卒業論文 2018 年度 (平成 30 年)

低対話型 Honeypot のコマンド拡張による
高対話型 Honeypot への近似

慶應義塾大学 総合政策学部
菅藤 佑太

卒業論文要旨 - 2018 年度 (平成 30 年度)

低対話型 Honeypot のコマンド拡張による 高対話型 Honeypot への近似

PC の普及や IoT デバイスのシステム高度化により，高度な処理系を組むことが可能になった．これによりデバイス上に Linux 系などの OS が搭載された機器が広く人々に使われるようになった．また，Linux 系 OS にリモートログインする手法として SSH がある．これを用いて不正に侵入する攻撃が行われている．侵入された際に侵入者がどのような挙動をしているのかを知る手段として，Honeypot がある．Honeypot は SSH で侵入しやすいような環境を作ることで，侵入者にログイン試行に成功したと検知させ，その際に実行したコマンドのログを収集するものである．また現在では Shell の挙動をエミュレートした Honeypot が広く使用されており，この Honeypot は実行できるコマンドが少ない実装になっている．そのため Honeypot への侵入者に侵入先が Honeypot であると検知されてしまう．そこで事前実験では Honeypot のコマンドを拡張し，拡張をしていない Honeypot とコマンドの拡張をした Honeypot で侵入ログを収集した．収集したログを確率的な算出方法を使用することで比較した結果，より多くの侵入者のコマンド実行ログのパターンを取得できることを示した．本研究ではコマンドを拡張した Honeypot の侵入ログがどれほど実際の OS に不正な SSH の侵入をされた際の侵入ログの近似を試みた．評価として，拡張をしていない Honeypot とコマンドの拡張をした Honeypot と，さらに実際の OS を使用した Honeypot で侵入ログを収集し，比較を行なった．この 3 つの侵入ログを自然言語処理の意味解析を用い，コマンドの一つ一つの意味をベクトル表現することで，拡張した Honeypot で収集した侵入ログが，拡張をしていない Honeypot で収集した侵入ログよりも，実際の OS を使用した Honeypot で侵入ログに意味的に近くなることが明らかとなった．

キーワード:
1. 自然言語処理，2. 意味解析，3. Honeypot，4. SSH

慶應義塾大学 総合政策学部
菅藤 佑太

Abstract of Bachelor's Thesis - Academic Year 2018

Approximation of high-interaction honeypots by Command Extension of low-interaction honeypot

By the spread of PCs and the system advancement of the IoT device, I was able to make high processing system. The apparatus that the OS's such as the Linux system were equipped with on a device came to be in this way used for people widely. In addition, technique to perform a remote login for the Linux system OS includes SSH. An attack to invade using this illegally is carried out. A window includes Honeypot what kind of ways an intruder behaves when it was invaded. Honeypot lets an intruder detect it when I succeeded in a login trial by making the environment where it is easy to invade in SSH and collects the log of the command that I carried out on this occasion. In addition, Honeypot which emulated behavior of Shell is used widely now, and there are few commands that this Honeypot can carry out; is implemented. Therefore it is detected by an intruder to Honeypot when invasion is Honeypot. Therefore I expanded the command of Honeypot by the prior experiment and collected invasion log in Honeypot and Honeypot which I expanded of the command which were not expanded. As a result of having compared the log that I collected by using a probabilistic calculation method, I showed that I could acquire a pattern of the command practice log of more intruders. Invasion log of Honeypot which expanded the command tried an approximation of the invasion log when it was invaded of unjust SSH for the real OS in this study how long. I collected invasion log more in Honeypot using the real OS and compared it with Honeypot and expanded Honeypot of the command which were not expanded as an evaluation. I used semantic analysis of the natural language processing in the invasion log of these three, and a semantically thing nearby became clear in invasion log than the invasion log that I collected in Honeypot which the invasion log that I collected in Honeypot which I expanded because a vector expressed a meaning of none of nothing of the command did not expand in Honeypot using the real OS.

