

# TensorFlowを使って テキストをクラス分類してみた

2016年2月28日 GDG神戸 機械学習勉強会 [AlphaGoの論文を読む会]

那由多屋 加藤 勇也



in SlideShare | Search  Upload 

Home Technology Education More Topics My Clipboards For Uploaders Collect Leads

# TensorFlowを使って テキストをクラス分類してみた

2016/2/28 GDG 神戸 機械学習勉強会 [AlphaGoの論文を読む会]  
講師 多摩 加藤 勇也



1  
◀ 1 of 50 ▶

## Recommended

Introduction to Information Architecture  
Mike Crabb

The Link Between Alcohol and Breast Cancer  
Dr. Omer Hameed

The Death of Polling?  
Ipsos MORI

Publishing Production: From the Desktop to the Cloud  
Deanta

Mobile App Prototyping for Designers  
Jono Young

How Email and Great Content Can Fuel Your Buyer's Journey

<http://goo.gl/mQHA9Y>

<http://www.slideshare.net/YuyaKato3/tensorflow-58795721>

コードはこちら：

<https://goo.gl/jqWc8V>

<https://github.com/nayutaya/20160228-gdg-kobe>

# 本日のお品書き

- 自己紹介 / 機械学習との関わり
- 取り組んだ問題
- 実装
- 評価

# 自己紹介

# 加藤 勇也

(かとう ゆうや / Yuya Kato)



なゆたや  
株式会社 那由多屋

代表取締役 (2007年～)

ソフトウェア開発 (2000年～)

大分出身・神戸在住 (2006年～)

趣味:

twitter: nayutaya  
GitHub: nayutaya  
Facebook: yuyakato1984

- ・ものづくり全般 (DIY、電子工作)
- ・3Dプリンタ (2013年～)
- ・ボルダリング (2011年～)

# 機械学習との関わり

# 『集合知プログラミング』



各種アルゴリズムをPythonコードで紹介

(例: k近傍法 / 単純ベイズ分類器 / SVM /  
ニューラルネットワーク / 焼きなまし法)

数学が苦手な僕でも、コードは読める

2008年から2009年にかけて読んだ

著者: Toby Segaran

発行: 2008年7月

ISBN: 9784873113647

一部のコードをRubyに移植した

機械学習の本はこれだけ...

# 本題

# 取り組んだ問題

TensorFlowを使って  
テキストをクラス分類してみた

# 取り組んだ問題

ある文字列を入力した時に、  
どのカテゴリに属すかを判定する



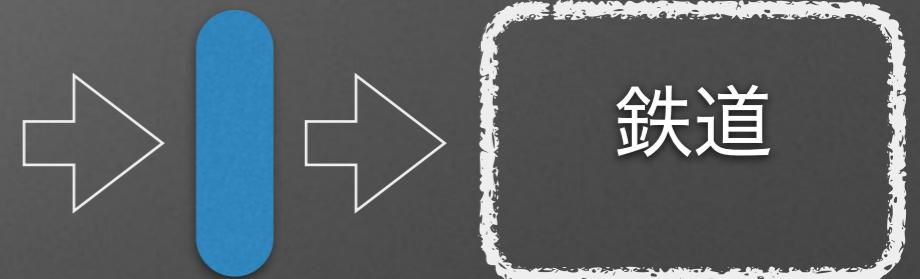
# 取り組んだ問題 - 具体的に

ある記事タイトルを入力した時に、  
鉄道に関する記事か否かを判定する

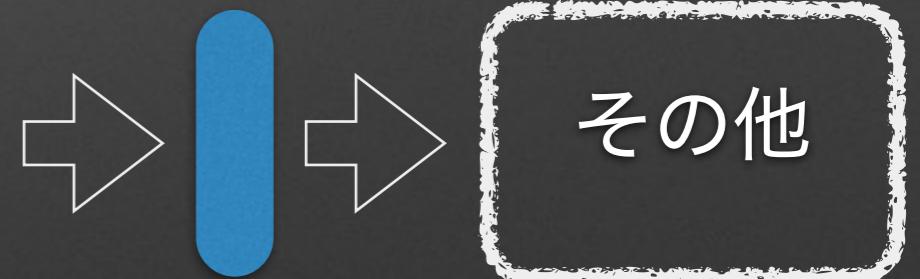


# 取り組んだ問題 - 具体例

“JR九州、半数以上が無人駅に  
3月のダイヤ改定受け”



