Analytics Vidhya
Learn everything about analytics

DATAHACK SUMMIT 2019

APPLYING DEEP TRANSFER LEARNING FOR NLP

DIPANJAN (DJ) SARKAR

13 - 16 November 2019 Bengaluru

Slides and Code

TBA

Session Agenda

Transfer Learning Brief

Deep Transfer Learning for NLP

Modeling Approaches

Hands-on Session

Case-Study Review

Transfer Learning Brief

Transfer Learning

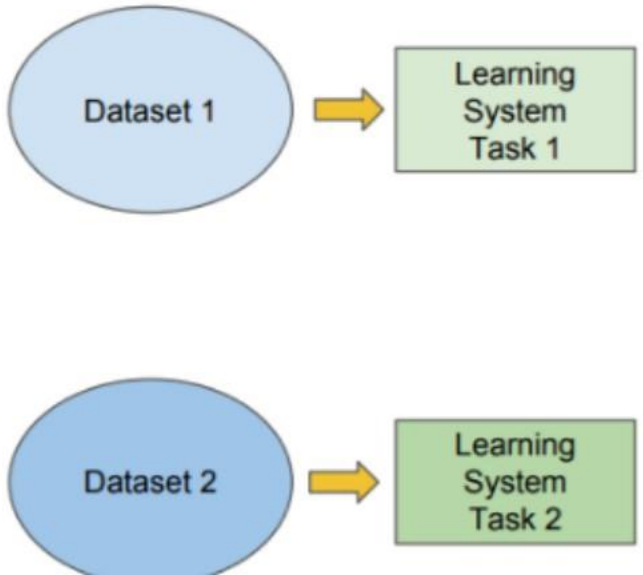
The ability to utilize knowledge learnt in prior tasks to new and novel tasks is Transfer Learning

- **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

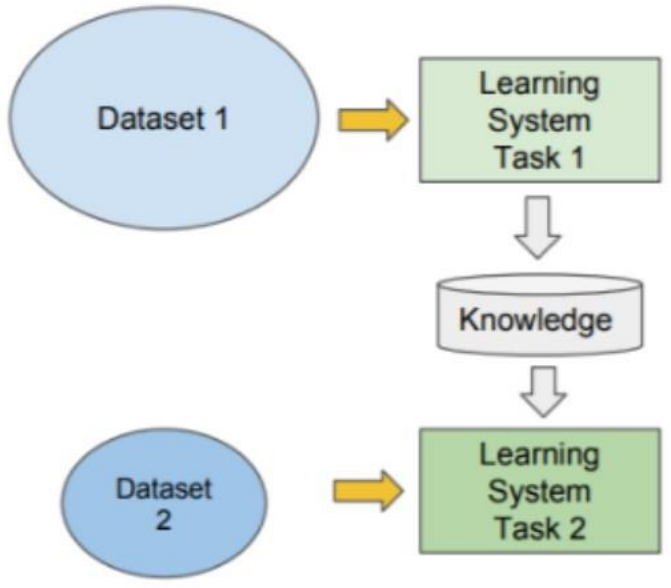
- Isolated, single task learning:
  - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks



vs

Transfer Learning

- 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: $P(X),\ X = \{x_1, \cdots, x_n\}, x_i \in \mathcal{X}$

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

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

- 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: $\eta(x_i) = y_i$

RUDER, 2017

Defining Transfer Learning

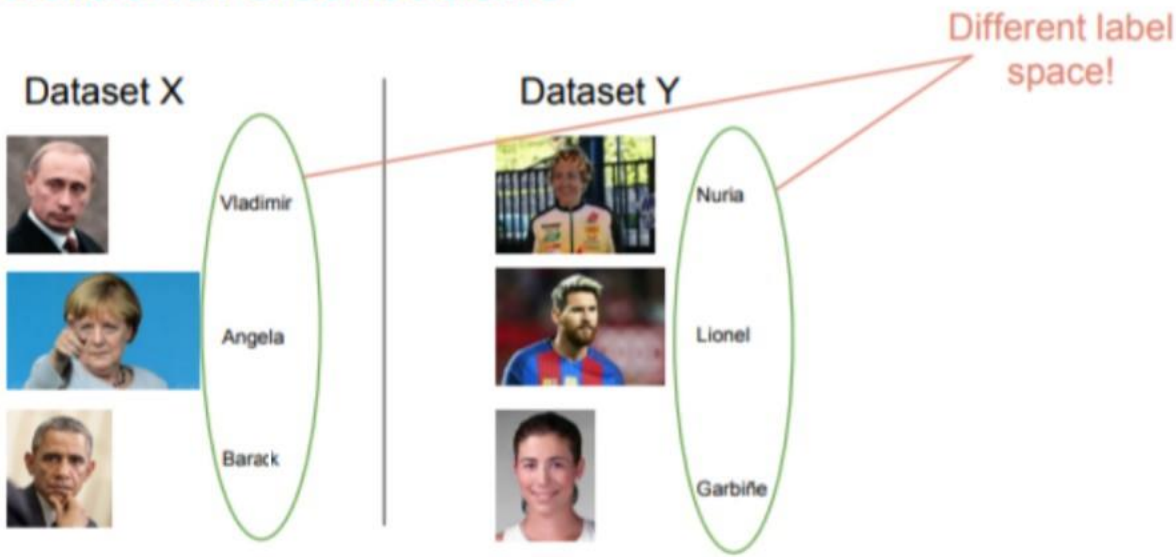
Given a source domain $\mathcal{D}_S$, a corresponding source task $\mathcal{T}_S$, as well as a target domain $\mathcal{D}_T$ and a target task $\mathcal{T}_T$, the objective of transfer learning now is to enable us to learn the target conditional probability distribution $P(Y_T|X_T)$ in $\mathcal{D}_T$ 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.

