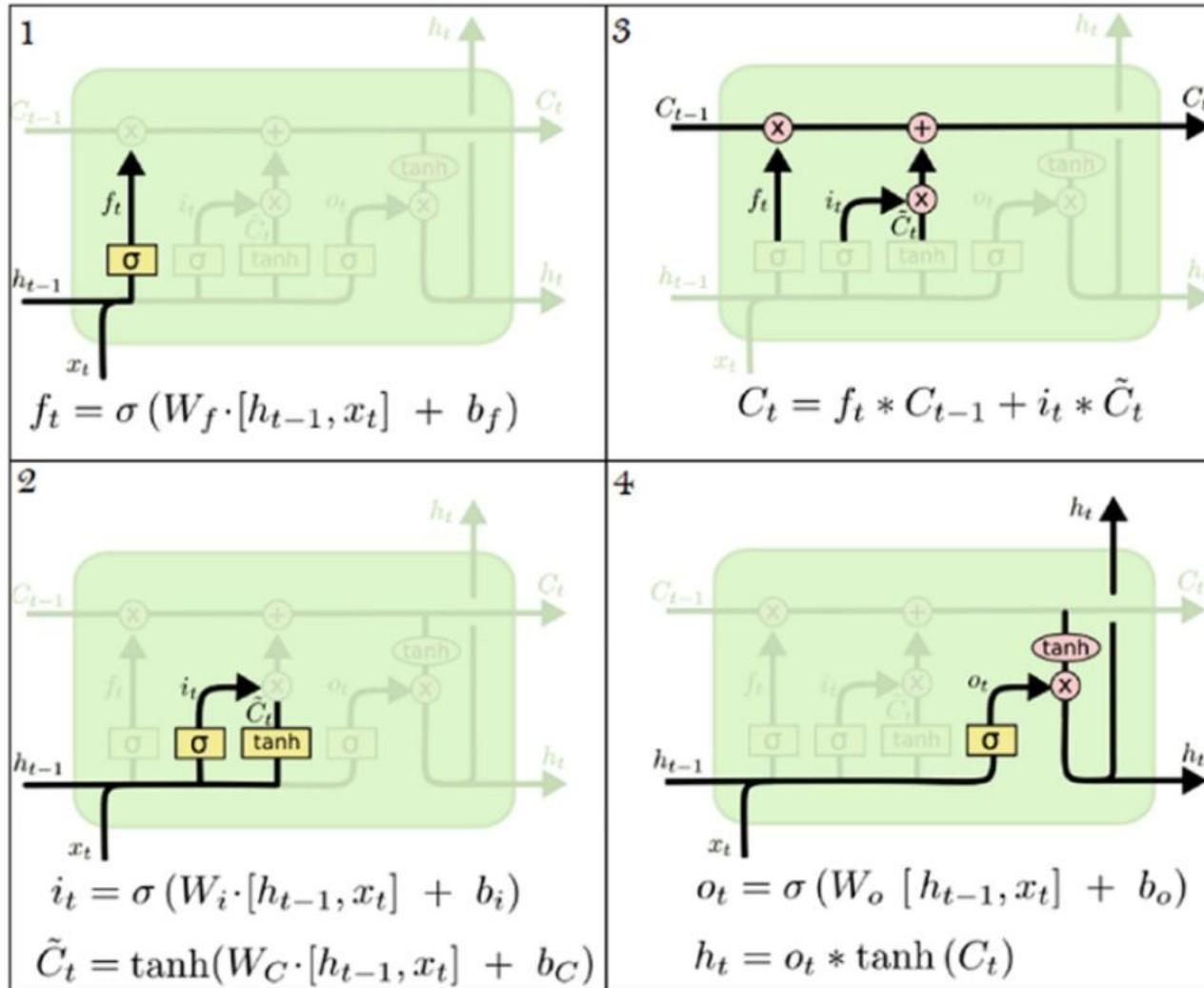
- **Embedding Layer populated with pre-trained embeddings**
 - FastText Embeddings
- **Bi-directional GRUs or LSTMs**
- **Global Attention Layer**
- **FC Dense Layers**

Sequential Models - LSTM



- LSTMs can remember longer sequences of data than RNNs
- The network takes three inputs
 - x_t is the input of the current time step
 - h_{t-1} is the output from the previous LSTM unit
 - c_{t-1} is the “memory” or cell state of the previous unit
- The forget gate decides what is relevant to retain from previous steps
- The input gate decides what information can be added from the current step
- The output gate decides what the next hidden state should be

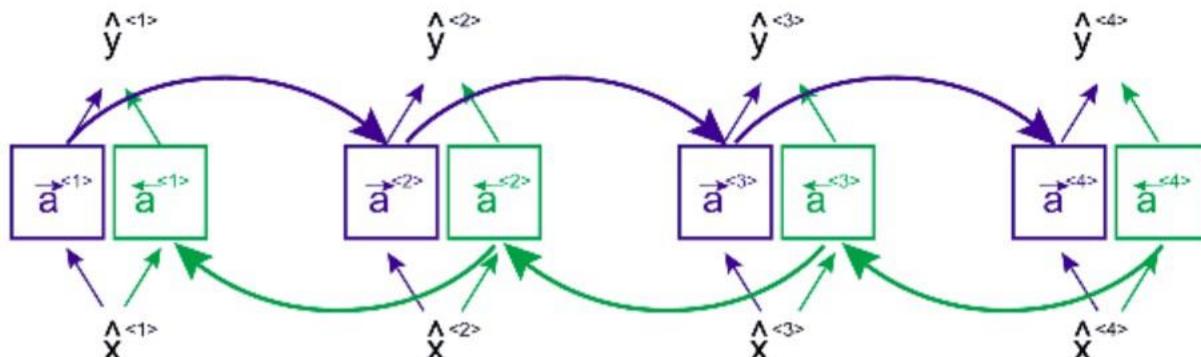
LSTM Cell Operations



The need for Bi-directional LSTMs

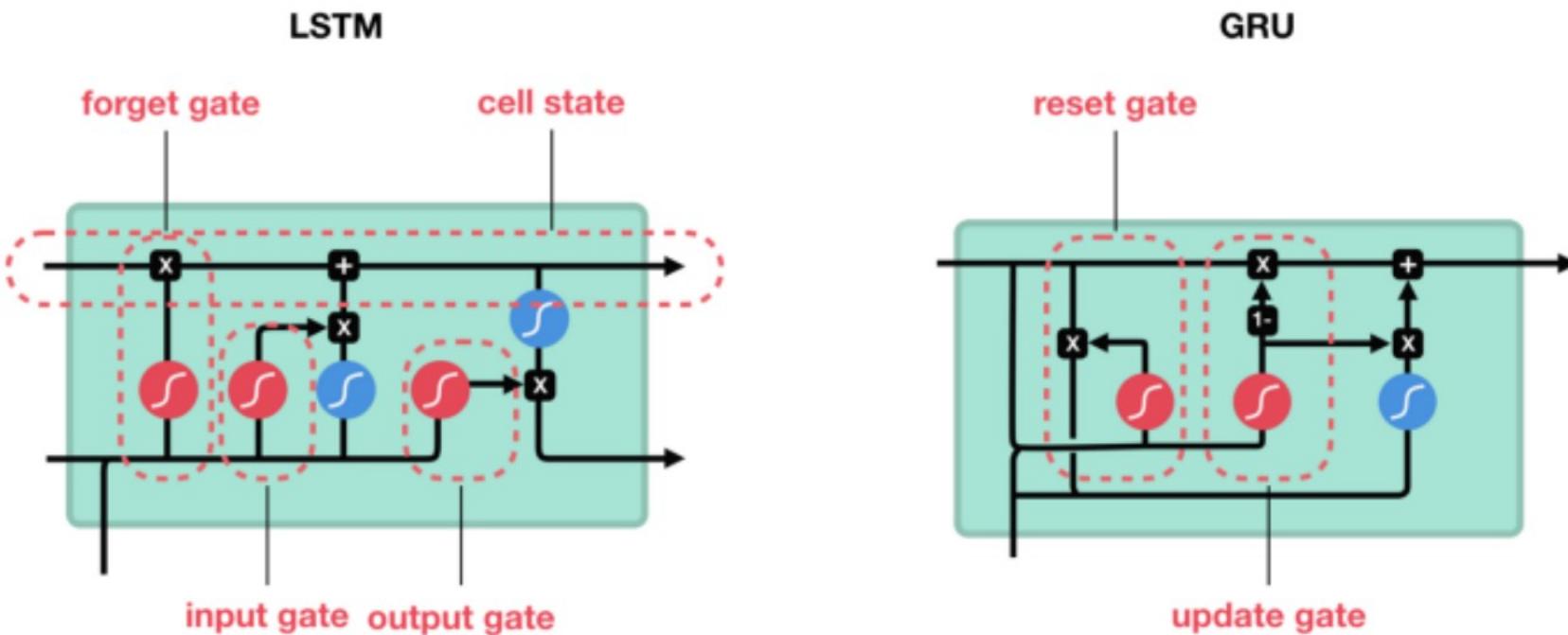
He said , "Teddy bears are on sale!"
not part of person name