Keywords :
1. Natural Language Processing, 2. Semantic analysis, 3. Honeypot, 4. SSH

Keio University, Faculty of Policy Management Studies
Yuta Sugafuji

目 次

図 目 次

表 目 次

第1章　序論

本章では本研究の背景，課題及び手法を提示し，本研究の概要を示す．

1.1　本研究の背景

1.1.1　通信機器の普及

PC の普及や IoT デバイスのシステム高度化により，高度な処理系を組むことが可能になった．これによりデバイス上に Linux 系などの OS が搭載された機器が広く人々に使われるようになった．また，Linux 系 OS にリモートログインする手法として SSH がある．これを用いて不正に侵入する攻撃が行われている．

1.1.2　honeypot

侵入された際に侵入者がどのような挙動をしているのかを知る手段として，Honeypot がある．これは実際の OS を用いたり，Shell の擬似的な挙動をアプリケーション上で実現し，敢えて SSH で侵入しやすいような環境を作ることで，侵入者にログイン試行に成功したと検知させ，その際に実行したコマンドのログを収集する．

1.2　本研究の問題と仮説

SSH の Honeypot は大きく二種類に分けることができ，一つは低対話型 Honeypot，もう一つは高対話型 Honeypot である．低対話型 Honeypot は実際の Shell の挙動をエミュレートしたアプリケーションである．
高対話型 Honeypot は実際の機器を設置し，その中に侵入させログを収集する．その設置時には他のホストに攻撃できないようにネットワークの設定や，root の権限が取られないように user 権限の設定を適切に行う．高対話型 Honeypot は低対話型 Honeypot と比較すると，Honeypot への侵入者が実行できるコマンドが多く，挙動も本物の OS と差異が極めて少なく，侵入先が Honeypot であると極めて検知しされにくい．そのため，高精度な攻撃ログを取得することができる．しかし，Honeypot として適切な設定を行なった OS が，設置後に発見された新たな OS の脆弱性を突かれることで，踏み台にされ他のホストに攻撃をしたりウイルスに犯されてしまうなどの危険を孕んでいる．そのため，設置コストが高く普及率も非常に低い．？

一方で低対話型 Honeypot はアプリケーションであるため，root 権限を取られるような危険が極めて少なく，アプリケーション内での脆弱性に限った問題しか存在しない．そのため設置コストが低く，比較的誰でも安全に設置できるため，普及率が高い．しかし，あくまでエミュレーションを行なったアプリケーションであるため，実際の Shell とは異なる挙動や，Honeypot に特有な挙動をすることがある．そのため，設置した Honeypot に侵入した悪意のあるユーザーに侵入先が Honeypot であると検知されてしまう可能性がある．本研究では低対話型 Honeypot に着目する．低対話型で実際の攻撃ログに近いログを収集するには，先述の Honeypot であることの検知を回避する必要がある．そこで本研究では，低対話型に実装されているコマンドの出力を，実際の Shell に近似することで検知を回避できるのではないかと考えた．

1.3 提案手法の実装

先述の Honeypot であることの検知を回避するために，本研究では低対話型 Honeypot を実際の Shell の挙動に近似するために，2 つの実験を行なった．一つは実際の Shell に実装されているが低対話型 Honeypot に実装されていないコマンドの実装した．もう一つは低対話型 Honeypot に特有の異常な挙動を修正を行った．

1.4 予備実験

本研究の予備実験として，SSH の低対話型 Honeypot に実装されていないコマンドで，悪意のある侵入者が使うコマンドを実装することで拡張を行なった低対話型 Honeypot と，素の低対話型 Honeypot でそれぞれ収集したコマンドログの比較を行なった．追加実装を施した SSH の低対話型 Honeypot の方がコマンドパターンとして多く収集できることを示した．

1.5 本研究の評価

提案手法の実装で拡張した低対話型 Honeypot と，素の Honeypot と，高対話型 Honeypot を設置し，それぞれ侵入者が実行したコマンドのログを収集し，比較を行った．収集したコマンドのログはコマンド 1 つ 1 つごとに自然言語処理の手法を用いて意味解析をし，コマンドの意味をベクトル空間上に表現した．本研究では，高対話型のログに近似することで，高度なログが収集できていると考えた．そこで，拡張した低対話型 Honeypot の侵入ログが素の Honeypot と比較して，高対話型 Honeypot の侵入ログにどれほど次元空間上で近似したのかを評価した．

