C-SPECC Presentation Fall 2018 Deep Learning for Privacy and Code Analysis

John Heaps

Learning Information Type Semantics to Verify Privacy Requirements

John Heaps
Mitra Bokaei Hosseini

The Goal

- Mobile and web apps are becoming increasingly popular and prevalent; when using the services personal information about the user is collected and stored

- Collected information can expose users to potential privacy risks if mishandled or misused (Facebook, Google, etc.)

- California law and many European nations require app developers to provide users with a legal privacy notice (privacy policy) detailing what information is collected, how it is used, and with whom it is shared

- However, it is difficult to determine compliance verification (i.e., if the app code performs as described by the privacy policy) (e.g., HIPPA (Health Insurance Portability and Accountability Act of 1996))

- Currently this is done by utilizing look-up tables, platform permissions, and information flow analysis, which uses “information types” (i.e., keywords that describe different types of information), however:
    - Information types must be compiled manually which is time consuming
    - Information types (especially in natural language) are ambiguous (hypernymy, synonymy, etc.) which leads to inconsistencies and false positives/negatives in analysis

- Example from Adobe privacy policy:
    - “When you activate your Adobe product, we collect certain *information about your device*, the Adobe product, and your product serial number”
    - *information about your device* => information about your {mobile device, laptop, desktop, etc.}
    - Can imply device id, ip address, contacts, etc.

- Information types can be represented in an ontology
    - Can take steps to create and maintain ontology automatically
    - Will heavily reduce ambiguity of information types

- Ontology: a vocabulary of words/phrases and the relationships between them
    - Example: device is hypernymy of mobile device
    - Ours contains hypernymy and synonymy

- Can we utilize deep learning to maintain an ontology of information types?

Convolutional Neural Network (CNN)

What is it?

- Simplest Definition: A neural network that uses convolution hidden layers
- Very good at image processing
- Relatively little preprocessing of data needed

Convolution

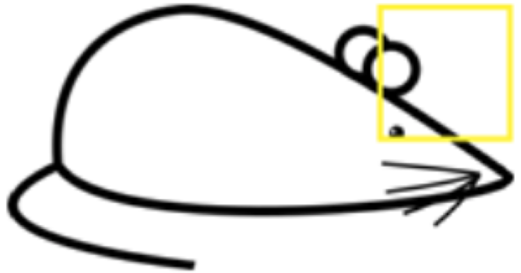
- In mathematics: an operation on two functions to produce a third function that expresses how the shape of one is modified by the other

- In neural networks: a matrix that acts as a filter (also called a kernel) to detect important features in data

- The filter is a matrix of weights that defines important features

- It slides over an image where the dot product is calculated between the filter and the current pixels it is sliding over (this is the convolution operation)

- Larger values indicate a stronger presence of the feature



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

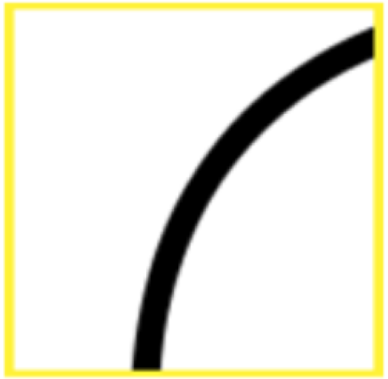
$*$

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

Source: https://towardsdatascience.com/convolutional-neural-networks-from-the-ground-up-c67bb41454e1


Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

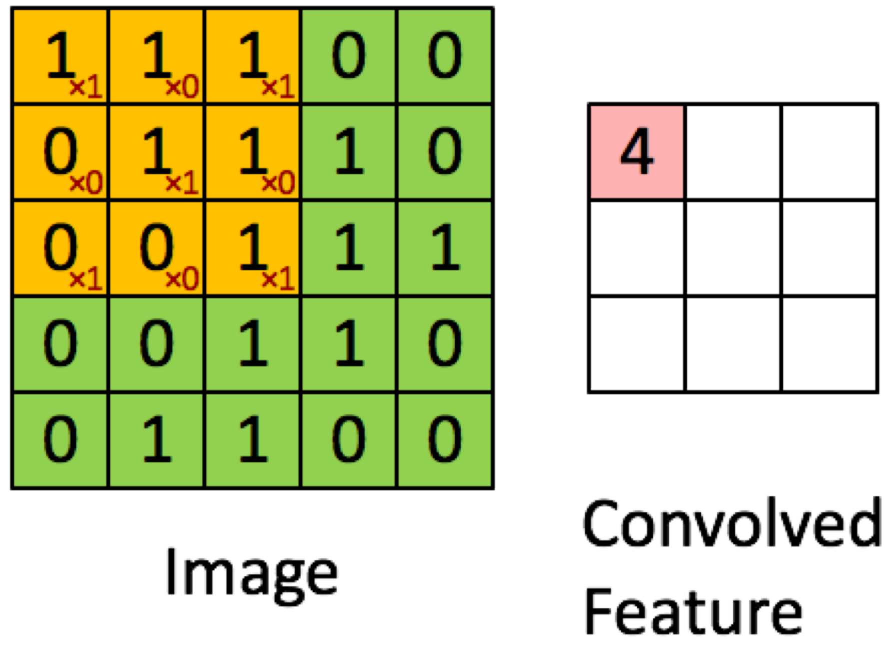
$*$

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Source: https://towardsdatascience.com/convolutional-neural-networks-from-the-ground-up-c67bb41454e1



1 ×1
1 ×0
1 ×1
0
0
0 ×0
1 ×1
1 ×0
1
0
0 ×1
0 ×0
1 ×1
1
1
0
0
1
1
0
0
1
1
0
0
4
Image
Convolved Feature

- The filter is the weights that are trained using backpropagation
- The filter size depends on a number of factors (type/size of data, programmer decisions, etc.)
- Usually multiple filters are trained at once to capture multiple important features
- Convolution is passed through a non-linear activation function before being processed by next layer (relu, tanh, etc.)
- Problem: this can take a LOT of time/calculations

Pooling

- Two main purposes:
  - Reduce the size of the image space (fewer computations)
  - Identify the important information

- Slides a window over the image and reduces the data in the window

- Many types of pooling, but most popular is “max pooling”

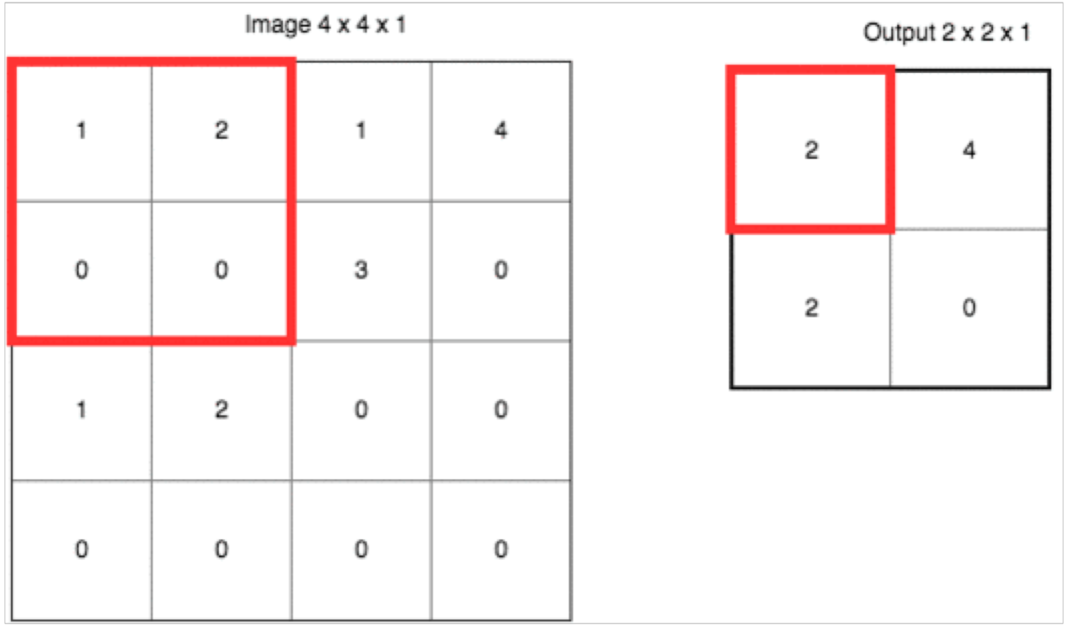


Image 4 x 4 x 1
1
2
1
4
0
0
3
0
1
2
0
0
0
0
0
0
Output 2 x 2 x 1
2
4
2
0

Word Embeddings

What is it?

- The mapping of a vocabulary of words/phrase to real-valued vector representations in order to perform calculations on them

- Vectors represent a relative semantic meaning between words/phrases in the vector space