He said , "Teddy Roosevelt was a great President !"
part of person name



- Bi-directional LSTMs are just putting two independent LSTMs together
- The input sequence is fed in forward order for one LSTM, and in reverse order for the other
- The outputs of the two networks are usually concatenated at each time step
- Preserving information from both past and future helps understand context better

LSTM vs GRU



sigmoid



tanh



pointwise
multiplication

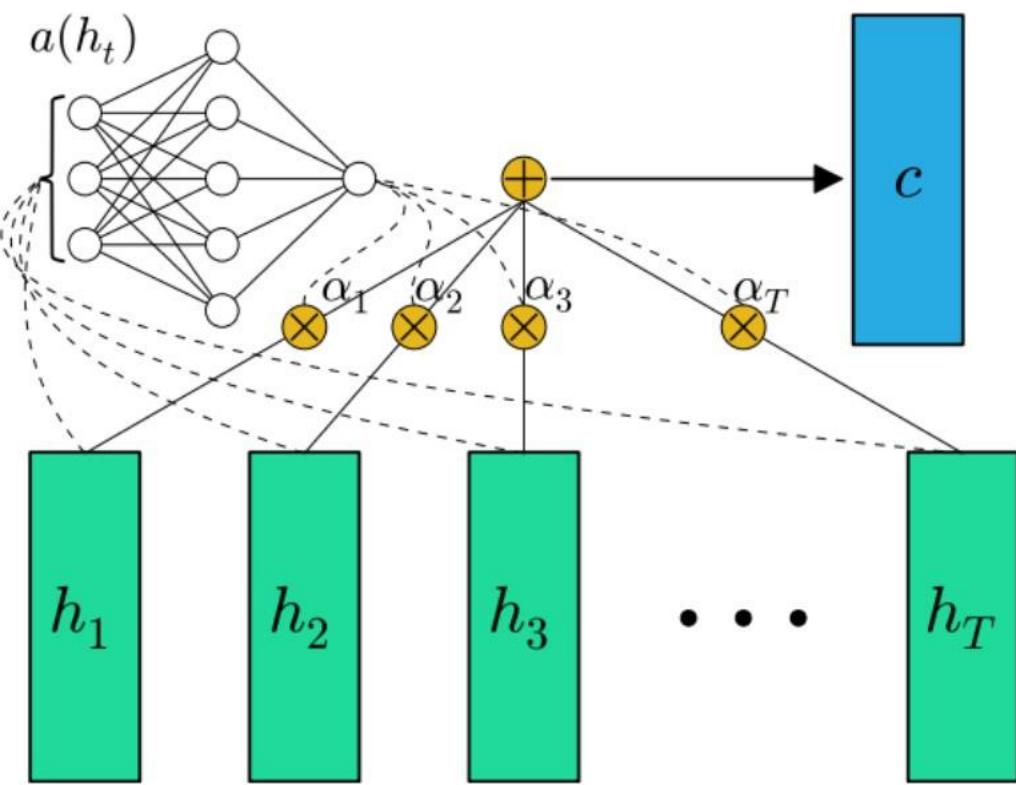


pointwise
addition



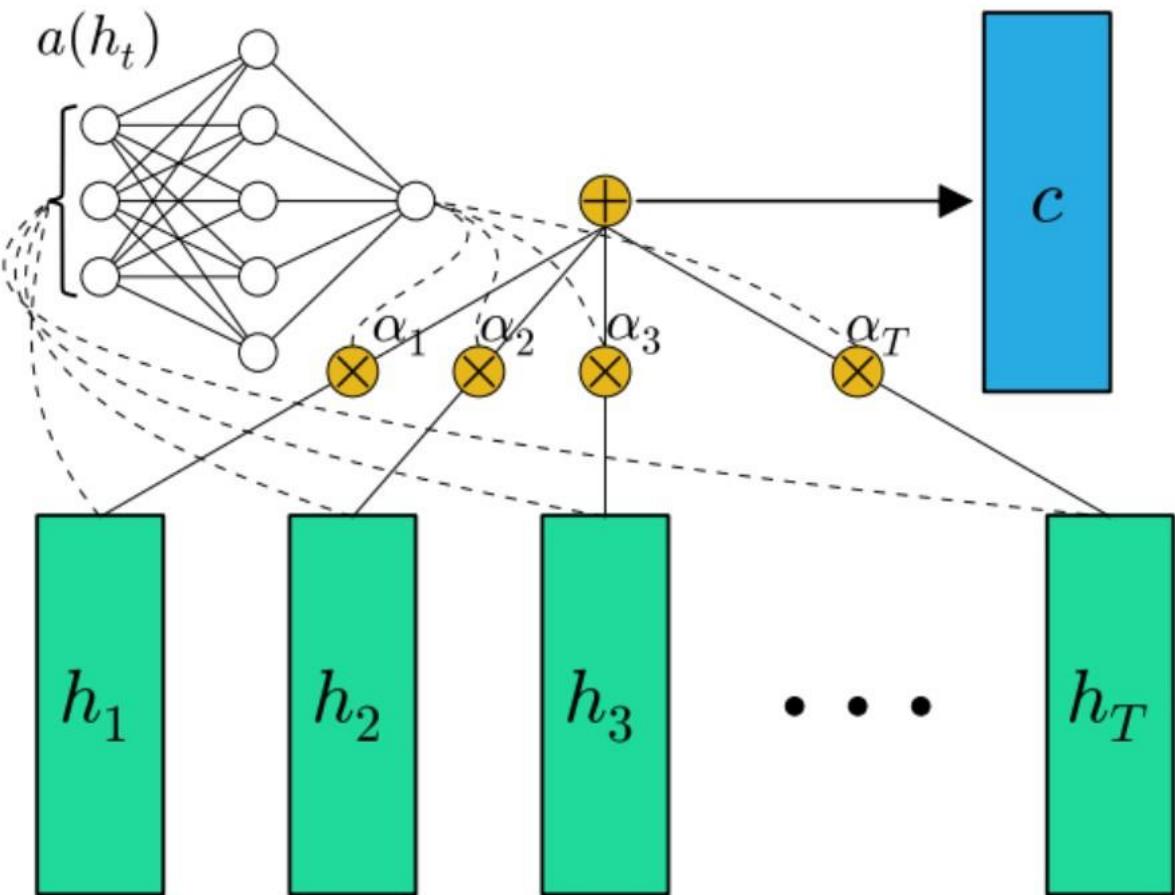
vector
concatenation

Attention please!



- Instead of using the output from the last GRU cell, send the entire sequence to a global attention layer
- Vectors from the hidden sequence h_t are fed into the learnable function $a(h_t)$
- Produces a probability vector α_t
- The final context vector c is a weighted average given by $\alpha_t \cdot h_t$

Attention please!