RUDER, 2017

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



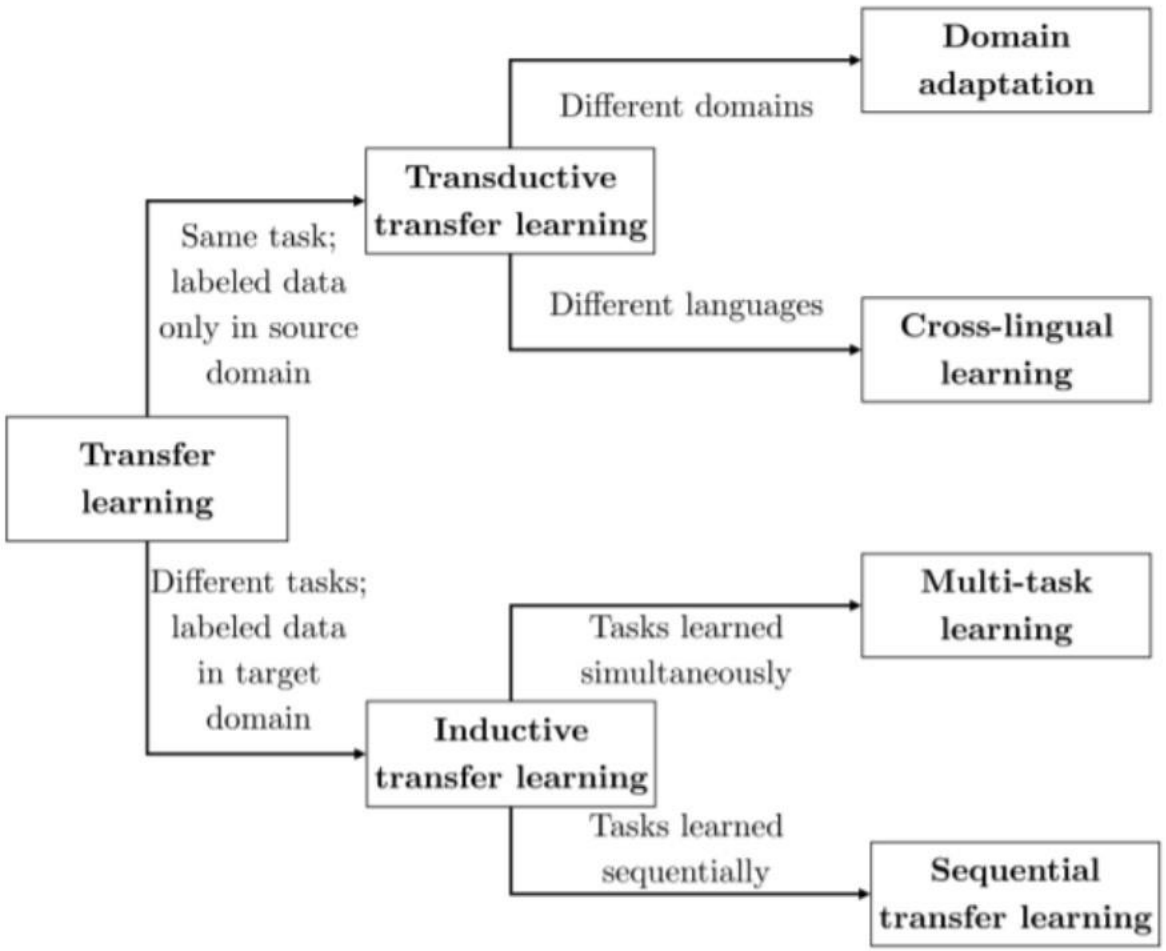
Transfer Learning Scenarios for NLP

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

RUDER, 2017

Transfer Learning Scenarios for NLP

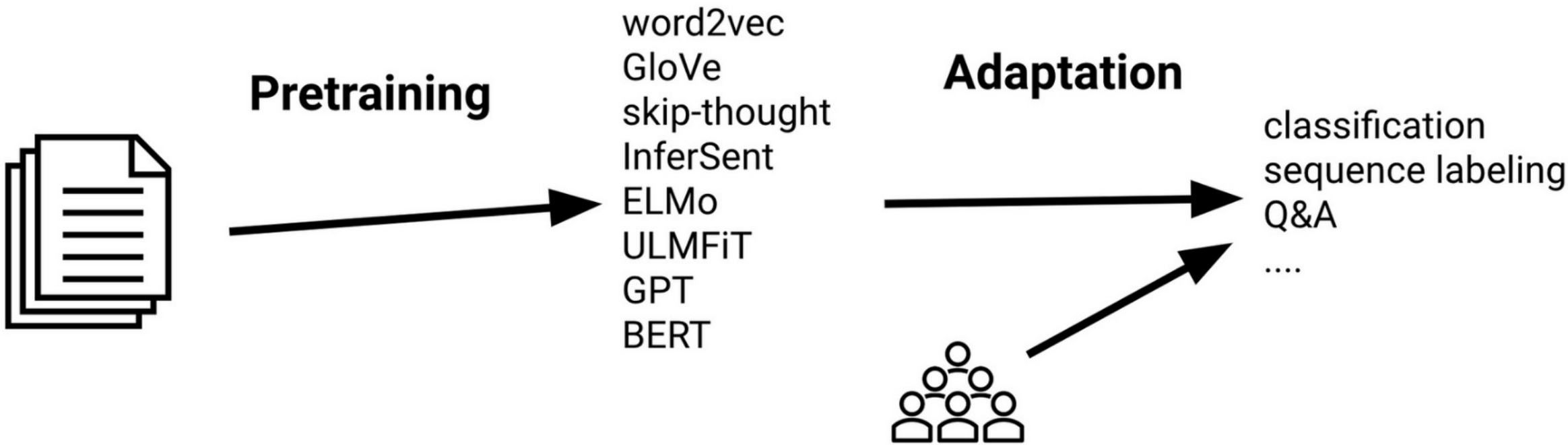


RUDER, 2019


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



RUDER, 2019

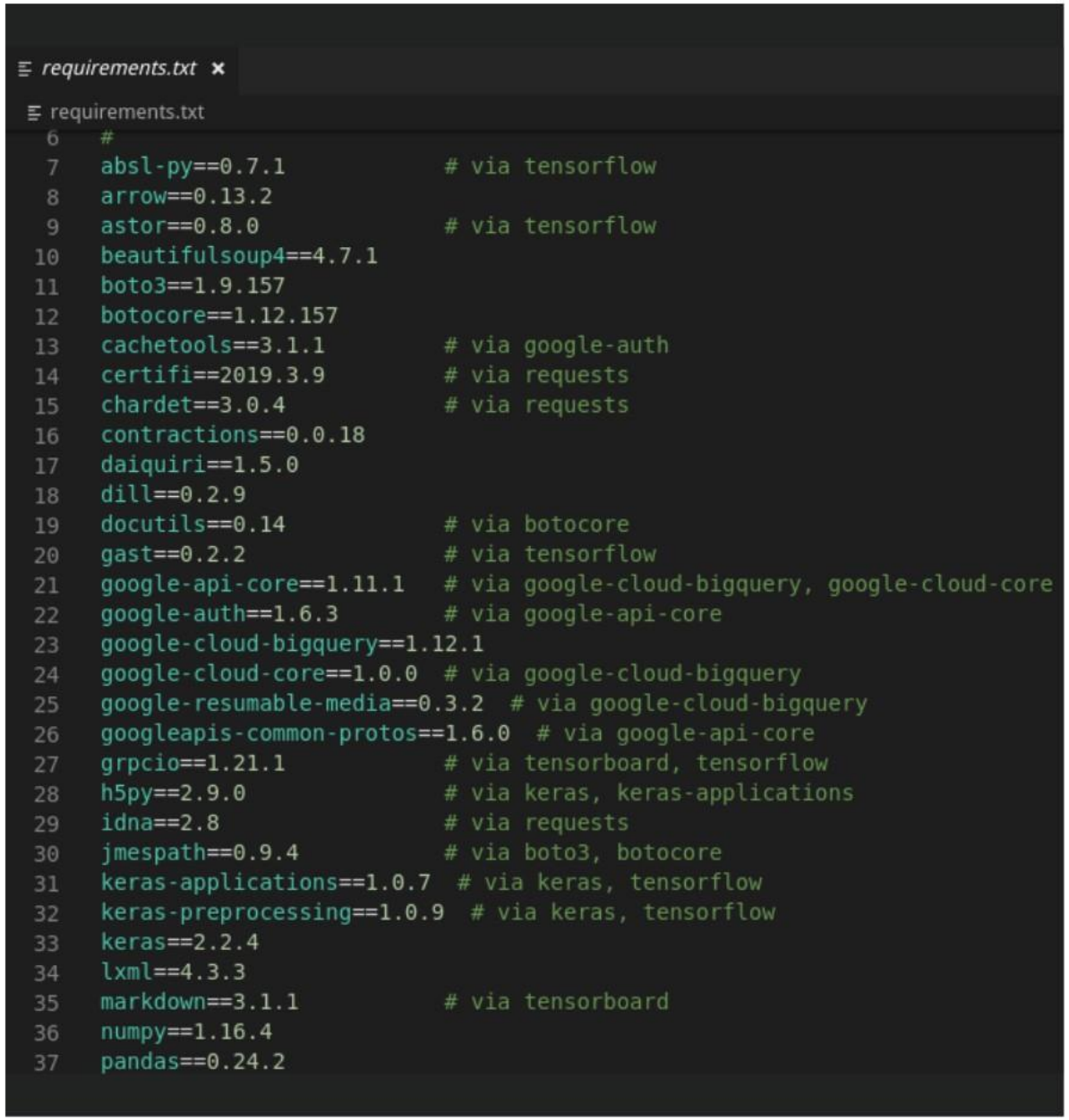
Case Study: Pro-active Security & Vulnerability Dependency Analytics

Developers don't code everything from scratch.