“電気・ガス料金、4月値下げ  
大手各社、原油安受け”



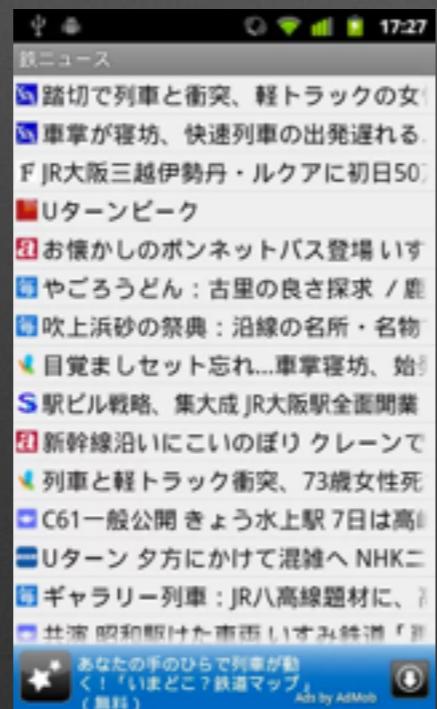
(記事タイトルはasahi.comより引用)

# なぜこの問題を選んだか？



Androidアプリ  
「鉄ニュース」  
(2010年～2016年)

手元に丁度良いデータがあったから



単純ベイズ分類器、  
多層パーセプトロンを  
Rubyで実装

# 実装

# 指針

- できるだけシンプルな方法を選択する
- KISSの原則 「Keep it simple, stupid!」  
    シンプルにしておけ！ばかやろう
- できるだけ具体的な数値を示す
- できるだけ再現性を高める
- AWS EC2上で評価
- コードは公開 (残念ながらデータは非公開)

# 説明しないこと

- 深層学習 (Deep Learning)
- RNN: Recurrent Neural Network
- RNN: Recursive Neural Network
- 高度な特徴化
  - word2vecのような分散表現  
(Word Embedding, Distributed Word Representation)

# 手段

**TensorFlow**を使って  
テキストをクラス分類してみた

TensorFlow is an Open Source Software  
Library for Machine Intelligence

[GET STARTED](#)

## About TensorFlow

TensorFlow™ is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device.



# TensorFlow

<https://www.tensorflow.org/>

Google社がオープンソースソフトウェア  
として公開（2015年9月～）

汎用計算ライブラリ

Version: r0.7

## MNIST For ML Beginners

[The MNIST Data](#)[Softmax Regressions](#)[Implementing the Regression](#)[Training](#)[Evaluating Our Model](#)

### Deep MNIST for Experts

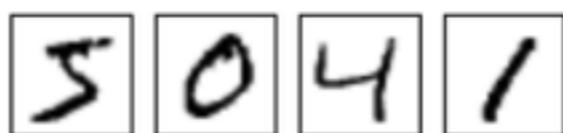
[Setup](#)[Load MNIST Data](#)[Start TensorFlow InteractiveSession](#)[Build a Softmax Regression Model](#)[Placeholders](#)[Variables](#)[Predicted Class and Cost Function](#)[Train the Model](#)[Evaluate the Model](#)[View the code on GitHub](#)

## MNIST For ML Beginners

This tutorial is intended for readers who are new to both machine learning and TensorFlow. If you already know what MNIST is, and what softmax (multinomial logistic) regression is, you might prefer this [faster paced tutorial](#). Be sure to [install TensorFlow](#) before starting either tutorial.