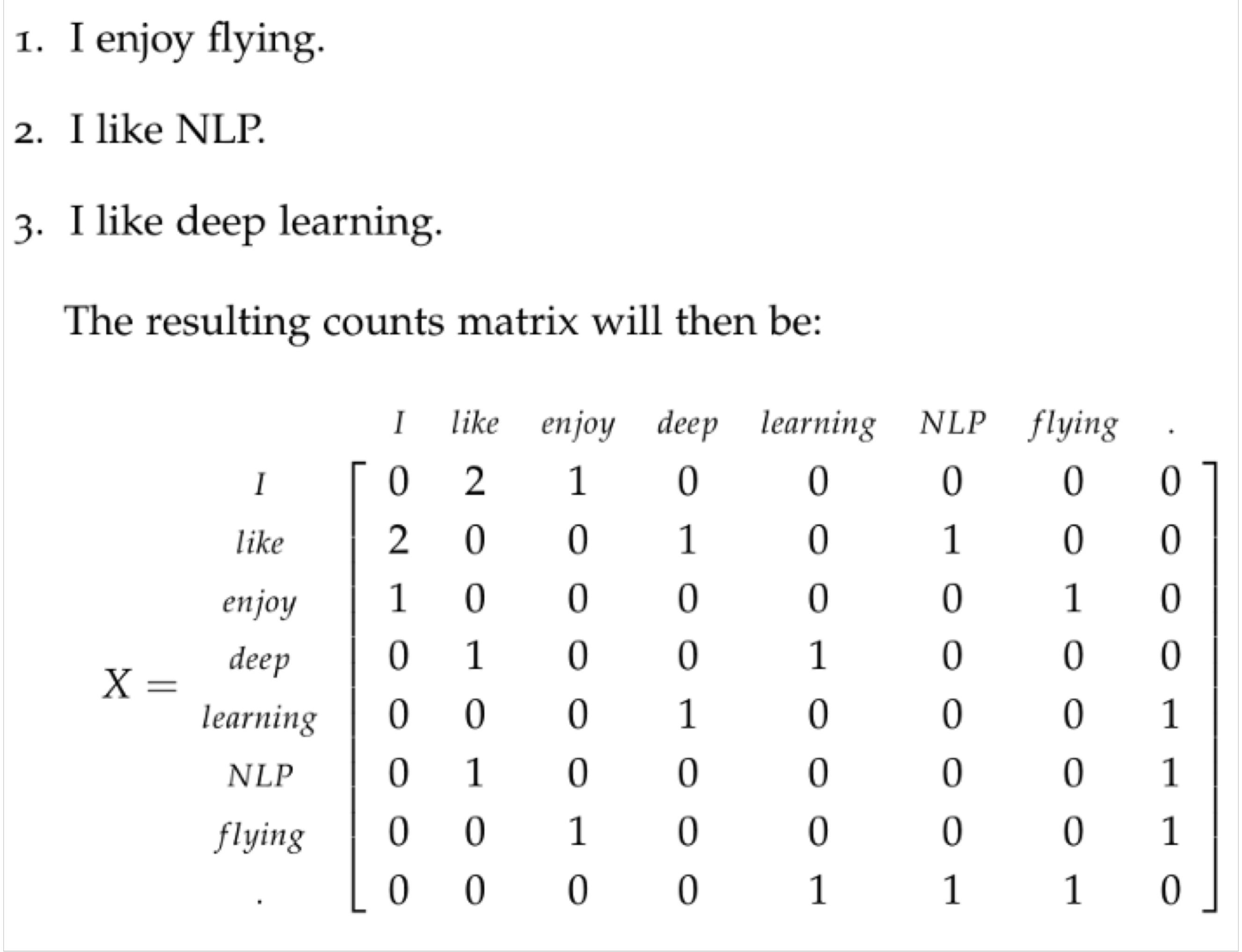
- Embeddings of words/phrases with similar semantic meaning should be grouped together in the vector space (e.g. - dog and puppy, etc.)

- Queen = King - Man + Woman

- Two of the most popular algorithms are GloVe and Skip-Gram

GloVe (Global Vectors for word representation)

- Word frequency based algorithm
- Constructs a word-to-word co-occurrence matrix
- Matrix factorization is utilized to determine the vector values for each word