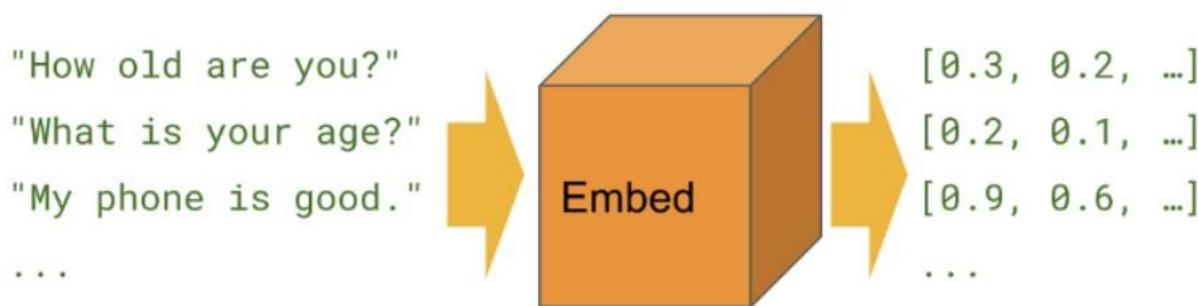
$$e_t = a(h_t) = \tanh(Wh_t + b)$$

$$\alpha_t = \text{softmax}(e_t) = \frac{\exp(e_t)}{\sum_{k=1}^T \exp(e_k)}$$

$$c = \sum_{t=1}^T \alpha_t h_t$$

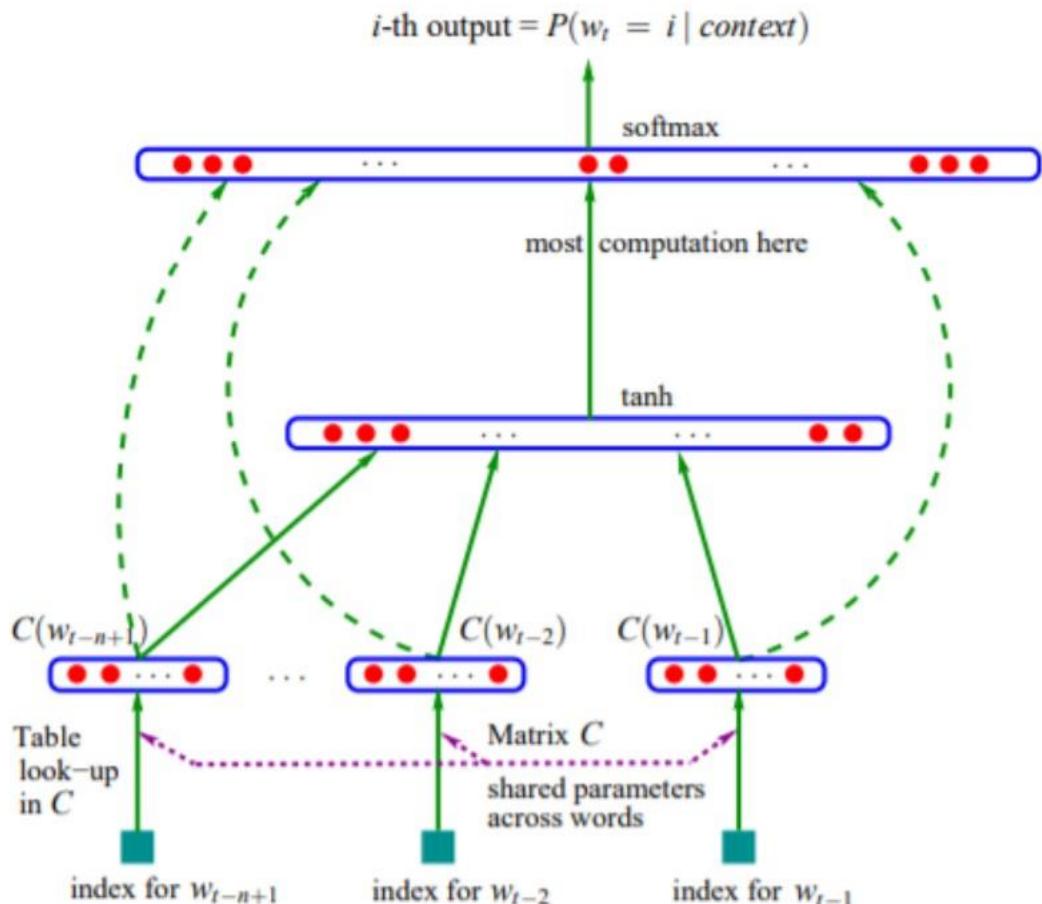
T is the total number of time steps in the input sequence

Universal Sentence Encoder (USE)



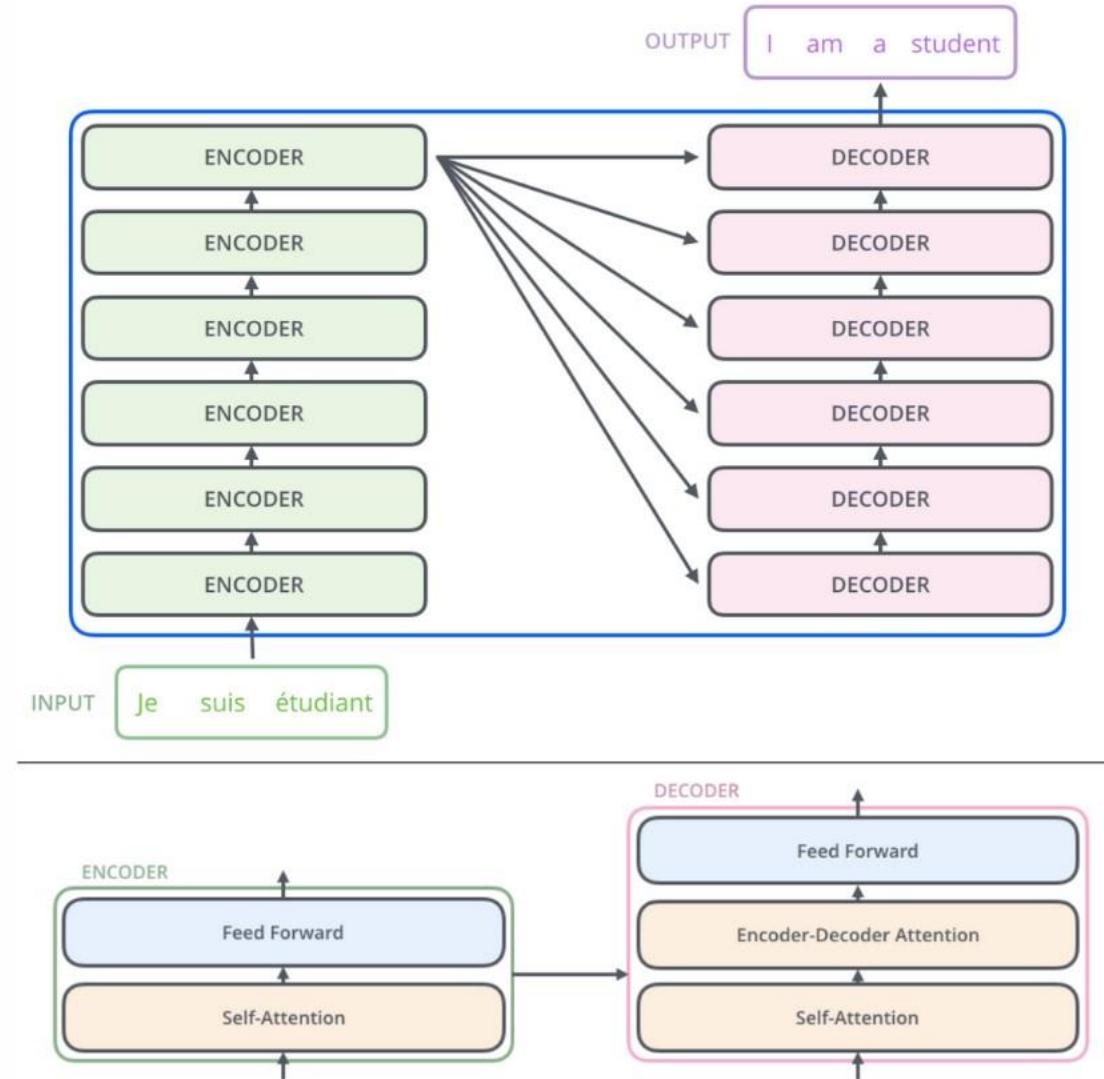
- USE encodes text of any length into fixed high dimensional vectors that can be used for various downstream tasks
- Pre-trained model has two architecture variants
 - Encoding sub-graph in Transformers
 - Deep Averaging Networks

Neural Network Language Model (NNLM)