When one learns how to program, there's a tradition that the first thing you do is print "Hello World." Just like programming has Hello World, machine learning has MNIST.

MNIST is a simple computer vision dataset. It consists of images of handwritten digits like these:



It also includes labels for each image, telling us which digit it is. For example, the labels for the above images are 5, 0, 4, and 1.

In this tutorial, we're going to train a model to look at images and predict what digits they are. Our goal isn't to train a really elaborate model that achieves state-of-the-art performance – although we'll give you code to do that later! – but rather to dip a toe into using TensorFlow. As such, we're going to start with a very simple model, called a Softmax Regression.

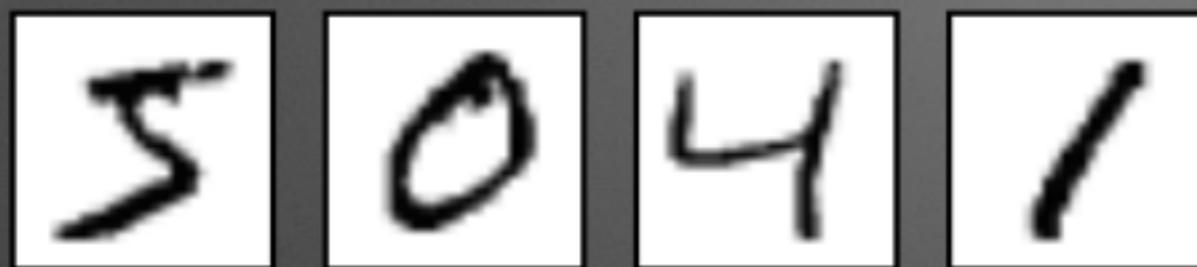
The actual code for this tutorial is very short, and all the interesting stuff happens in just three lines. However, it is

# MNIST For ML Beginners

(エムニスト)