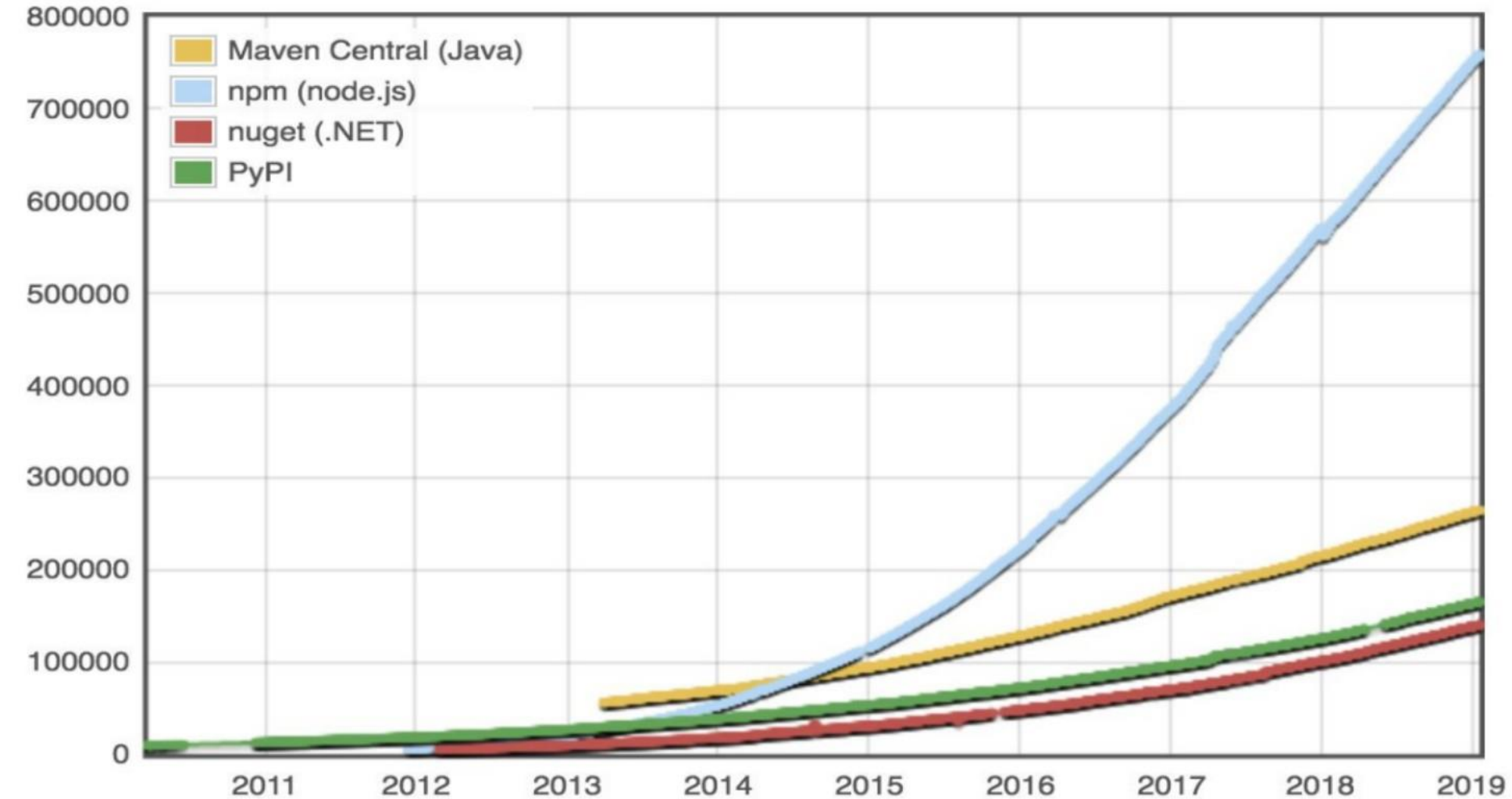
THEY “DEPEND” ON CODE WRITTEN BY OTHERS, THANK THE OPEN SOURCE SOFTWARE REVOLUTION

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



Dependency Landscape - So many options!

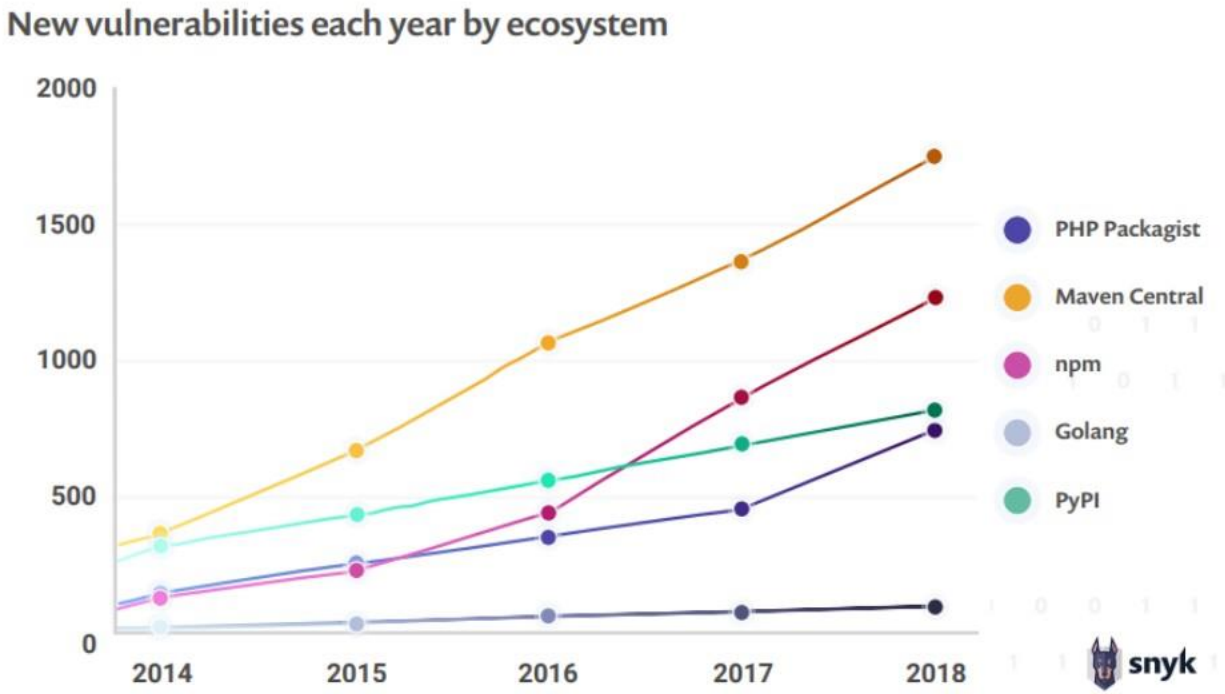


Dependency Security Vulnerability Landscape

What are security vulnerabilities?

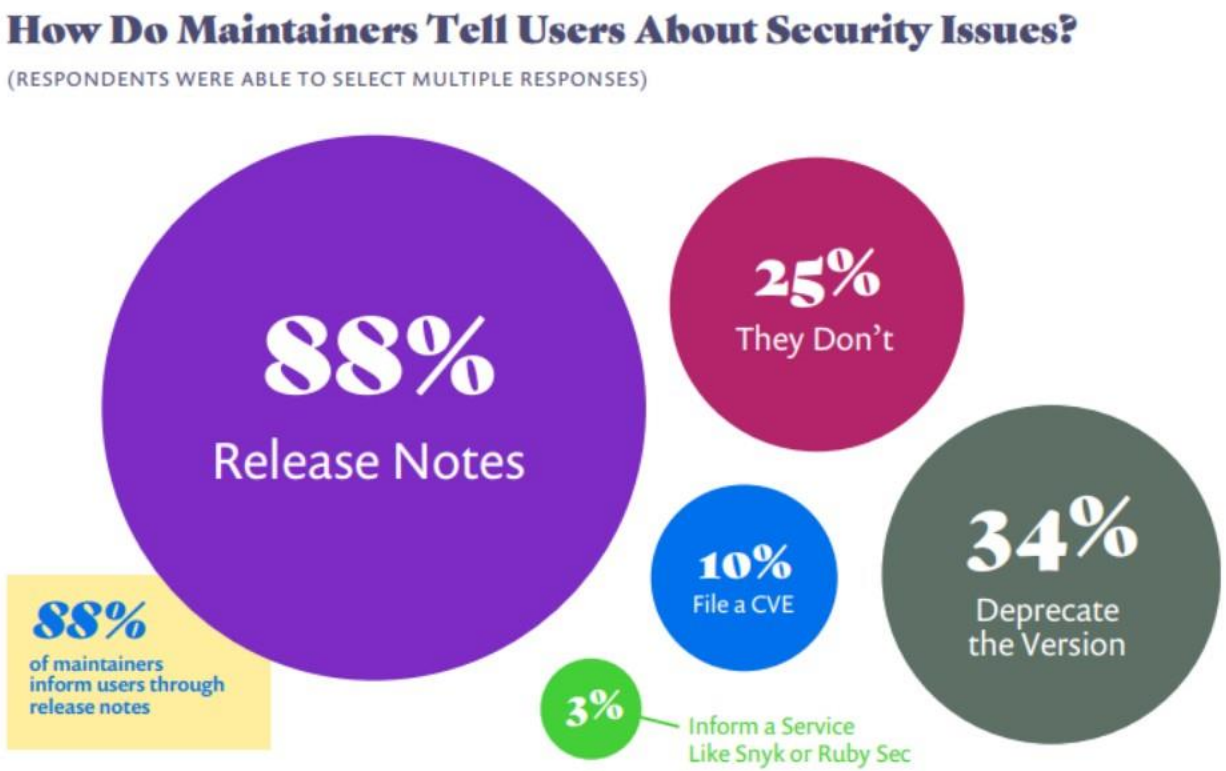
- Software security vulnerabilities are bugs or issues in the software which can lead to malicious attacks
- Commonly termed popularly as CVEs (Common Vulnerabilities and Exposures
- A vulnerability is a mistake in software code that provides an attacker with direct access to a system or network
- An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