1. I enjoy flying.
2. I like NLP.
3. I like deep learning.

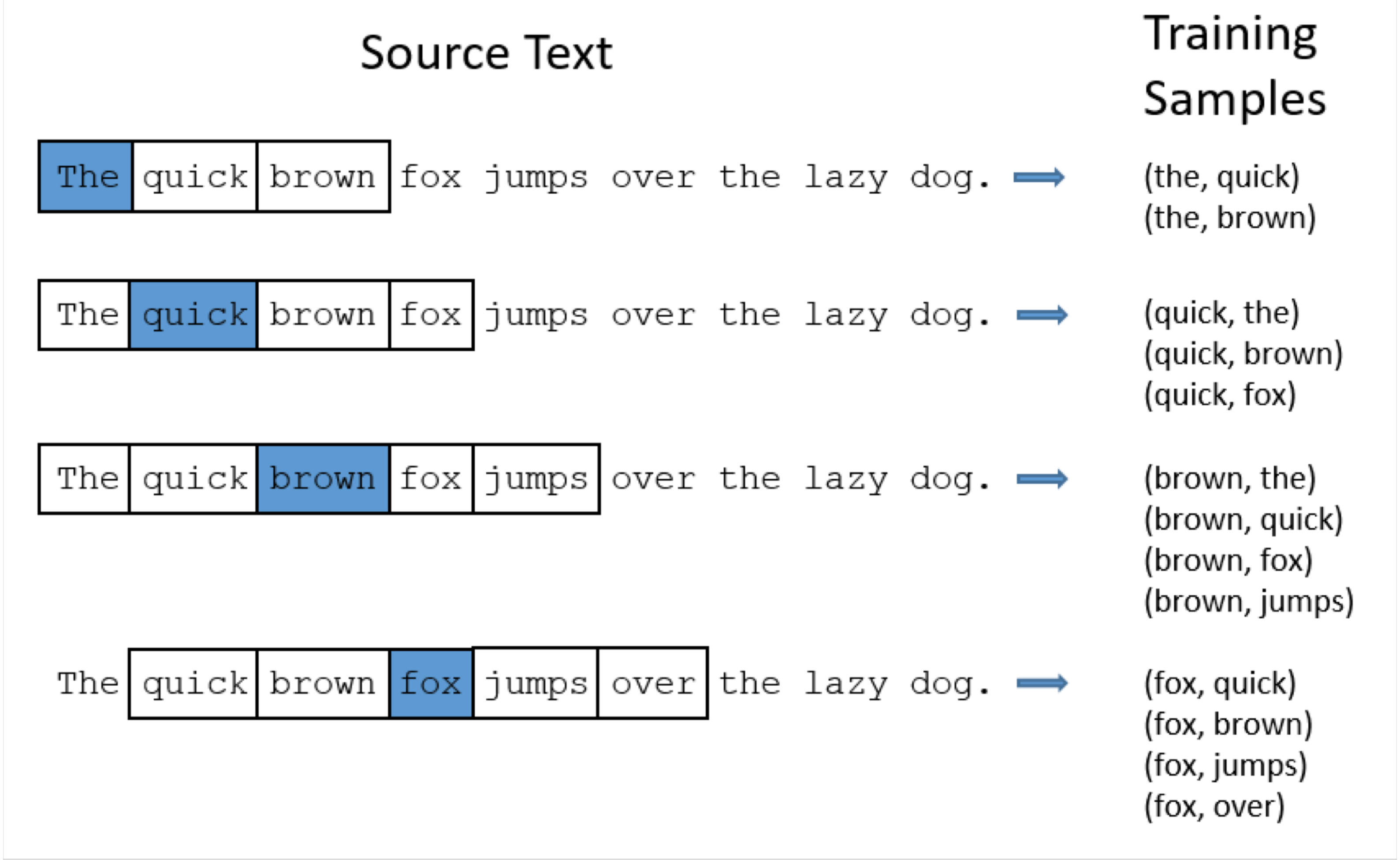
The resulting counts matrix will then be:

$$X = \begin{array}{c|cccccccc} & I & like & enjoy & deep & learning & NLP & flying & . \\ \hline I & 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ like & 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ enjoy & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ deep & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ learning & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ NLP & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ flying & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ . & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{array}$$

Source: https://towardsdatascience.com/word-to-vectors-natural-language-processing-b253dd0b0817

Skip-Gram

- Word prediction based algorithm
- A window size is defined
- For each word in the corpus the surrounding words (defined by the window size) are used as context for that word
- A neural network is used to predict the context for that word and modifies the vector values for that word during backpropagation



Source: https://towardsdatascience.com/word-to-vectors-natural-language-processing-b253dd0b0817

Network Training and Results

The Data

- Word Embeddings
  - Trained domain-specific word embeddings using 77,556 English privacy policies collected from mobile apps from Google Play Store
  - Used Word2Vec which is a Skip-gram algorithm

- Base Ontology
  - We must have an existing ontology to train our model
  - Current ontology has 367 information types extracted from 50 privacy policies
  - The relations between each pair of information types were annotated by 6 experts with hypernymy, synonymy, or unrelated, roughly resulting in 67,161 comparisons
  - Resulted in 1583 hypernym pairs, 310 synonym pairs, and 65,268 unrelated pairs
  - 90% of each of the pairs was used for training and 10% for testing

- Extending Ontology:
  - We found 74 more unique information types from 6 other privacy policies to be added to the existing ontology
  - A ground truth (hypernym and synonym relations) was determined for the extended ontology
  - Pairs were formed between each 74 new information types and 367 existing information types which were given to our trained model to be classified

The Approach

- To add a new information type to the base ontology, we must determine relationships between it and ALL current information types in the ontology

- Input to the model will be a pair of information types (1 new, 1 existing)