1.5.1 予備実験

本研究の予備実験として，SSH の低対話型 Honeypot に実装されていないコマンドで，悪意のある侵入者が使うコマンドを実装することで拡張を行なった低対話型 Honeypot と，素の低対話型 Honeypot でそれぞれ収集したコマンドログの比較を行なった．追加実装を施した SSH の低対話型 Honeypot の方がコマンドパターンとして多く収集できることを示した．

1.6 本論文の構成

本論文における以降の構成は次の通りである．

2 章では，本研究の要素技術となる Shell と Honeypot と自然言語処理について整理する． **??**章では，本研究における問題の定義と，解決するための要件，仮説について説明する． **??**章では，本提案手法について解説する． **??**章では，本研究の事前実験や Honeypot の拡張についての実装方法や実装例について述べる． **??**章では，求められた課題に対しての評価を行い，考察する． **??**章では，関連研究を紹介し，本研究との比較を行う． **??**章では，本研究のまとめと今後の課題，展望についてまとめる．

第2章　本研究の要素技術

本章では，本研究の要素技術となる Shell と Honeypot と時系列データの扱いについて各々整理する．

2.1　Honeypot

使われているデバイスへの不正な SSH によって侵入された際，実際に攻撃が行えない環境へとフォワードし，その中で攻撃を試行させ，侵入者のログを収集する手段として Honeypot がある．SSH の Honeypot?は低対話型 Honeypot と高対話型 Honeypot の大きく二種類に分けることができる．

2.1.1　低対話型 Honeypot

SSH の低対話型 Honeypot は実際の Shell の挙動をエミュレートしたアプリケーションである．実際の Shell の挙動をエミュレートしただけのアプリケーションなので，脆弱性がアプリケーション内に限られる．そのため，root 権限を侵入者に許してしまい，踏み台にされてしまうなどの危険が極めて少ない．しかし，エミュレーションには限界があるため，コマンドやその挙動について，実際の Shell とは異なる挙動をすることがある．そのため，侵入者に侵入先が Honeypot であると検知されてしまう．検知されることで，攻撃者は実際の攻撃を行わず，本来取れるはずの攻撃ログが収集できない可能性を含んでいる．そのため，収集ログの精度に問題がある．

2.1.1.1　Kippo

Kippo は，悪意のある SSH のログイン試行者や侵入者の挙動やログを記録するために使用される Python で実装された SSH の低対話型 Honeypot である?．Kippo は前身の Kojoney?に大きく影響を受けている．ネットワークは Twisted?というフレームワークで組まれている．Kippo のプロジェクトは低対話型 Honeypot として 2009 年に登場し，Raspberry Pi?などを筆頭としたシングルボードコンピュータ?の普及と相まって広く設置された．Kippo の機能の特徴としては収集したコマンドログ を時系列データとして保存されており，”playlog”という Kippo 内にあるプログラムを実行することで，過去のコマンドログ を実際にタイピングしてるかのように出力できる．また，侵入者によってダウン

ロードされたファイルも実行ができないように保存できる．Kippo は後述の Cowrie の後継実装である．? Kippo は IoT デバイスの高度化広く設置された SSH の低対話型 Honeypot のうちの一つであったが，実装されているコマンドも 17?と少なく，また Kippo 特有の異常な挙動が存在するなどと多くの問題があった．

2.1.1.2 Cowrie

Cowrie は Python で実装された SSH の低対話型 Honeypot であり，実装は Kippo のコマンドの拡張や，攻撃者がリダイレクトでマルウェアを送り込む手法をとって送り込んだマルウェアを収集可能にしたりするなど，様々な機能を拡張したものである．Kippo 特有の異常な挙動を改善しており，実装コマンド数は 38?と Kippo より少し多くなっているものの?，Cowrie 特有の異常な挙動もまだまだ多い．

2.1.2 高対話型 Honeypot

高対話型 Honeypot は脆弱性を残した実際の OS を用いた Honeypot である．実際の OS をそのまますると，その OS から外部の他のホストへと攻撃することができてしまう．また，予期しない OS の脆弱性を突かれることで，OS 自体を完全に侵入者に制御されてしまう問題がある．そのため，Honeypot として適切な設定を行う必要がある．