Vulnerabilities in Application Dependencies



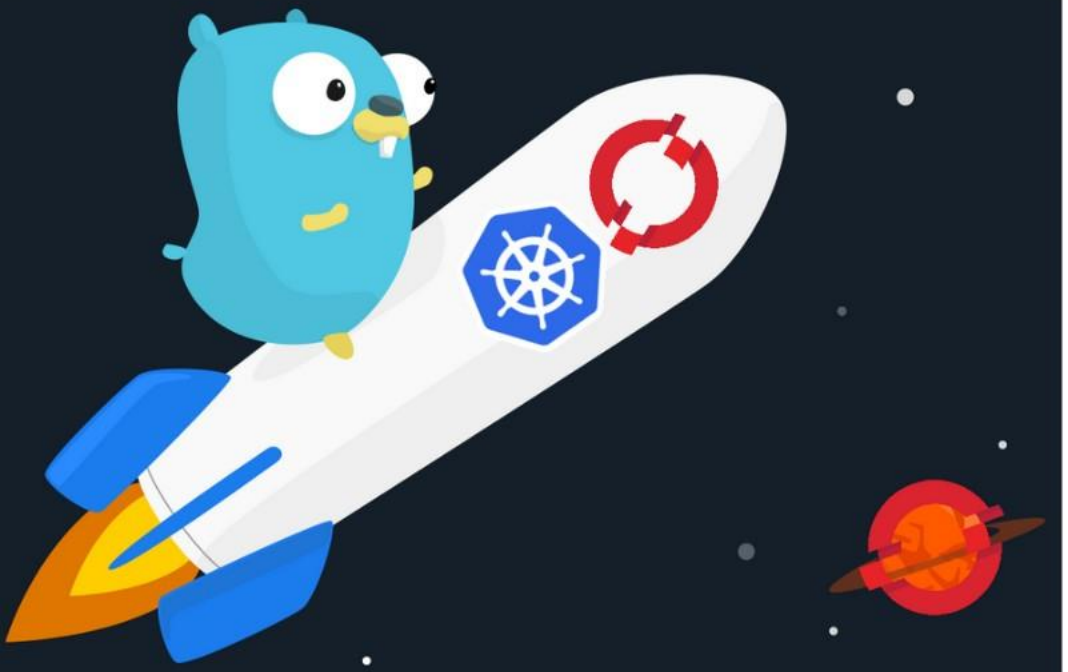
- A definite increasing trend in vulnerabilities across all popular eco-systems
- Vulnerabilities if not identified, are often missed and not published, for more complex eco-systems like golang
- 43% increase in vulnerabilities in 2017 and a 33% increase in 2018

Limitations with Existing Vulnerability Analytics



- 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

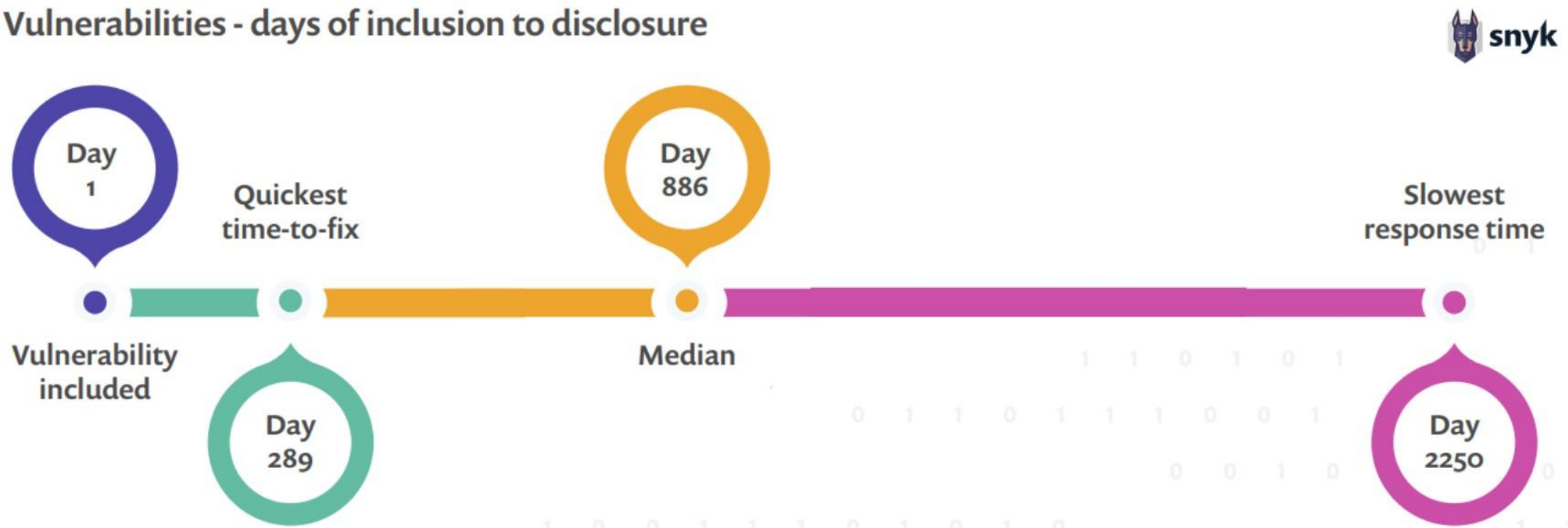


Key Objectives

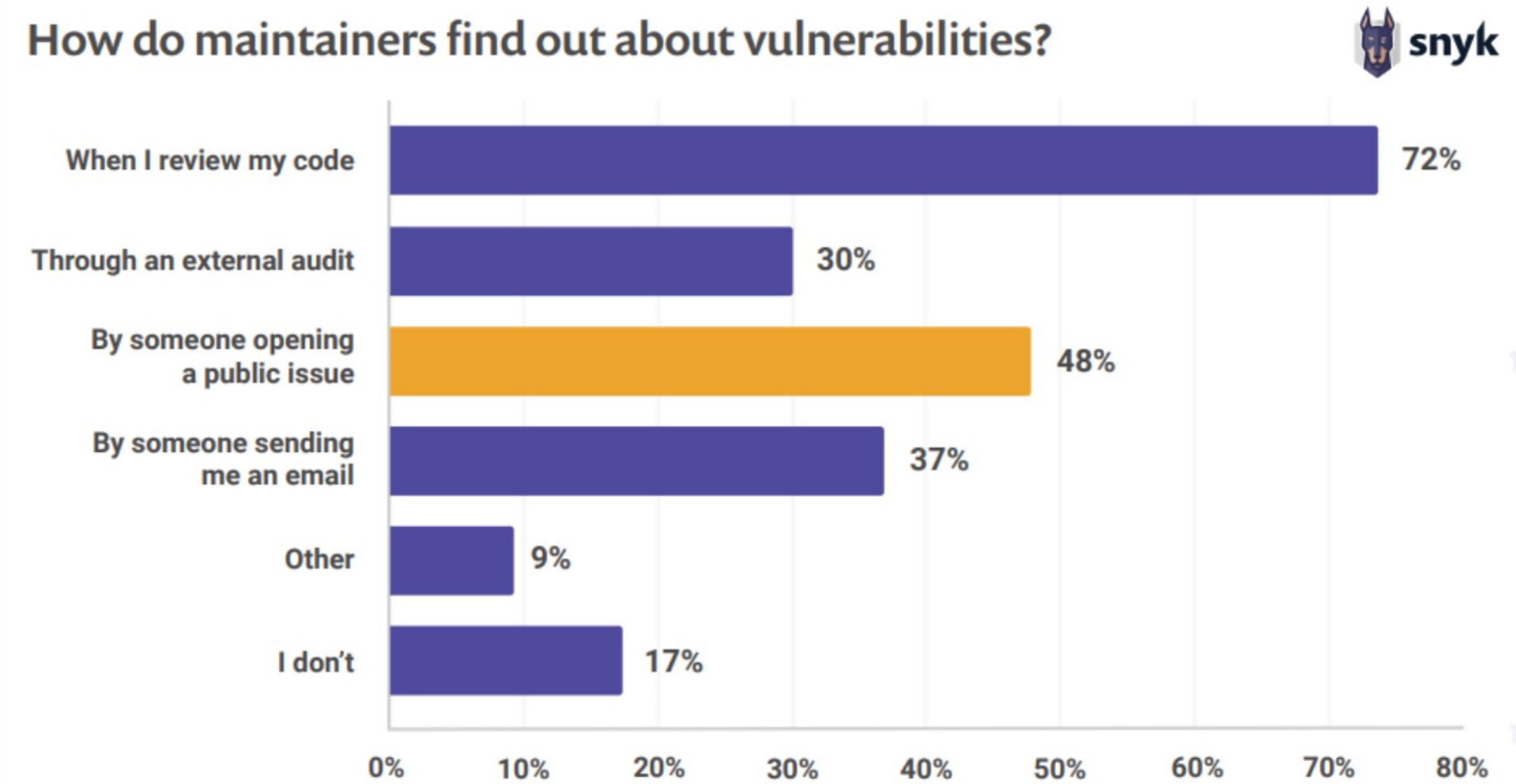
- Proactively predict probable vulnerabilities across direct & transitive dependencies in the golang - Kubernetes - OpenShift eco-system
- Integrate findings into developer and release processes of OpenShift in the long term
- Engage with the developer community in the future around proactive vulnerability insights