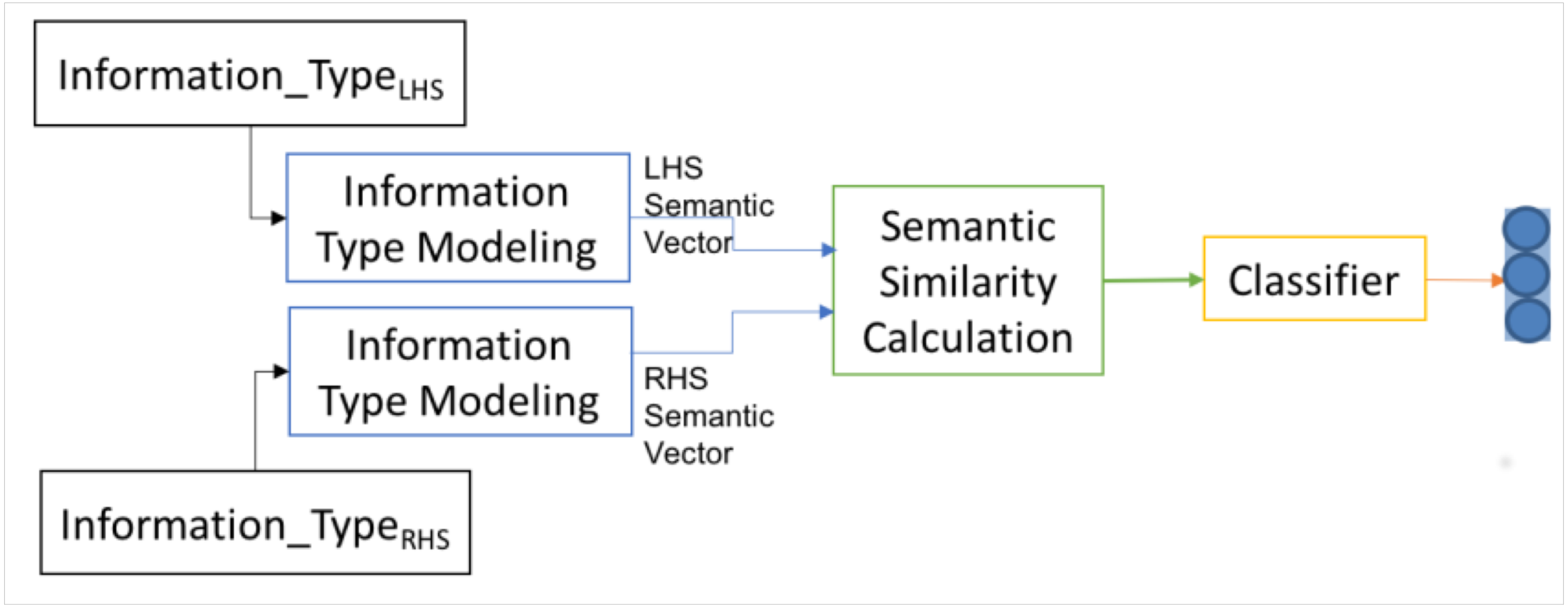
- The semantic meaning of each information type will be modeled using a CNN (as it is difficult to directly find embeddings for information types)