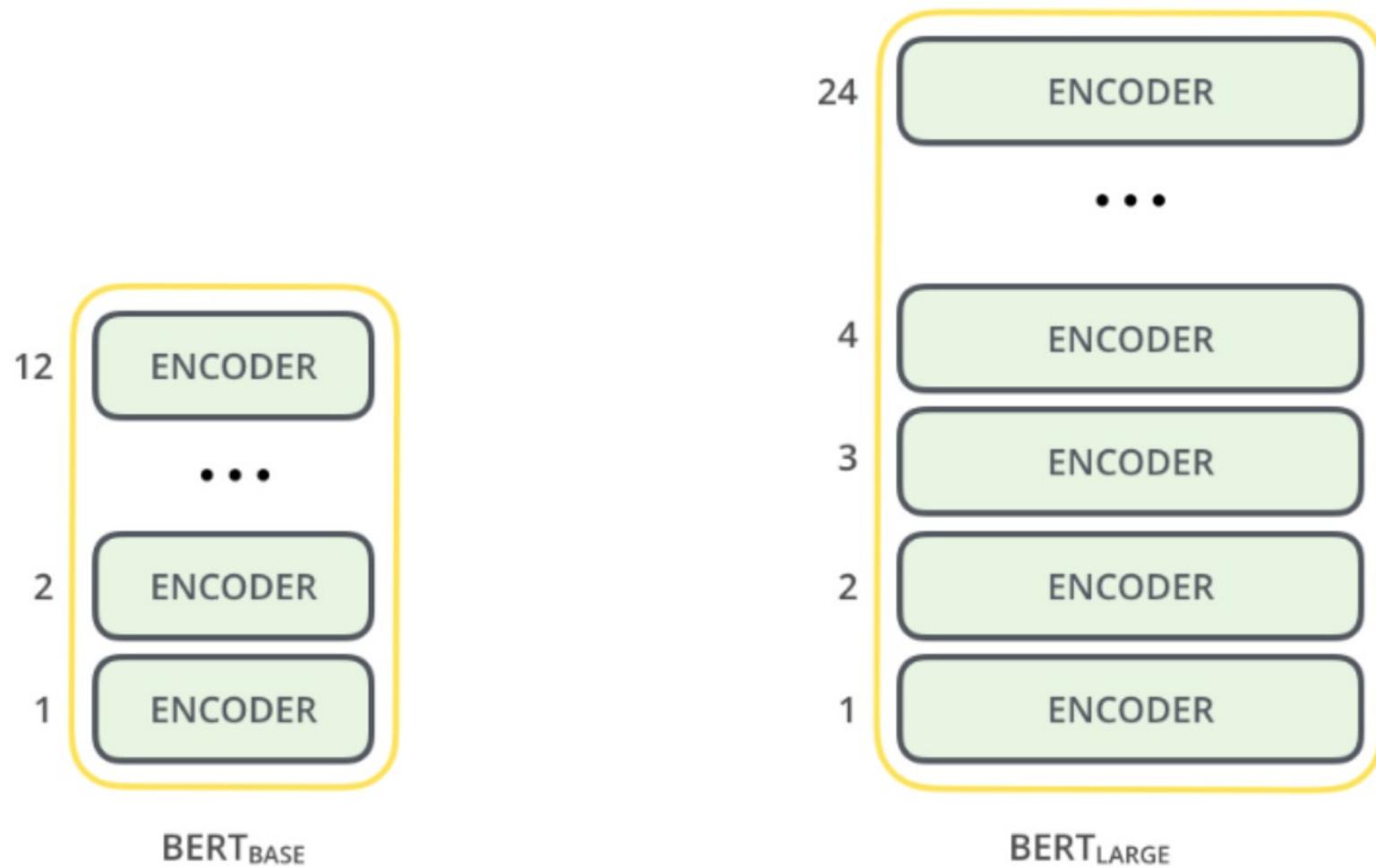


- NNLM based sentence encoder also gives fixed length vectors for any document
- Pre-trained model is a 3 hidden layer neural network trained based on next word prediction tasks
- Embedding layer in pre-training is shared across all words

Transformer Model



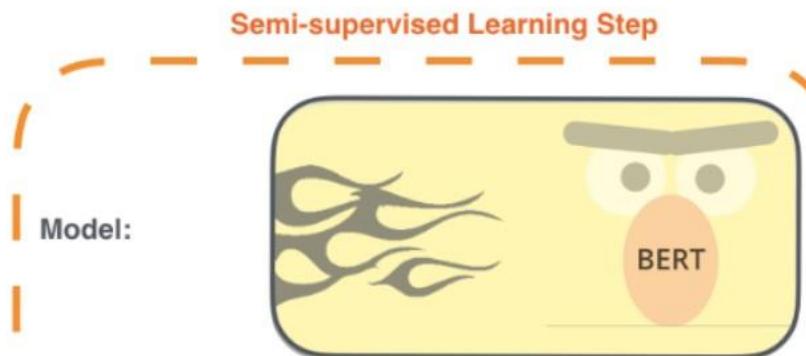
BERT - Trained Transformer Encoder Stack



Fine-tuning BERT

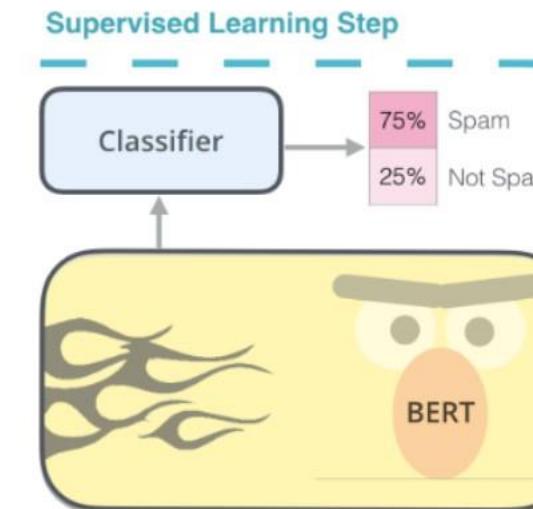
1 - Semi-supervised training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.



Objective:
Predict the masked word
(language modeling)

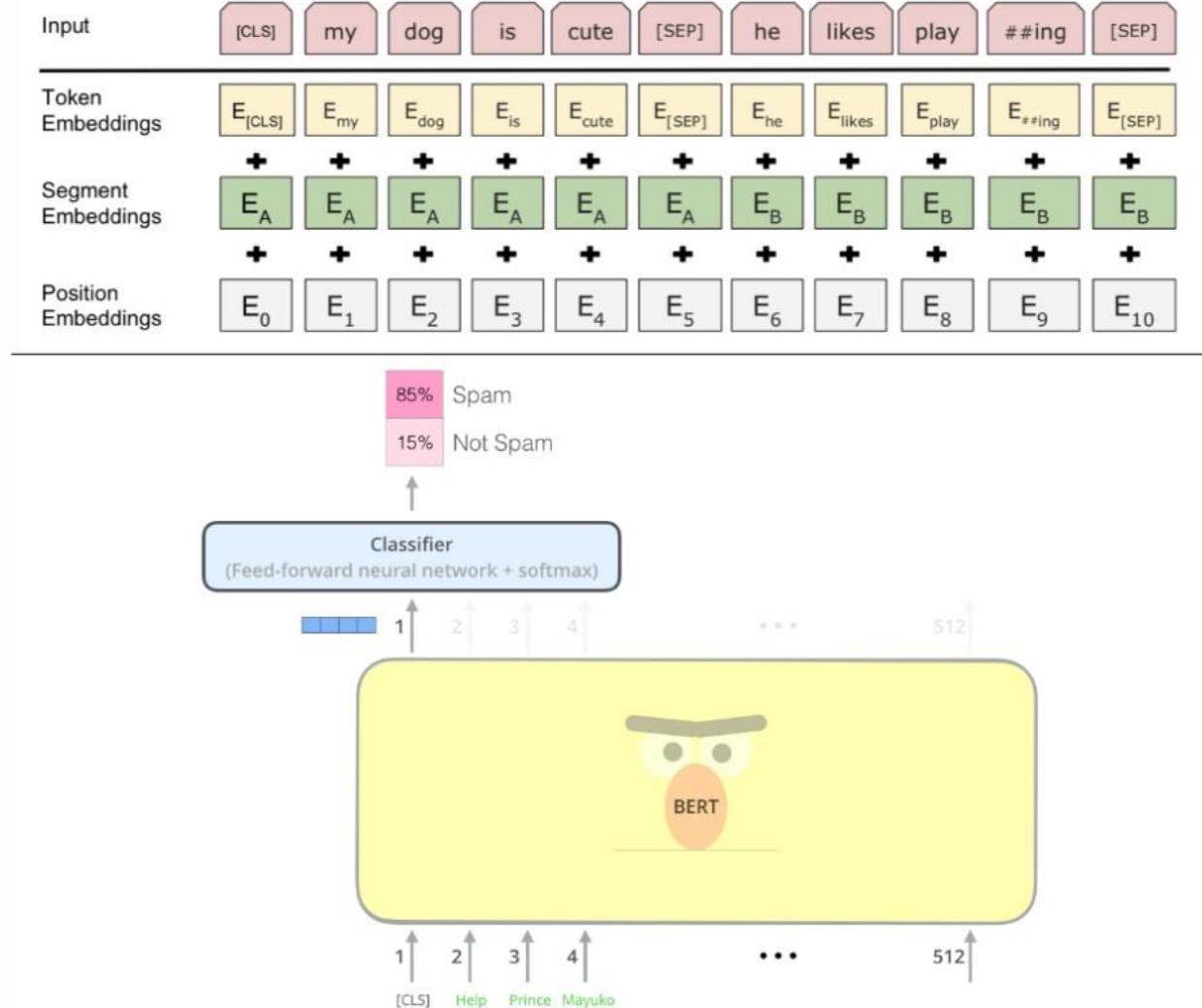
2 - Supervised training on a specific task with a labeled dataset.