2.1.2.1 Honeynet

2.1.2 で先述した通り，Honeypot で使用される OS から外部への通信で他のホストを攻撃したりするなどの攻撃を行ってしまう問題や，予期しない OS の脆弱性を突かれることで，OS 自体を完全に侵入者に制御されてしまう問題があるため，Honeypot として適切な設定を行う必要がある．そのため，Honeynet ではネットワーク全体を Honeywall という独自のファイアウォールの機能を実行する．これは Honeypot のネットワークの設定を管理するだけではなく，ネットワーク介して送信されるすべてのデータの中央集権のポイントして機能する．これによってネットワークが危険にさらされた侵入者からの攻撃から保護することが可能である．?

2.1.3 SSH の Honeypot の比較

以上をまとめた SSH の低対話型 Honeypot と SSH の高対話型 Honeypot の比較を行った表を表 2.1 に示す．

表 2.1: 種類ごとの Honeypot の比較

Honeypot の種類	設置コスト (リスク)	Honeypot であることの検知されにくさ
低対話型 Honeypot	設置コストが低い	検知されやすい
高対話型 Honeypot	設置コストが高い	検知されにくい

2.2 Shell

Shell は OS のユーザーのためにインタフェースで，カーネルのサービスへのアクセスを提供するソフトウェアである．本研究での”Shell”はコマンドラインシェルのことを指す．

2.2.1 Secure Shell

Secure Shell（セキュアシェル、SSH）は、暗号や認証の技術を利用して、安全にリモートコンピュータと通信するためのプロトコルである．パスワードなどの認証部分を含むすべてのネットワーク上の通信が暗号化される．?SSH における問題としては，通信する上での認証方法には鍵認証を推奨されているが，デフォルトではパスワード認証になっている．パスワード認証のままだとパスワードの総当たり攻撃を受けたり，パスワードが標準のままの設定であることで不正なログイン試行によって侵入を許してしまう．

2.2.2 BusyBox

BusyBox は標準 UNIX コマンドで重要な多数のプログラムを単一のバイナリファイルに含むプログラムである．BusyBox に含まれる，多数の標準 UNIX コマンドで必要とするプログラムの実行ファイルは，Linux という OS を BusyBox だけでディストリビューションできるよう，”Linux 上で最小の実行ファイル”として設計されている．一般にインストールされる実行ファイルは一部だけを実装できるように選択することができる．一般的には BusyBox のコマンドは 200 以上も用意されている．?[1]．BusyBox をインストールして実際に各コマンドを実行するためには，BusyBox 内にある各コマンドにアクセス可能なように path を通すだけで良い．

2.3 自然言語処理

人間が日常的に使っている自然言語をコンピュータに処理させる一連の技術である．本研究において，自然言語処理は意味解析のために使用した．意味解析には様々な手法があり，現在では大きくシソーラス解析とベクトル空間分析がある．

[1]今回使用した BusyBox に含まれるコマンドの数は 219

2.3.1 シソーラス解析

シソーラスとは単語を意味レベルで分解し，抽象度の高いものから低いものへと遡っていくことができ，それを体系づけた類語辞書のことである．シソーラスには様々な言語において有名な辞書が存在する．有名なシソーラスとしては Princeton University の WordNet がある．?

2.3.1.1 Wordnet

WordNet は英単語が synset と呼ばれる同義語のグループに分類され，簡単な定義や，他の同義語のグループとの関係が記述されているデータベースである．WordNet のデータベースは約 11 万 5000 の synset に分類された約 15 万語を収録し，全体で 20 万 3000 の単語と意味の組み合わせがある．?