- A semantic similarity will be calculated between the two CNN outputs

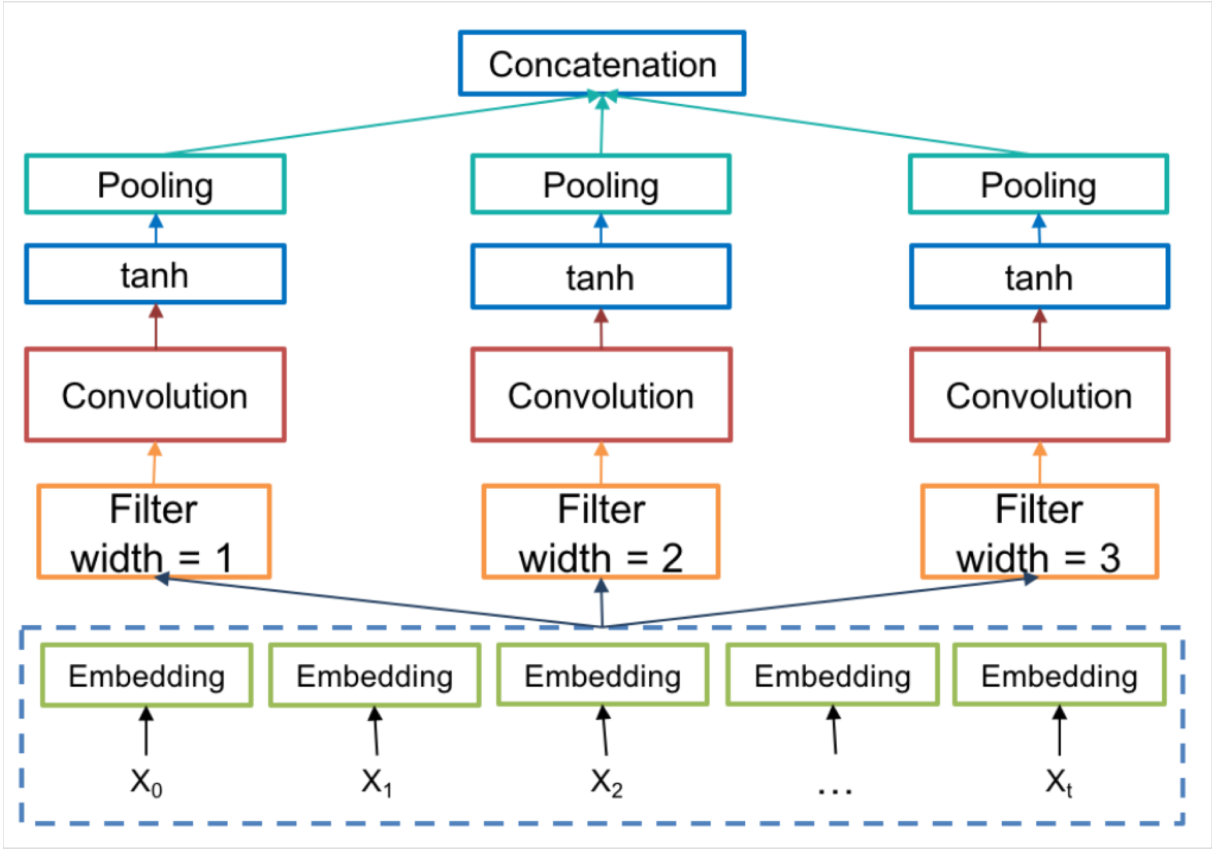
- Softmax will determine if the pairs relationship is hypernymy, synonymy, or unrelated

- New information type will be added based on all determined relationships

Model Architecture



CNN


Concatenation
Pooling
Pooling
Pooling
tanh
tanh
tanh
Convolution
Convolution
Convolution
Filter width = 1
Filter width = 2
Filter width = 3
Embedding
Embedding
Embedding
Embedding
Embedding
X_0
X_1
X_2
…
X_t