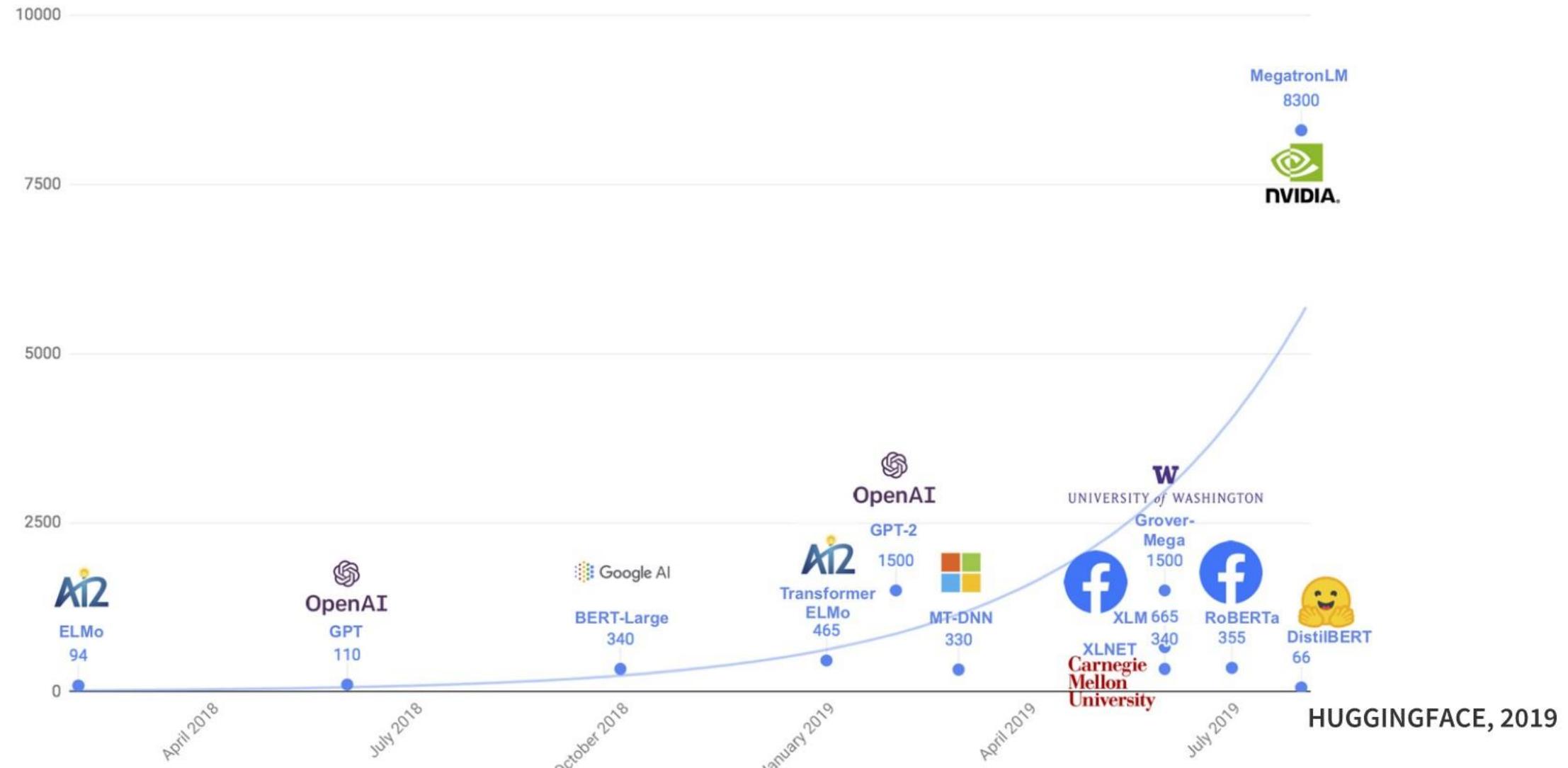
Dataset:

Email message	Class
Buy these pills	Spam
Win cash prizes	Spam
Dear Mr. Atreides, please find attached...	Not Spam

Fine-tuning BERT



Challenges with Massive Size of Pre-trained Models



DistilBERT - Compressing BERT

- **Compression technique known as knowledge distillation is used**
- **Teacher-Student Training is used - training student to mimic output distribution of teacher model (BERT)**

Trained with cross-entropy on soft targets (probabilities from teacher model)

- **HuggingFace uses KL-divergence loss to train DistilBERT**
- **Has almost halved the number of parameters from BERT and retains 95% of performance**

A photograph showing a close-up of a person's hands typing on a white Apple keyboard. The keyboard is positioned in front of a white iMac computer. The iMac screen displays a dark blue desktop background with several small, light-colored icons in the top right corner. The overall lighting is warm and focused on the hands and the keyboard.

Hands-on Session

Case Study: Pro-active Security & Vulnerability Dependency Analytics



Developers don't code everything from scratch.

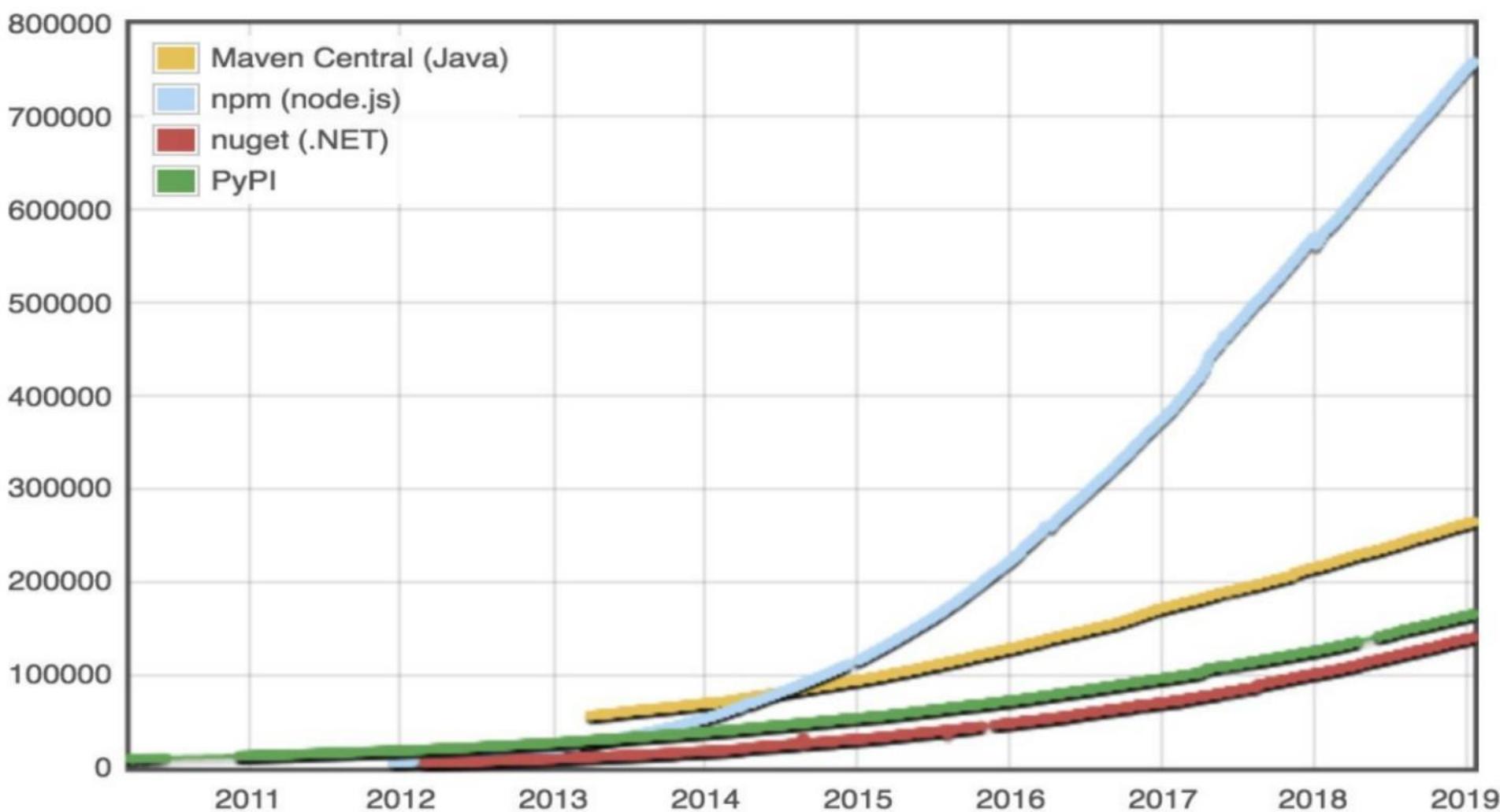
THEY "DEPEND" ON CODE WRITTEN BY OTHERS, THANK THE OPEN SOURCE SOFTWARE REVOLUTION