2.3.2 ベクトル空間解析

単語の意味を表現するため，単語の文章での出現回数や，その単語の周辺の単語をマトリクス上に表現することで，その単語を数学的に解釈できるようにしている．

2.3.2.1 ベクトル空間モデル

ベクトル空間モデルとは文章を多次元空間上にベクトルとして表現し，それぞれのベクトルの比較を行うことで類似度を算出するためのモデルである．文章の類似度が高いほどベクトルの方向が近いということなので，比較した文章のベクトルのなす角が小さければ文章の類似度が高いということになる．
m 個の単語が使用されている文章 d における各単語の重要度を $w_{d1}, w_{d2}, w_{d3}, \ldots, w_{dm}$ とすると，文章 d のベクトルは以下のように表される．

$$\vec{d} = (w_{d1}, w_{d2}, w_{d3}, \ldots, w_{dm})$$

また，同様にして n 個の単語が使用されている文章 e をベクトル表現すると，

$$\vec{e} = (w_{e1}, w_{e2}, w_{e3}, \ldots, w_{en})$$

と表すことができる．したがって，$\vec{d}$ と $\vec{e}$ のなす角 θ における $\cos\theta$ は以下のように表される．

$$\cos\theta = \frac{\vec{d} \cdot \vec{e}}{|\vec{d}||\vec{e}|}$$

ベクトル化した時の単語の重要度はTF-IDFのアルゴリズム (※ 2.3.2.1.1) を用いて算出した重みを用いることで，これを表すことができる．？ 上記の例であれば，文章dにおける単語の重要度が$tf(t_1,d)\cdot idf(t_1), tf(t_2,d)\cdot idf(t_2), tf(t_3,d)\cdot idf(t_3), \ldots, tf(t_m,d)\cdot idf(t_m)$であるので，文章dのベクトルは以下のように表される．

$$\vec{d} = (tf(t_1,d)\cdot idf(t_1), tf(t_2,d)\cdot idf(t_2), tf(t_3,d)\cdot idf(t_3), \ldots, tf(t_m,d)\cdot idf(t_m))$$

また，同様にして文章eもベクトル表現すると，

$$\vec{e} = (tf(t_1,e)\cdot idf(t_1), tf(t_2,e)\cdot idf(t_2), tf(t_3,e)\cdot idf(t_3), \ldots, tf(t_n,e)\cdot idf(t_n))$$

と表すことができ，これを$\cos\theta = \frac{\vec{d}\cdot\vec{e}}{|\vec{d}||\vec{e}|}$に代入すると (※$m \leqq n$)，

$$\begin{aligned}
\cos\theta &= \frac{\vec{d}\cdot\vec{e}}{|\vec{d}||\vec{e}|} \\
&= ((tf(t_1,d)\cdot idf(t_1))(tf(t_1,e)\cdot idf(t_1)) + (tf(t_1,d)\cdot idf(t_2))(tf(t_2,e)\cdot idf(t_2)) + \ldots \\
&+ (tf(t_m,d)\cdot idf(t_m))(tf(t_m,e)\cdot idf(t_m)))\cdot \\
&\frac{1}{\sqrt{(tf(t_1,d)\cdot idf(t_1))^2 + (tf(t_2,d)\cdot idf(t_2))^2 + \ldots + (tf(t_m,d)\cdot idf(t_m))^2}}\cdot \\
&\frac{1}{\sqrt{(tf(t_1,e)\cdot idf(t_1))^2 + (tf(t_2,e)\cdot idf(t_2))^2 + \ldots + (tf(t_n,e)\cdot idf(t_n))^2}} \\
&= \frac{\sum_{i=1}^{m}((tf(t_i,d)\cdot idf(t_i))(tf(t_i,e)\cdot idf(t_i))}{\sum_{i=1}^{m}\sqrt{(tf(t_i,d)\cdot idf(t_i))^2}\sum_{i=1}^{n}\sqrt{(tf(t_i,e)\cdot idf(t_i))^2}} \qquad (2.1)
\end{aligned}$$

と表すことができる．これがベクトル空間でTF-IDFで抽出した単語の重み付けを行い，二つの文章の類似度を算出するモデルである．

2.3.2.1.1 TF-IDF

TFとはTerm Frequencyのことで，文章内での単語の出現頻度を表す．数式では以下のように表される．

$$tf(t,d) = \frac{n_{t,\,d}}{\sum_{s\ni d} n_{s,\,d}}$$

$tf(t,d)$はTFの値で，文章d内に含まれる単語tの出現頻度を表す．
$n_{t,\,d}$は文章dにおける単語tの出現回数を表す．
$\sum_{s\ni d} n_{s,\,d}$は文章dにおける全ての単語の出現回数を表す．
以上を踏まえTFの値とは，

$$\text{文章 d 内に含まれる単語 t の出現頻度} = \frac{\text{文章 d における単語 t の出現回数}}{\text{文章 d における全ての単語の出現回数}}$$