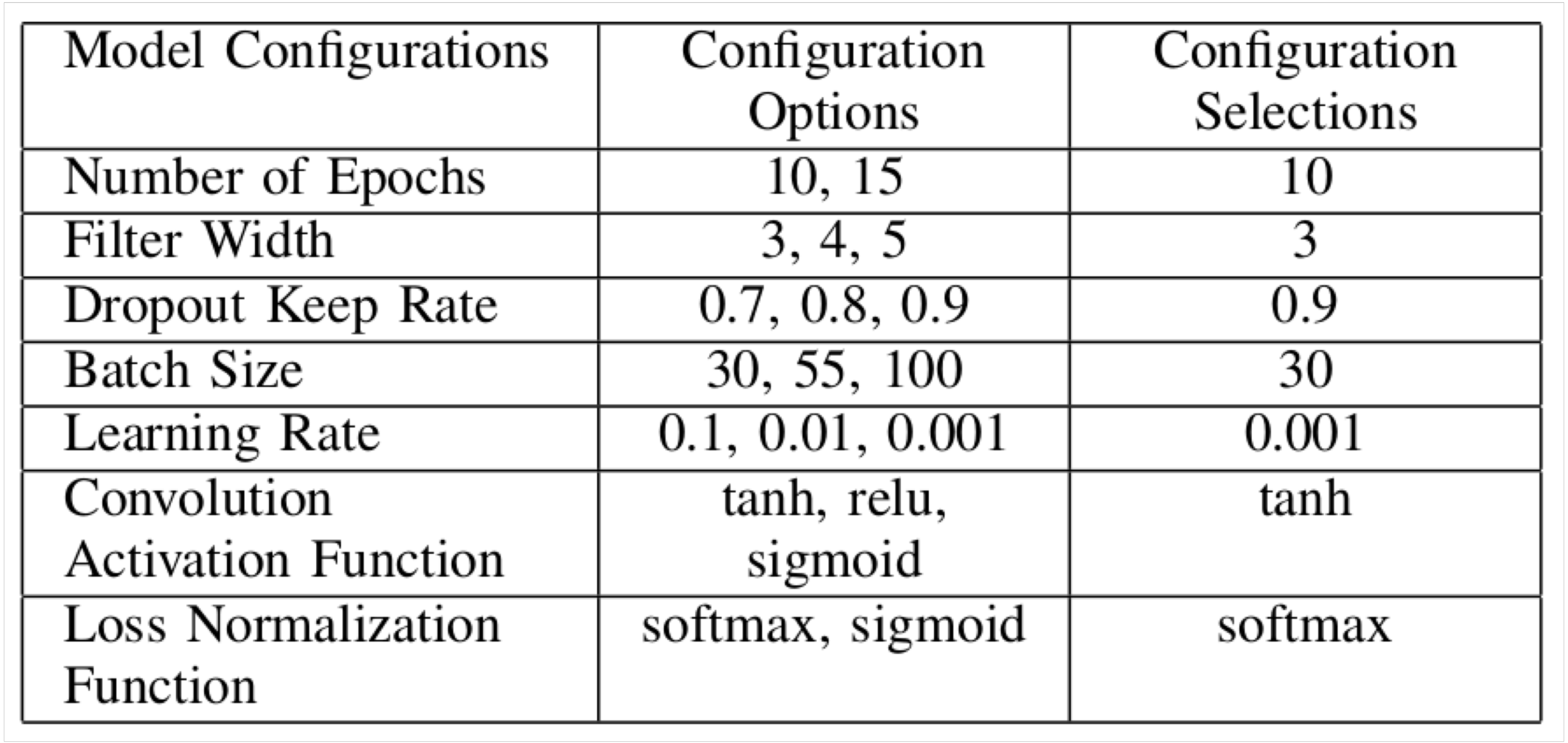
Semantic Similarity Calculation

$$dir = V_{LHS} \odot V_{RHS}$$

$$Dis = |V_{LHS} - V_{RHS}|$$

$$sim = \sigma(W\,dir + U\,dir + b)$$

Model Configurations (Hyper Parameters)



Model Configurations	Configuration Options	Configuration Selections
Number of Epochs	10, 15	10
Filter Width	3, 4, 5	3
Dropout Keep Rate	0.7, 0.8, 0.9	0.9
Batch Size	30, 55, 100	30
Learning Rate	0.1, 0.01, 0.001	0.001
Convolution Activation Function	tanh, relu, sigmoid	tanh
Loss Normalization Function	softmax, sigmoid	softmax

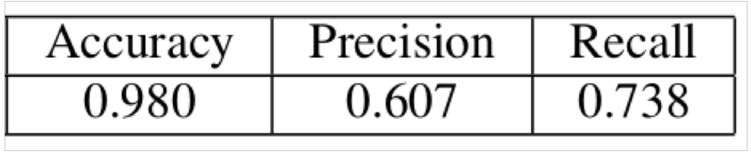
The Results

- True Positive (TP) = pair predicted as hypernym or synonym and is present in the ground truth
- False Positive (FP) = pair predicted as hypernym or synonym and is not present in the ground truth
- False Negative (FN) = pair predicted as unrelated and is not present in the ground truth
- True Negative (TN) = pair predicted as unrelated and is present in the ground truth

$$accuracy = (TP + TN)/(TP + TN + FP + FN)$$

$$precision = TP/(TP + FP)$$

$$recall = TP/(TP + FN)$$



Accuracy	Precision	Recall
0.980	0.607	0.738

- We also performed an example violation detection using the base ontology and new ontology extended by our model
- We used a mapping from API method calls to information types to determine if a violation existed
- Performed the detection over 501 apps and found 26 new violations across 14 apps

Future Work

- Compare results of using RNN to CNN
- Find ways to reduce the FP and FN
- Currently, the model can determine the type of relationship between information type pairs, but cannot determine the direction of that relationship (hypernymy); will find an alternative to the semantic similarity measure in order to determine this

? Questions ?

Understanding the Meaning of Code Elements Using Deep Learning

John Heaps

The Goal

- Program analysis on software source code can be used to assist in many developer tasks

- Many tools and approaches exist that rely on mappings from code elements and patterns to high-level concepts to perform analysis
    - Is usually a manual or semi-automated process which can be quite costly
    - Difficult and costly to stay up-to-date with new code versions (e.g. - languages, APIs, etc.)