Business Significance

Vulnerabilities - Discovery to Disclosure



Vulnerabilities - Public Data can be tapped for insights

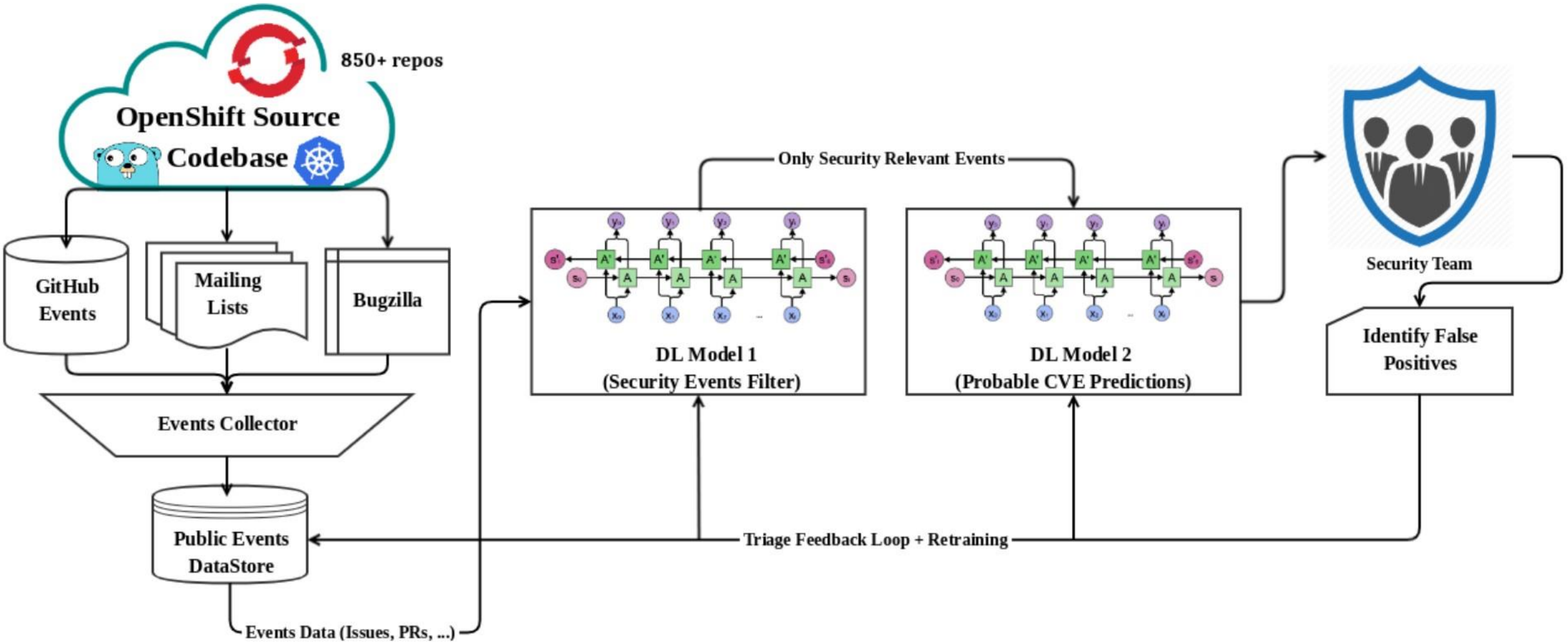


Business Significance - Current Market Space

- 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

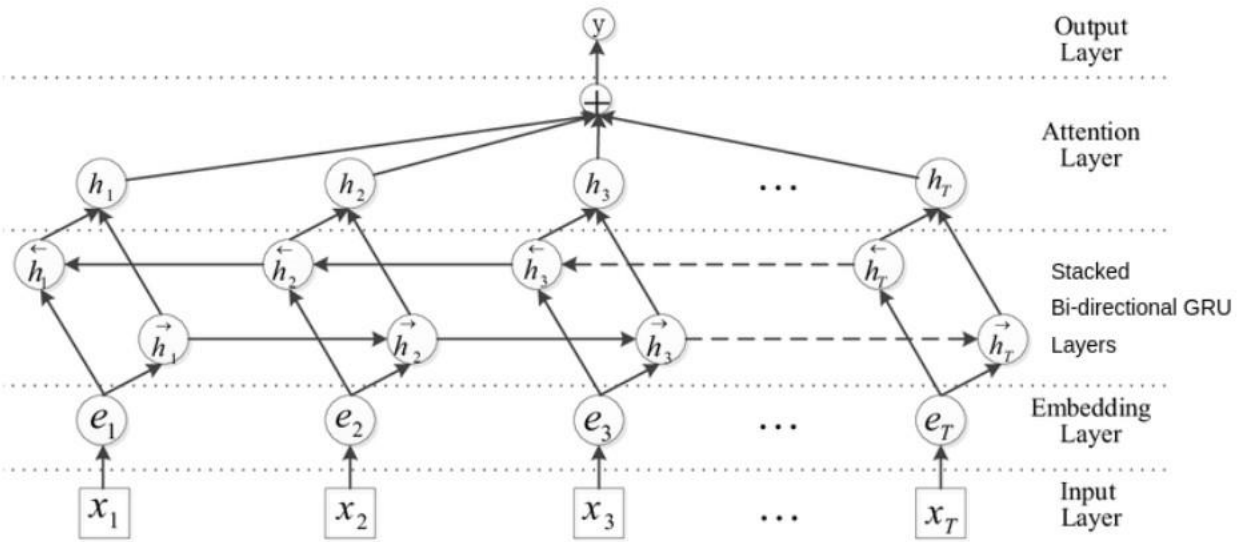
- Regularly monitor 850+ repositories \ dependencies being used by OpenShift
- Extract useful content from public events like issues, PRs, commits, comments, conversations
- DL Model 1 filters out security irrelevant events which are not of concern
- DL Model 2 uses filtered data to predict events which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