を数式で表したものである．
IDF とは Inverse Document Frequency のことで，ある単語が様々な文章においてどれほど使われているのかを表す．数式では以下のように表される．

$$idf(t) = \log \frac{N}{df(t)} + 1$$

$idf(t)$ は IDF の値で，単語 t が全文章数 N でどれほど使われているのかを表す．
N は全文章数を表す．
$df(t)$ は単語 t が出現する文章の数を表す．
以上を踏まえ IDF の値とは，

$$\text{単語 t が全文章数 N でどれほど使われているのか} = \frac{\text{全文章数}}{\text{単語 t が出現する文章の数} + 1}$$

を数式で表したものである．
このような TF の値と IDF の値を重みとすることで，文章を特徴付ける単語の抽出をするものが TF-IDF である．上記の TF と IDF の値より，if-idf の値は

$$ifidf(t, d) = tf(t, d) \cdot idf(t)$$

から算出することができる．

2.3.2.2 word2vec

word2vec は 2 層からなるニューラルネットワークである．word2vec には 2 つのアーキテクチャがあり，一つは $ContinuousSkip - gramModel$，もう一つは $ContinuousBag - of - WordsModel$ である．$ContinuousSkip - gramModel$ は入力に文章中の任意の単語を用意し，出力に文章においてその任意の単語の前後の周辺語を用意し，ニューラルネットワークに読み込ませることで第一層から第二層への重みを獲得することが目的である．$ContinuousBag - of - WordsModel$ では逆に出力に文章中の任意の単語を用意し，入力に文章においてその任意の単語の前後の周辺語を用意し，同様にしてニューラルネットワークに読み込ませることで第一層から第二層への重みを獲得することが目的である．本研究ではより精度の高い $ContinuousSkip - gramModel$（以降，$Skip - gramModel$ と呼ぶ.）を使用した．？

2.3.2.3 Continuous Skip-gram Model

$Skip-gramModel$ は先述の通り，与えられた単語に対してその周辺語を予測するためのモデルのことである．このモデルは2層からなるニューラルネットで，入力にはOne-hotベクトルを用いる．One-hot ベクトルとは $(0, 0, 0, \ldots, 1, \ldots, 0)$ のように，単語のインデックスから抽出する単語だけを 1 と表記することで表現するベクトルのことである．
入力層から隠れ層への重みは $V \times N$ のマトリクス W で表され，W の各列は単語ベクトルとなっている．隠れ層から出力層への重みはマトリクス W を転置した $N \times V$ のマトリクス W' となっている．
これをモデル化したものの出力の条件付き確率を考える．

$$p(w_O|w_I) = \frac{exp(v'^{\mathrm{T}}_{W_V} \cdot v_{w_I})}{\sum_{W_v \in V} exp(v'^{\mathrm{T}}_{W_V} \cdot v_{w_I})} \tag{2.2}$$

この w_I は入力する単語，w_O は w_I の周辺語を表す．v_{w_I} や　$v'^{\mathrm{T}}_{W_V}$ は単語を表すベクトルであり，v は入力ベクトルで v' は出力ベクトルである．コンテクストサイズとは先述したように，入力単語の周辺語をどこまでとするかのサイズのことである．$p(w_O|w_I)$ はコンテクストサイズを考慮していない確率であるが，このコンテクストサイズを考慮して先述したモデルの同時確率 $p(w_{O,1}, w_{O,2}, w_{O,3}, \ldots, w_{O,C}|w_I)$ を考える．

$$p(w_{O,1}, w_{O,2}, w_{O,3}, \ldots, w_{O,C}|w_I) = \prod_{c=1}^{C} \frac{exp(v'^{\mathrm{T}}_{W_c} \cdot v_{w_I})}{\sum_{W_v \in V} exp(v'^{\mathrm{T}}_{W_V} \cdot v_{w_I})} \tag{2.3}$$