- An automated technique that can construct mappings from code elements to high-level concepts will significantly enhance usability of code analysis tools in practice

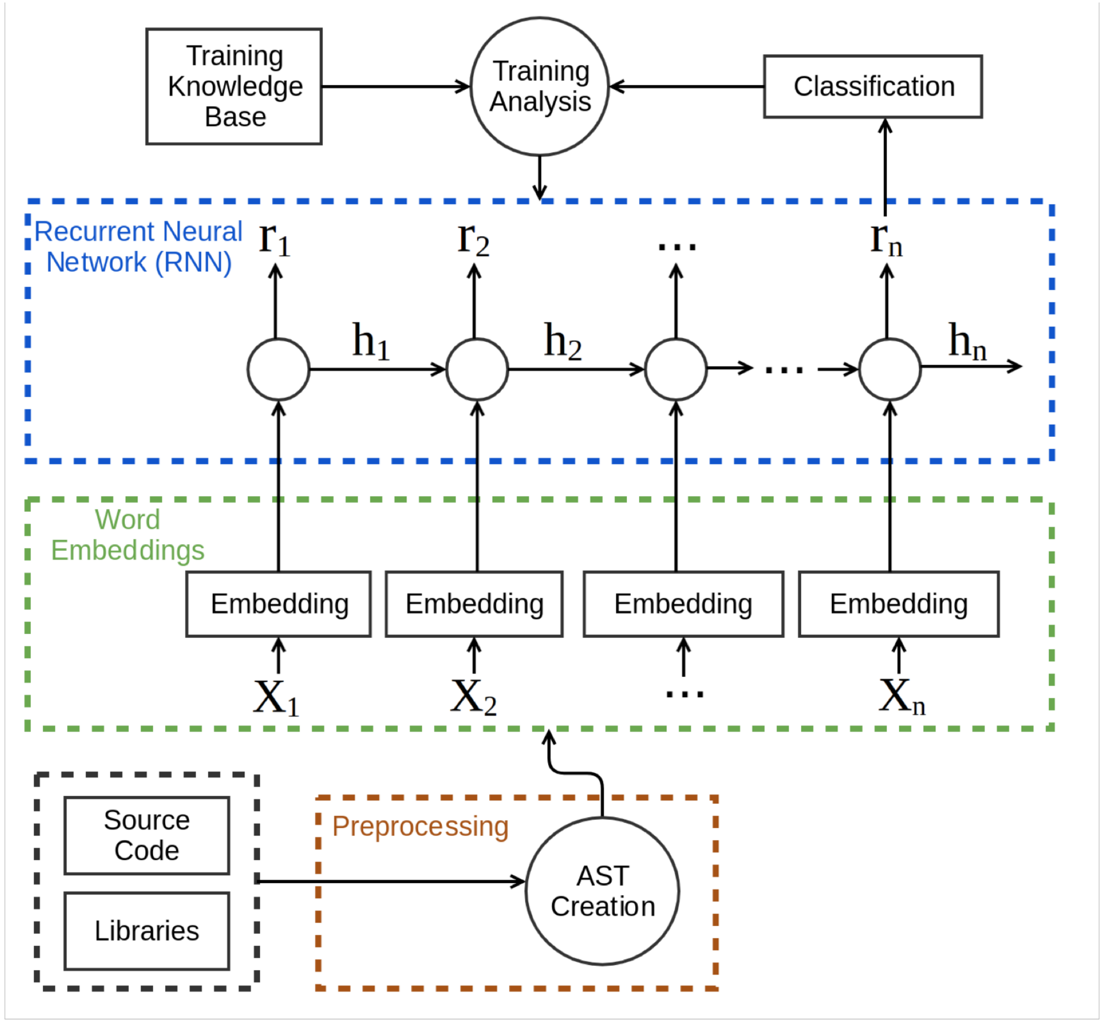
- We propose using Word2Vec to represent code elements and deep learning to determine mappings

The Approach

- Word Embeddings
  - Preprocessing of code:
    - Code is very different from natural language, particularly the structure
    - We will convert code to an Abstract Syntax Tree (AST)
  - Using an AST, Word2Vec context can be based on structure, not just a window size
  - Will perform an analysis on embeddings to determine if they are reasonable (i.e. - like code elements should be grouped together in the word embedding vector space)

- Deep Learning
  - We will train a Recurrent Neural Network (RNN) to perform classification
  - RNN is appropriate as a code element is either dependent on what elements have come before it in its scope or dependent on a definition which is a series of code elements

- Training Analysis
  - We will compile a ground truth of mappings and use 90% for training and 10% for testing


Training Knowledge Base
Training Analysis
Classification
Recurrent Neural Network (RNN)
r_1
r_2
…
r_n
h_1
h_2
…
h_n
Word Embeddings
Embedding
Embedding
Embedding
Embedding
X_1
X_2
…
X_n
Source Code
Libraries
Preprocessing
AST Creation

Current State of Research

- Data
  - Our initial approach will focus on access control libraries in Java
  - Identified 4 access control libraries:
    - Spring Security
    - Casbin
    - Apache Shiro
    - Google IAM
- Preprocessing
  - We are currently modifying Word2Vec to accept dynamic number of inputs based on code structure (vs. the normal static window size based on word position)
- Knowledge Base and Classification
  - We are currently compiling data for the Knowledge Base and the ground truth for classification results to be measured against

? Questions ?