Deep Learning Model Architectures

Modeling Approaches

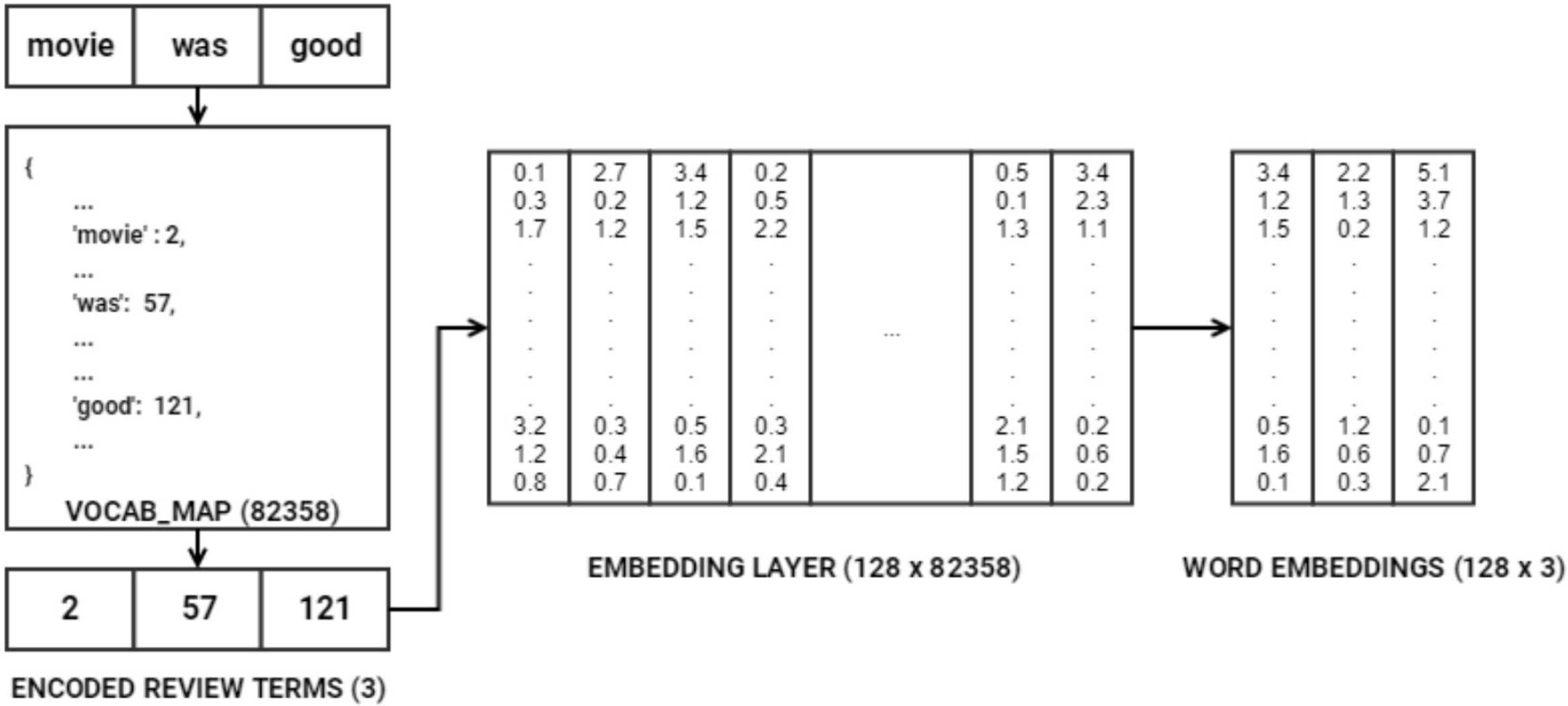
- **Baseline Machine Learning Models**
  Linear, Trees, Ensembles, Model Stacking
- **Baseline Deep Learning Models**
  DNNs, CNNs, LSTMs
- **Universal Language Model Fine-tuning (ULMFiT)**
- **Embeddings from Language Models (ELMo)**
- **Universal Sentence Embeddings**
  USE, NNLM
- **Bi-directional LSTMs\GRUs + Attention**
- **Bi-directional Encoder Representations from Transformers (BERT)**

Deep Learning Model Architecture

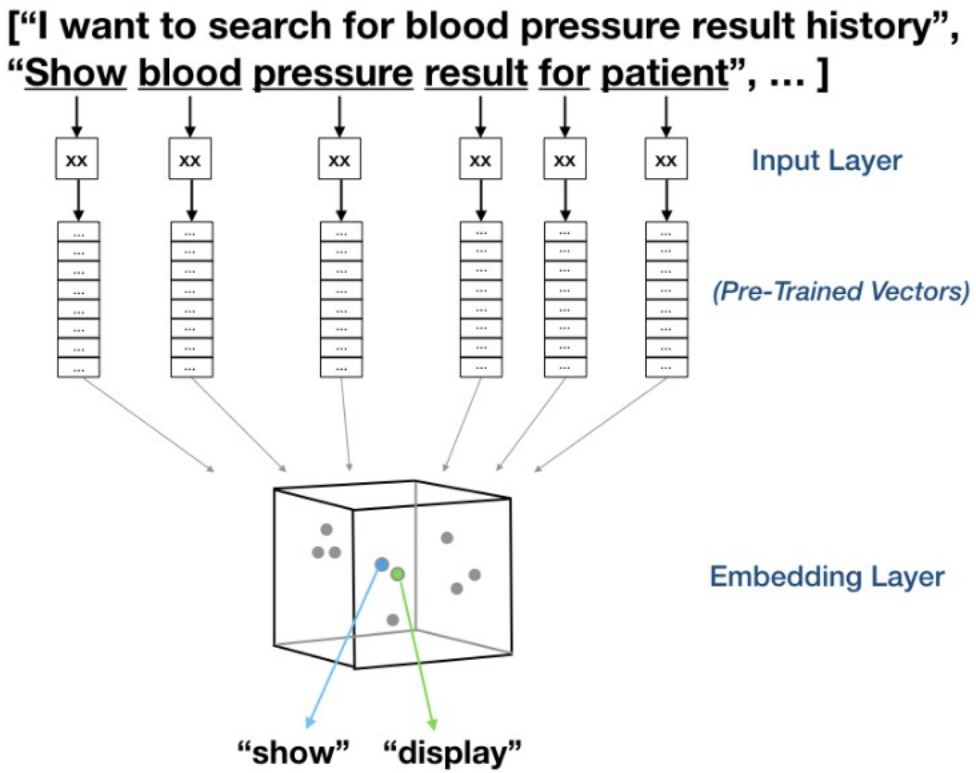


- **Embedding Layer populated with pre-trained embeddings**
  - FastText Embeddings
  - Paragram \ GloVe Embeddings
- **Stacked 2-layer Bi-directional Gated Recurrent Units (GRUs)**
- **Global Attention Layer**
- **FC Dense Layers**