What are Dependencies?

- Open-source or commercial reusable libraries and frameworks
- Help build applications and software with ease
- Consists of features tackling specific problems or tasks

```
requirements.txt x
requirements.txt
 6  #
 7  absl-py==0.7.1          # via tensorflow
 8  arrow==0.13.2
 9  astor==0.8.0           # via tensorflow
10  beautifulsoup4==4.7.1
11  boto3==1.9.157
12  botocore==1.12.157
13  cachetools==3.1.1      # via google-auth
14  certifi==2019.3.9       # via requests
15  chardet==3.0.4          # via requests
16  contractions==0.0.18
17  daiquiri==1.5.0
18  dill==0.2.9
19  docutils==0.14          # via botocore
20  gast==0.2.2             # via tensorflow
21  google-api-core==1.11.1 # via google-cloud-bigquery, google-cloud-core
22  google-auth==1.6.3       # via google-api-core
23  google-cloud-bigquery==1.12.1
24  google-cloud-core==1.0.0 # via google-cloud-bigquery
25  google-resumable-media==0.3.2 # via google-cloud-bigquery
26  googleapis-common-protos==1.6.0 # via google-api-core
27  grpcio==1.21.1           # via tensorboard, tensorflow
28  h5py==2.9.0              # via keras, keras-applications
29  idna==2.8                # via requests
30  jmespath==0.9.4          # via boto3, botocore
31  keras-applications==1.0.7 # via keras, tensorflow
32  keras-preprocessing==1.0.9 # via keras, tensorflow
33  keras==2.2.4
34  lxml==4.3.3
35  markdown==3.1.1          # via tensorboard
36  numpy==1.16.4
37  pandas==0.24.2
```

Dependency Landscape - So many options!



Dependency Security Vulnerability Landscape

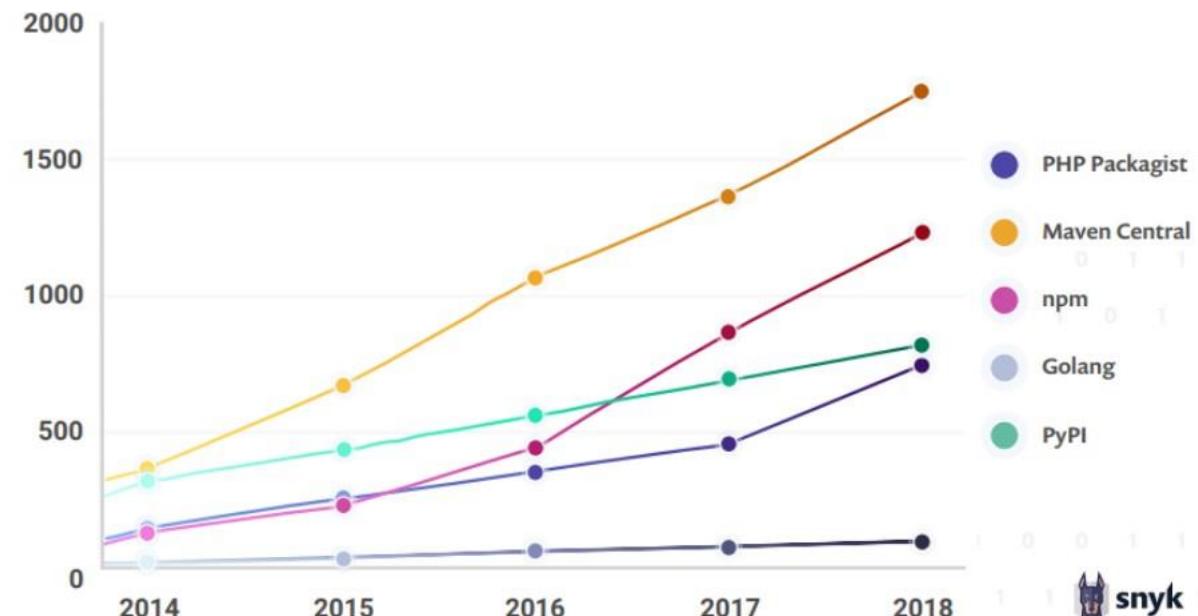


What are security vulnerabilities?

- Software security vulnerabilities are bugs or issues in the software which can lead to malicious attacks
- Commonly termed popularly as CVEs (Common Vulnerabilities and Exposures)
- A **vulnerability** is a mistake in software code that provides an attacker with direct access to a system or network
- An **exposure** is a mistake that gives an attacker indirect access to a system or network

Vulnerabilities in Application Dependencies

New vulnerabilities each year by ecosystem



- A definite increasing trend in vulnerabilities across all popular eco-systems
- Vulnerabilities if not identified, are often missed and not published, for more complex eco-systems like golang
- 43% increase in vulnerabilities in 2017 and a 33% increase in 2018

Limitations with Existing Vulnerability Analytics

How Do Maintainers Tell Users About Security Issues?

(RESPONDENTS WERE ABLE TO SELECT MULTIPLE RESPONSES)

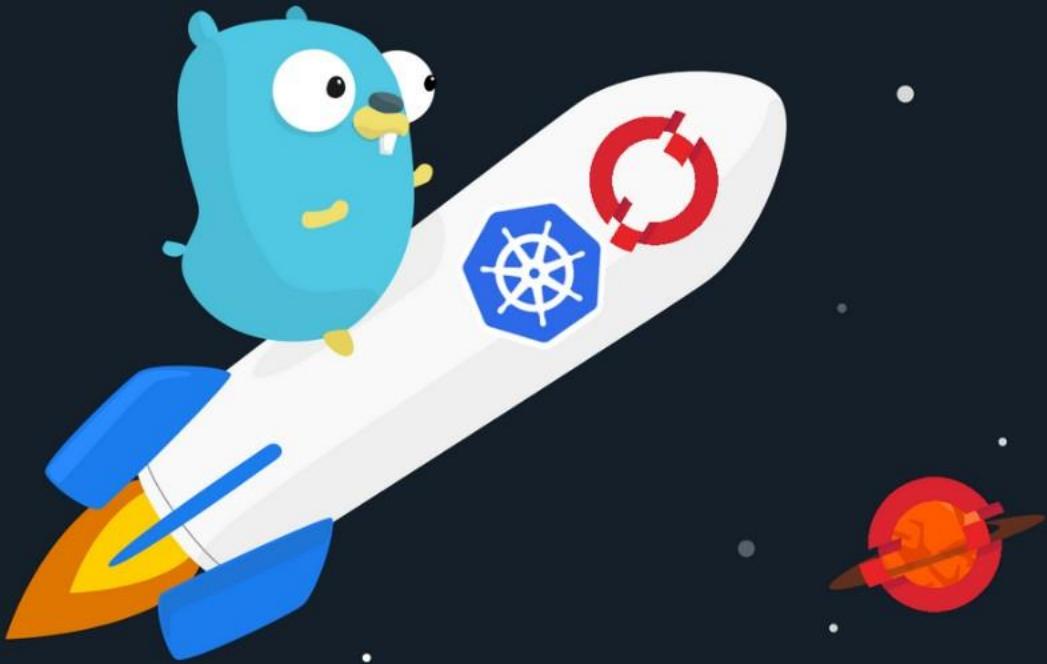


- Often vulnerabilities are present but not disclosed as CVEs
- Time elapsed between a vulnerability being present and a public disclosure can be long
- If a CVE is present, malicious users probably already know all about it
- The real risk is using dependencies which might have unreported vulnerabilities