この $p(w_{O,1}, w_{O,2}, w_{O,3}, \ldots, w_{O,C}|w_I)$ という確率を表す関数 $\prod_{c=1}^{C} \frac{exp(v'^{\mathrm{T}}_{W_V} \cdot v_{w_I})}{\sum_{W_v \in V} exp(v'^{\mathrm{T}}_{W_V} \cdot v_{w_I})}$ が最大となるような単語ベクトル v を求めることが，このモデルの目的である．
このモデルを用いてニューラルネットを構築する．先述の通り，入力層 x は One-hot ベクトルを用いる．One-hot ベクトルとは $(0, 0, 0, \ldots, 1, \ldots, 0)$ のように，単語のインデックスから抽出する単語だけを 1 と表記することで表現するベクトルのことである．
隠れ層 h は，入力層から隠れ層への重み W を入力データ x にかけたものである．したがって隠れ層 h は

$$h = Wx$$

と表すことができる．また，任意の入力 w_I $(= x_i)$ は重み W が掛けられるが，入力が One-hot ベクトルなので，w_I に対応する単語ベクトルがそのまま出力されることになる．したがって，隠れ層は

$$h = Wx_{w_I} = v_{w_I}$$

と表すことができる．
出力層 u_c は，隠れ層 h に隠れ層から出力層への重み W' が掛けられたものであるので，

$$u_c = W'h = W'v_{w_I}$$

と表すことができる．また，出力層はコンテクストサイズに応じて出力のユニット数 c が変動する．したがって，任意のユニット C における最終的な出力 $y_{c,i}$ に softmax 関数を掛けて，

$$\begin{aligned} y_{c,i} &= \frac{exp(u_{c,i})}{\sum_{v=1}^{V} exp(u_{c,v})} \\ &= \frac{exp({v'}_i \cdot v_{w_I})}{\sum_{v=1}^{V} exp({v'}_v \cdot v_{w_I})} \\ &= p(w_i|w_I) \end{aligned}$$

と表され，式 (2.2) で表した確率と同じになることが確認できる．
したがって式 (2.3) で表された同時確率 $p(w_1, w_2, w_3, \ldots, w_C|w_I)$ の最大化をするために，単語ベクトル v と単語ベクトル v' を最適化する．すなわち重み W と重み W' を最適化することを考える．word2vec では最適化のために確率的勾配降下法を用いており，目的関数として以下の式 (2.4) を定める．

$$E = -\log p(w_1, w_2, w_3, \ldots, w_C|w_I) \tag{2.4}$$

後述の損失関数の導出を円滑にするため，最大化問題から最小化問題へするために負の符号を付し，また同時確率であることから確率の値が極端に小さくなる可能性を考慮し，対数を取ることで乗法から和法へと変換することでアンダーフローを防いだ．
式 (2.4) に式 (2.3) を代入すると，

$$\begin{aligned} E &= -\log p(w_1, w_2, w_3, \ldots, w_C|w_I) \\ &= -\log \prod_{c=1}^{C} \frac{exp(u_{C,w_C})}{exp(\sum_{v=1}^{V} exp(u_{C,V})} \\ &= -\sum_{C=1}^{C} \log \frac{exp(u_{C,w_C})}{exp(\sum_{v=1}^{V} exp(u_{C,V})} \quad (\because \log_a MN = \log_a M + \log_a N) \\ &= -\sum_{C=1}^{C} (\log exp(u_{C,w_C}) - \log \sum_{V=1}^{V} exp(u_{C,V})) \quad (\because \log_a \frac{M}{N} = \log_a M - \log_a N) \\ &= -\sum_{C=1}^{C} (u_{C,w_C} - \log \sum_{V=1}^{V} exp(u_{C,V})) \quad (\because \log_e exp(x) = x) \end{aligned} \tag{2.5}$$

となる．この式 (2.5) を重み W と重み W' で偏微分し，誤差を求めることを考える．
まずは重み W' で E を偏微分し，W' の更新式を得る．また，以下の図**??**は，入力ベクトルが出力ベクトルのどこに含まれているのかを表したものである．