Embedding Layer



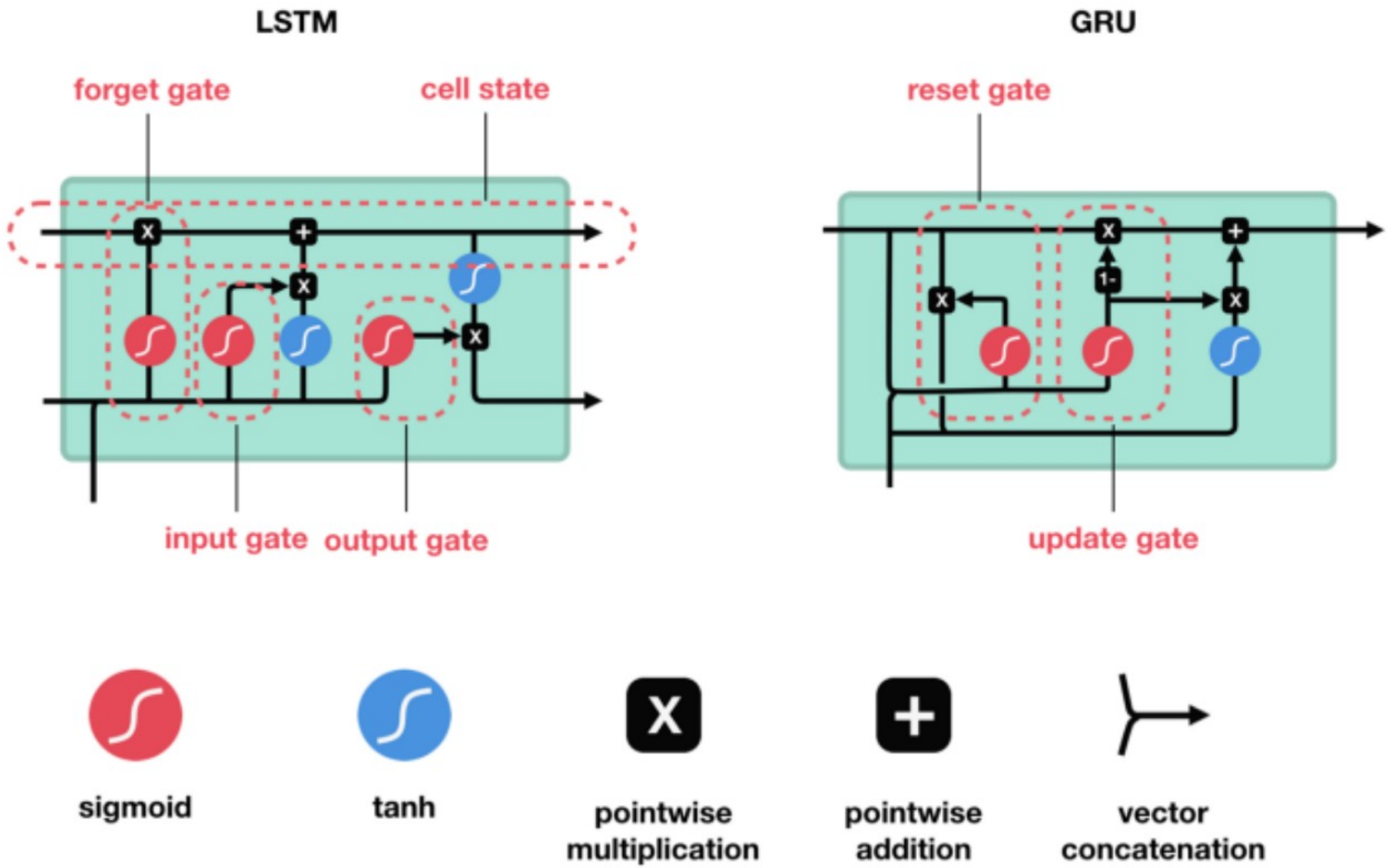
Embedding Layer



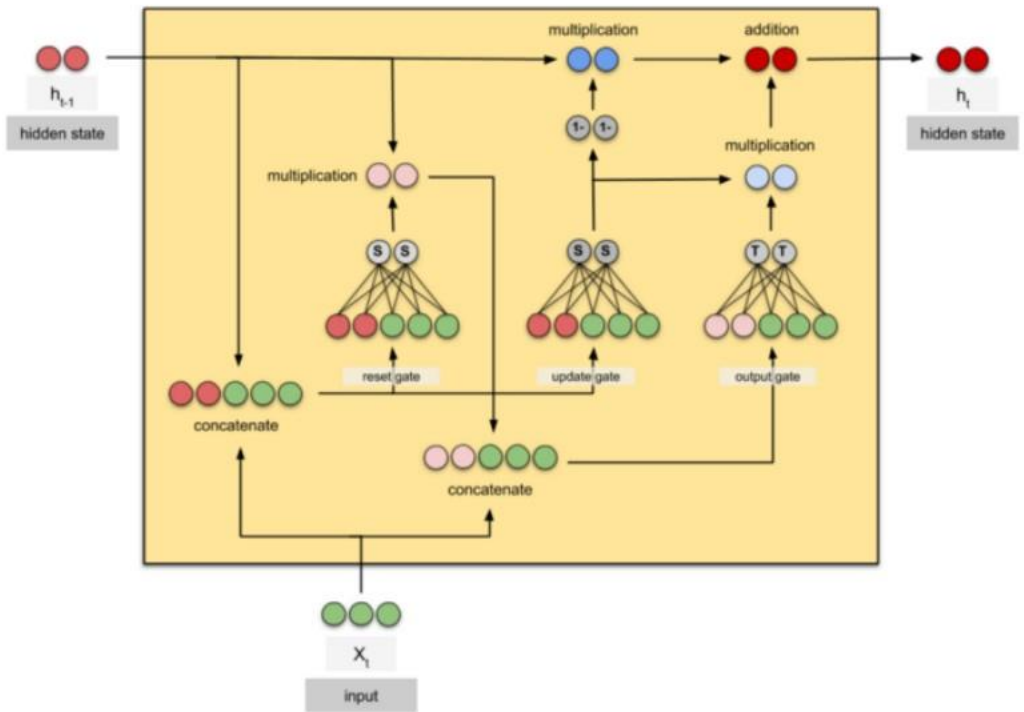
a	1
am	2
as	3
act	4
all	5
...	6
...	7
...	8
...	9
...	10
...	11
...	12
...	...
...	100
...	...
...	1000
...	...
...	10000
...	...
...	...

- Don't initialize layer with random weights
- Initialize embedding layer with pre-trained embeddings
- Averaged embeddings from FastText, GloVe \ ParaGram
- Model performed better than random initialization

LSTM vs GRU

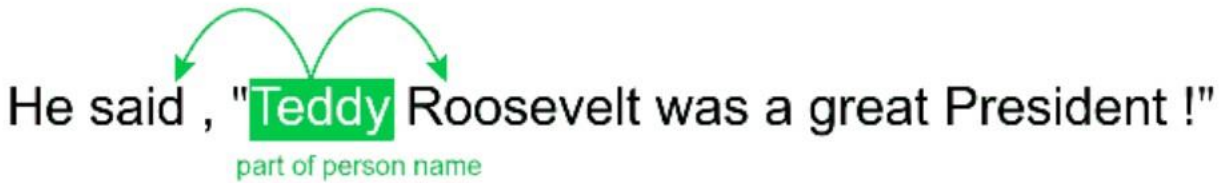

LSTM
forget gate
cell state
input gate
output gate
GRU
reset gate
update gate
1-
sigmoid
tanh
pointwise multiplication
pointwise addition
vector concatenation

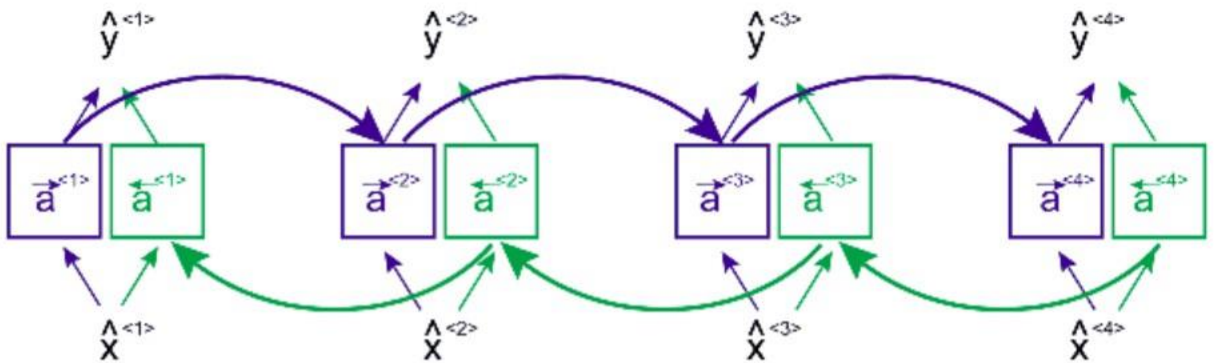
GRU - The subtle differences



- The update gate acts similar to the forget and input gate of an LSTM
- The update gate decides what information to throw away and what new information to add
- The reset gate decides how much past information should be retained
- Fewer tensor ops and speedier to train than LSTMs