Key Objectives

Key Objectives



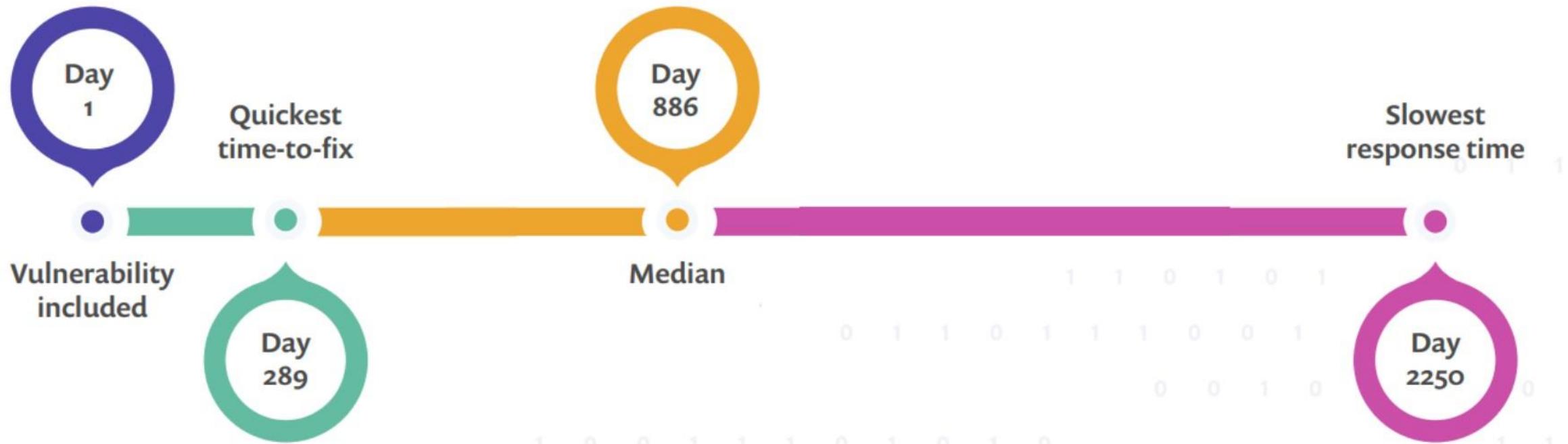
- Proactively predict probable vulnerabilities across direct & transitive dependencies in the golang - Kubernetes - OpenShift eco-system
- Integrate findings into developer and release processes of OpenShift in the long term



Business Significance

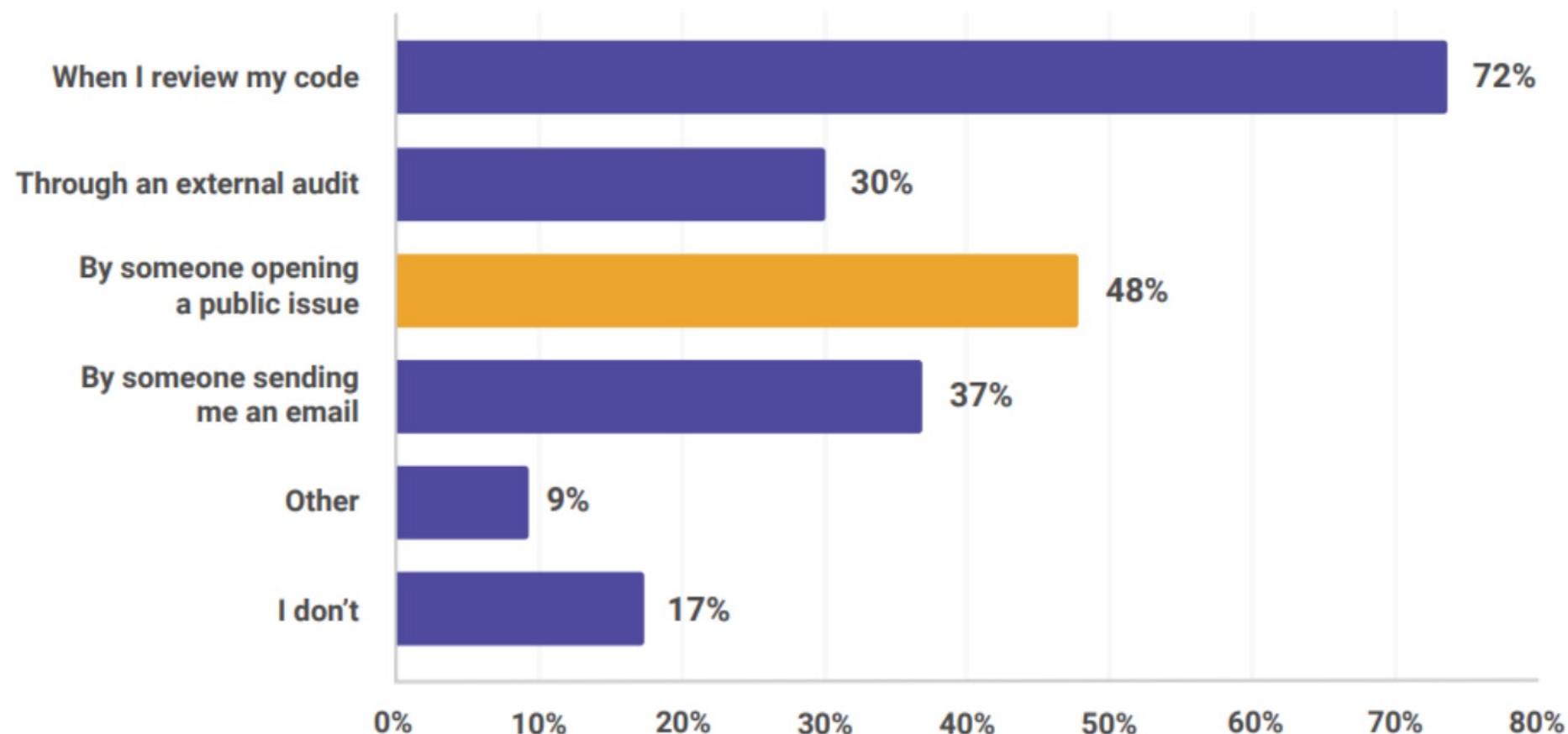
Vulnerabilities - Discovery to Disclosure

Vulnerabilities - days of inclusion to disclosure



Vulnerabilities - Public Data can be tapped for insights

How do maintainers find out about vulnerabilities?