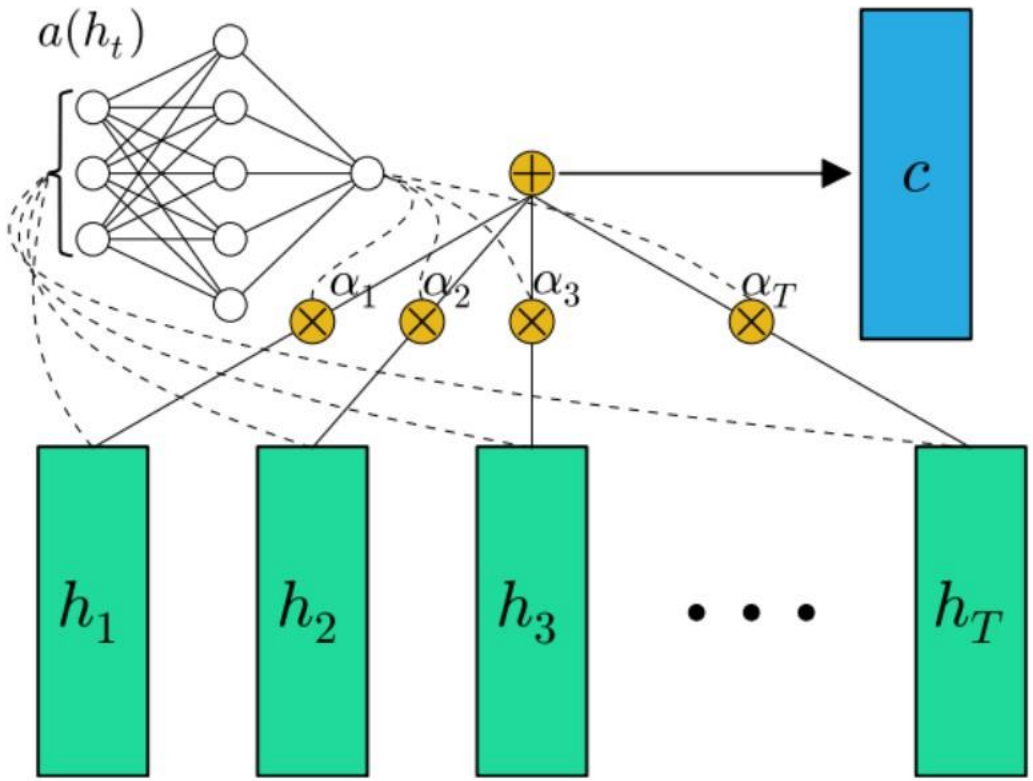
The need for Bi-directional GRUs





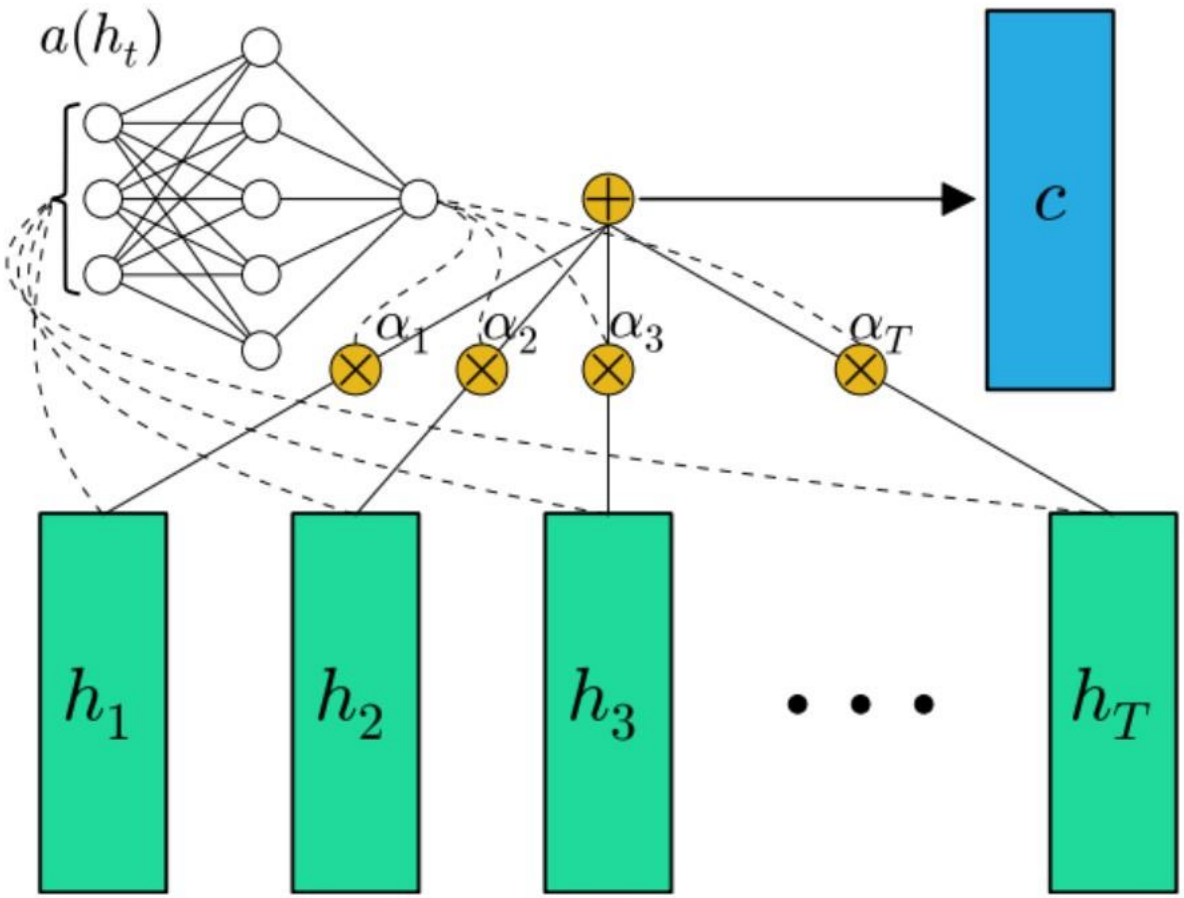
- Bi-directional GRUs are just putting two independent GRUs together
- The input sequence is fed in forward order for one GRU, and in reverse order for the other
- The outputs of the two networks are usually concatenated at each time step
- Preserving information from both past and future helps understand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

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 $\alpha_t.h_t$

Attention please!



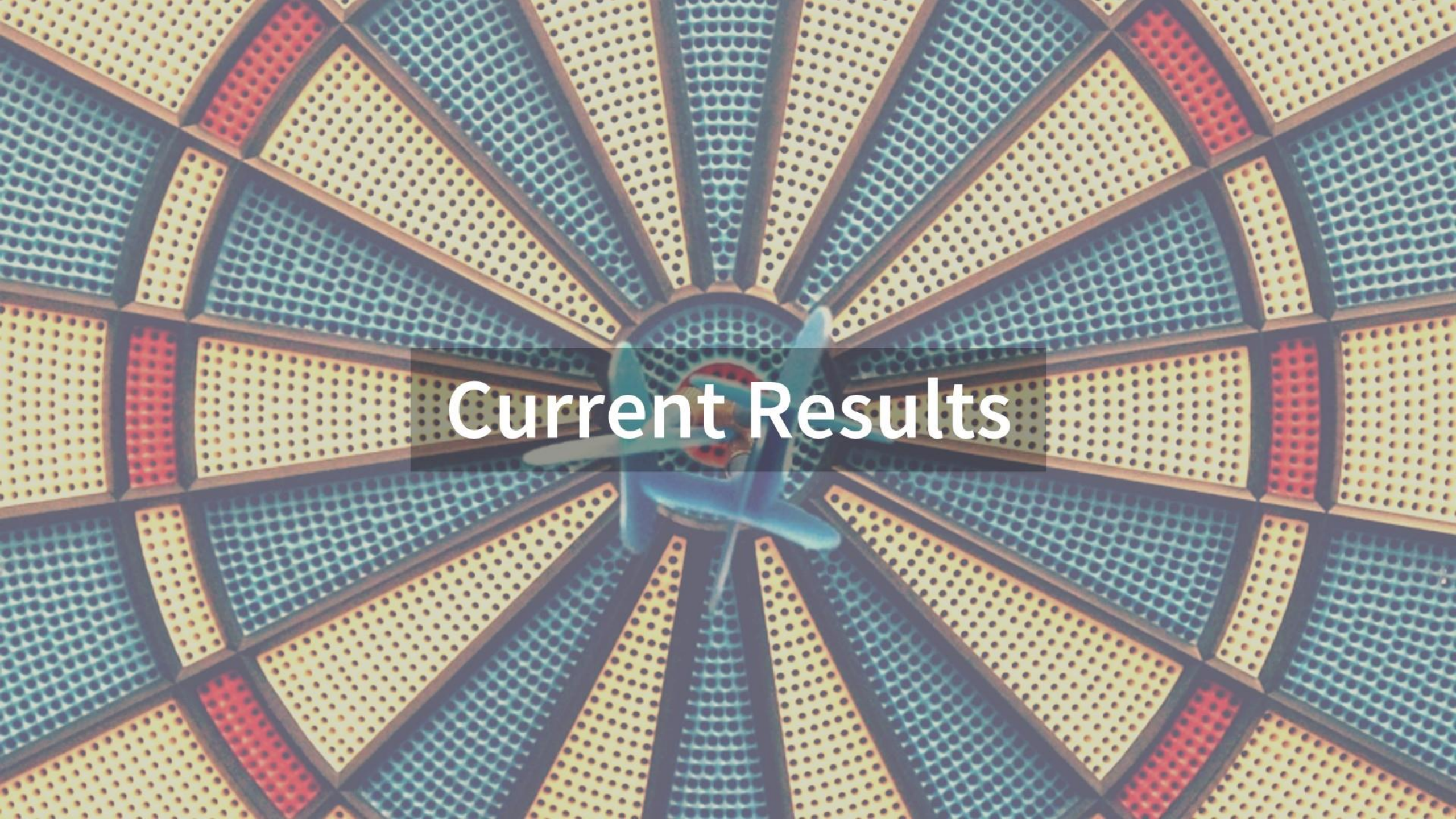
$$e_t = a(h_t) = \tanh(Wh_t + b)$$

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

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

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

Hands-on Tutorial



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
% Security Issues correctly identified	94.0%
% Probable Vulnerabilities correctly identified	70.0%
% Probable Vulnerability predictions actually true	65.0%
% Non-vulnerabilities correctly identified	99.0%

Analytics Vidhya
Learn everything about analytics

DATAHACK SUMMIT 2019

THANK YOU