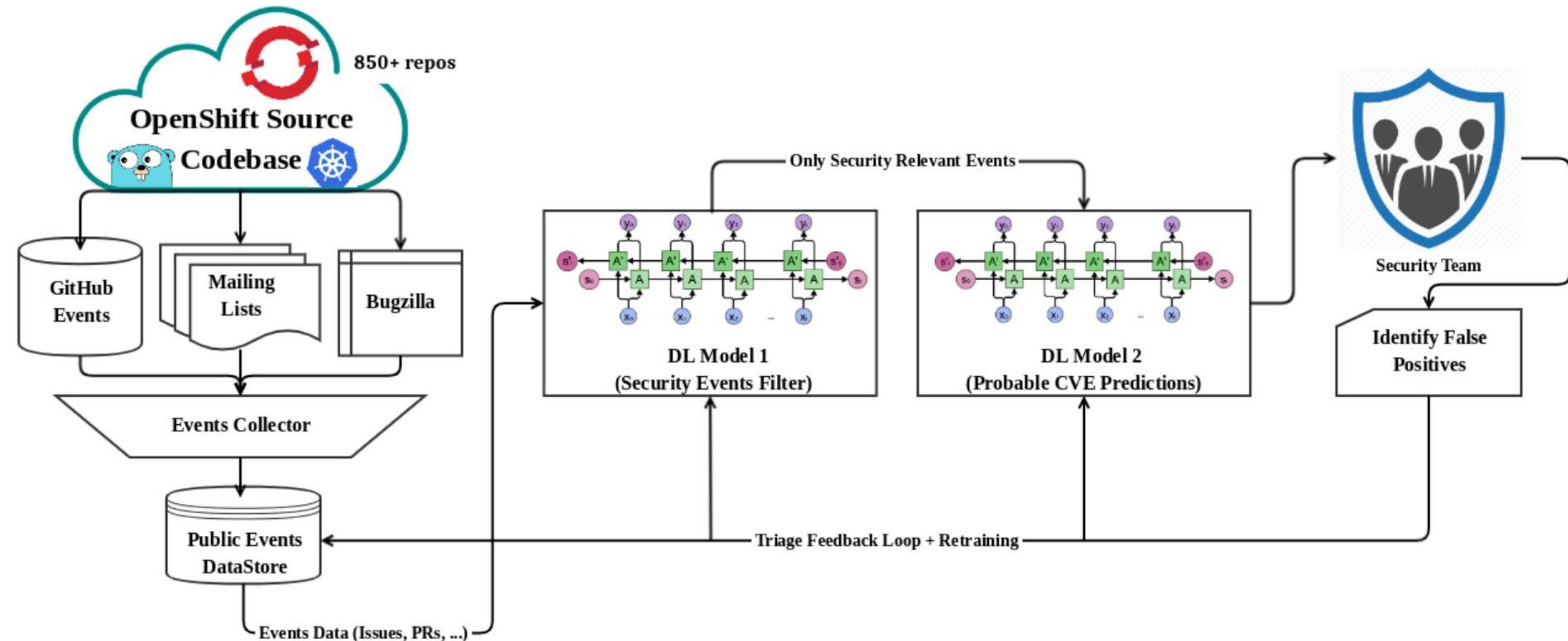
Business Significance - Current Market Space

- The golang eco-system is quite complex
- No mature package managements systems as compared to Python, Node, Java
- Security Tools and Platforms are not yet mature in the golang eco-system
- NVD feed is incomplete: Significant % of vulnerabilities (40-50%) would get initiated with a issue/PR
- Considerable time lag (days / weeks) can be observed between an identified vulnerability(issue/PR) and a created CVE

Vulnerability Detection Architecture



Solution Architecture - Overview



Proactive Vulnerability Detection Workflow

- Regularly monitor 850+ repositories \ dependencies being used by OpenShift
- Extract useful content from public events like issues, PRs, commits, comments, conversations
- DL Model 1 (Bi-directional GRU+Attention) filters out security irrelevant events which are not of concern
- DL Model 2 (Fine-tuned BERT) uses filtered data to predict events which could be about probable vulnerabilities
- Security Experts help us triage predictions and improve our models

A perspective view of a long, modern subway tunnel. The walls are made of light blue tiles, and there are yellow safety lines running along the floor and walls. The ceiling has a series of rectangular lights. In the distance, a train is visible at the end of the tunnel.

Deep Learning Model Architectures

Modeling Approaches

- **Baseline Machine Learning Models**

Linear, Trees, Ensembles, Model Stacking

- **Baseline Deep Learning Models**

DNNs, CNNs, LSTMs

- **Universal Language Model Fine-tuning (ULMFiT)**

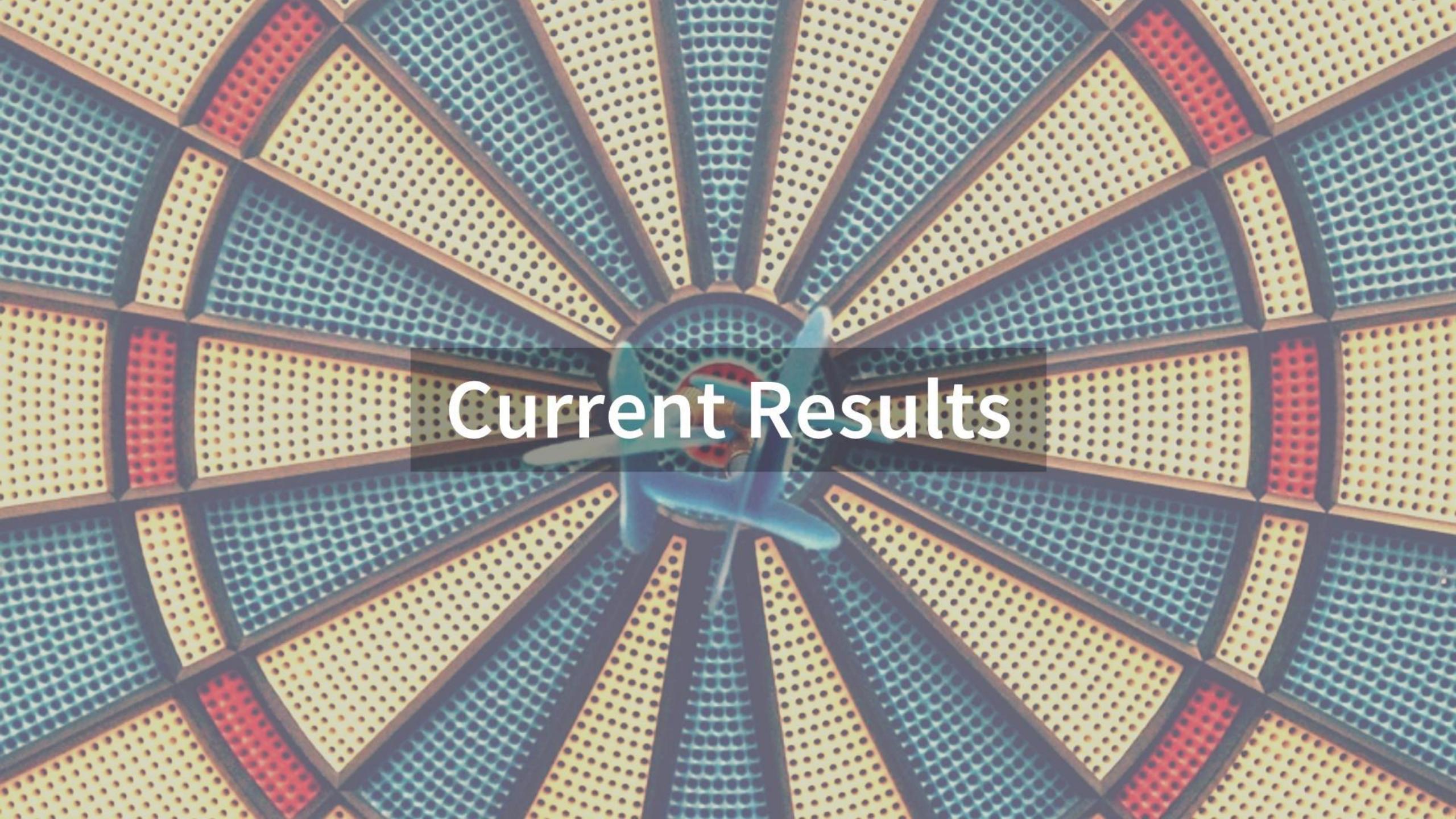
- **Embeddings from Language Models (ELMo)**

- **Universal Sentence Embeddings**

USE, NNLM

- **Bi-directional LSTMs\GRUs + Attention**

- **Bi-directional Encoder Representations from Transformers (BERT)**



Current Results

Model Performance Evaluation

- Dataset was focused on GitHub events from repositories which had a past history of known CVEs for the last 3 years [2016, 2017, 2018]
- Dataset had close to 200K+ events of interest and only 650+ CVEs (highly imbalanced)

Model Performance Evaluation Metrics

Performance Metric	Test Dataset Score (Bi-GRU)	Test Dataset Score (BERT)
% Security Issues correctly identified	94.00%	94.00%
% Probable Vulnerabilities correctly identified	70.0%	80.0%
% Probable Vulnerability predictions actually true	65.00%	71.00%
% Non-vulnerabilities correctly identified	99.0%	99.0%



THANK YOU