<https://www.tensorflow.org/versions/r0.7/tutorials/mnist/beginners/index.html>

# MNIST For ML Beginners



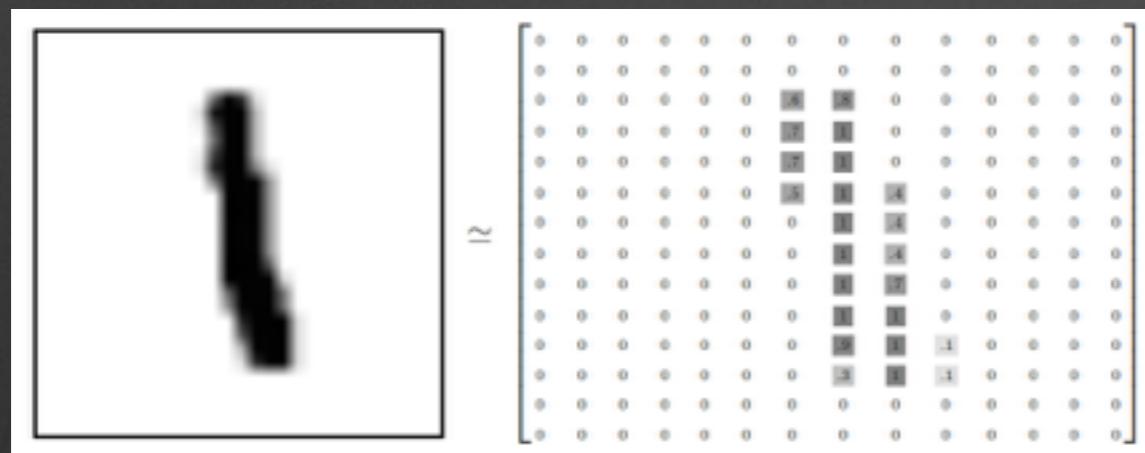
手書き文字認識

$28 * 28 \text{ px} = 784$ 次元

10クラスに分類

Softmax回帰

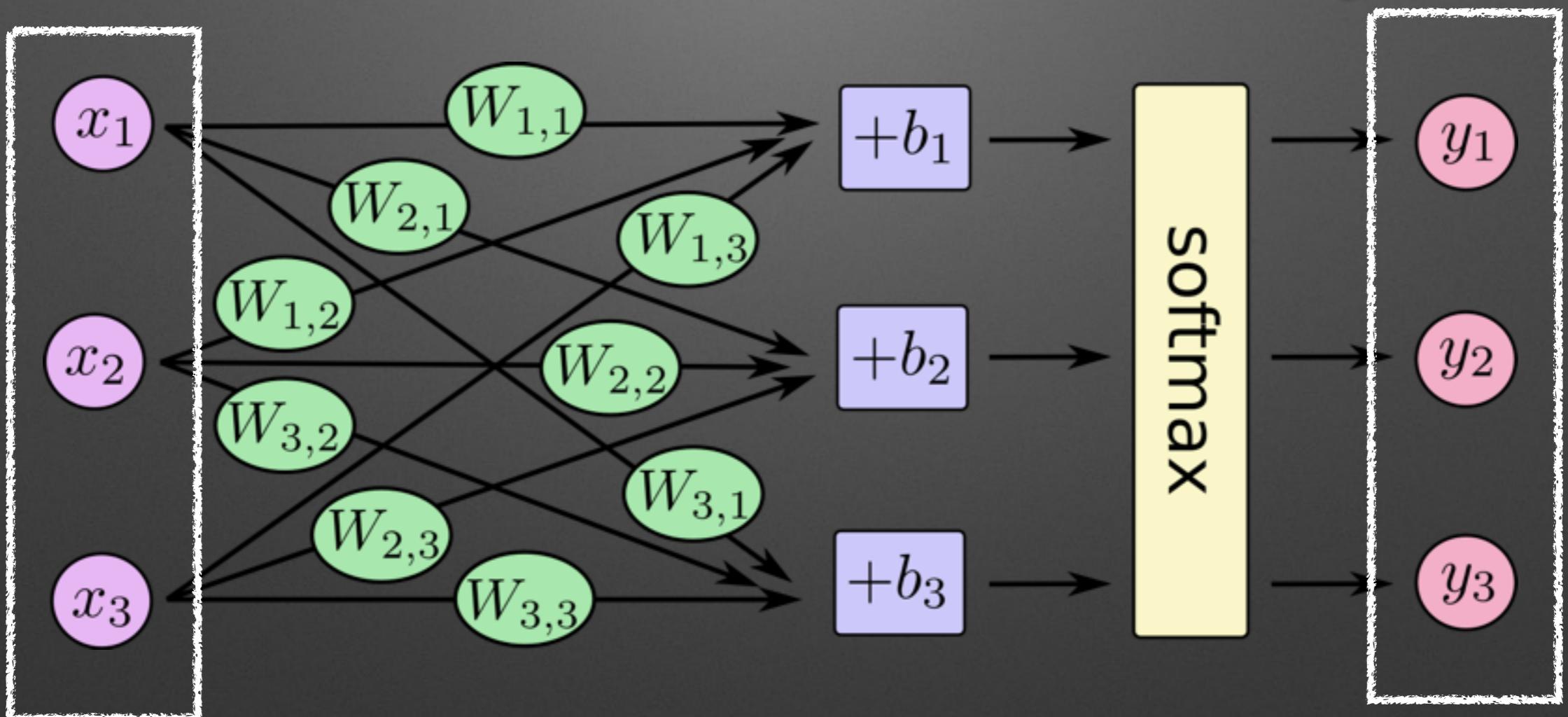
(Softmax Regression)



(画像は <https://www.tensorflow.org/versions/r0.7/tutorials/mnist/beginners/index.html> より引用)

# Softmax回帰

x: 入力    W: 重み    b: バイアス    y: 出力



ピクセル輝度

0~9

(画像は <https://www.tensorflow.org/versions/r0.7/tutorials/mnist/beginners/index.html> より引用)

# どうやってテキストを入力する？

入力は実数

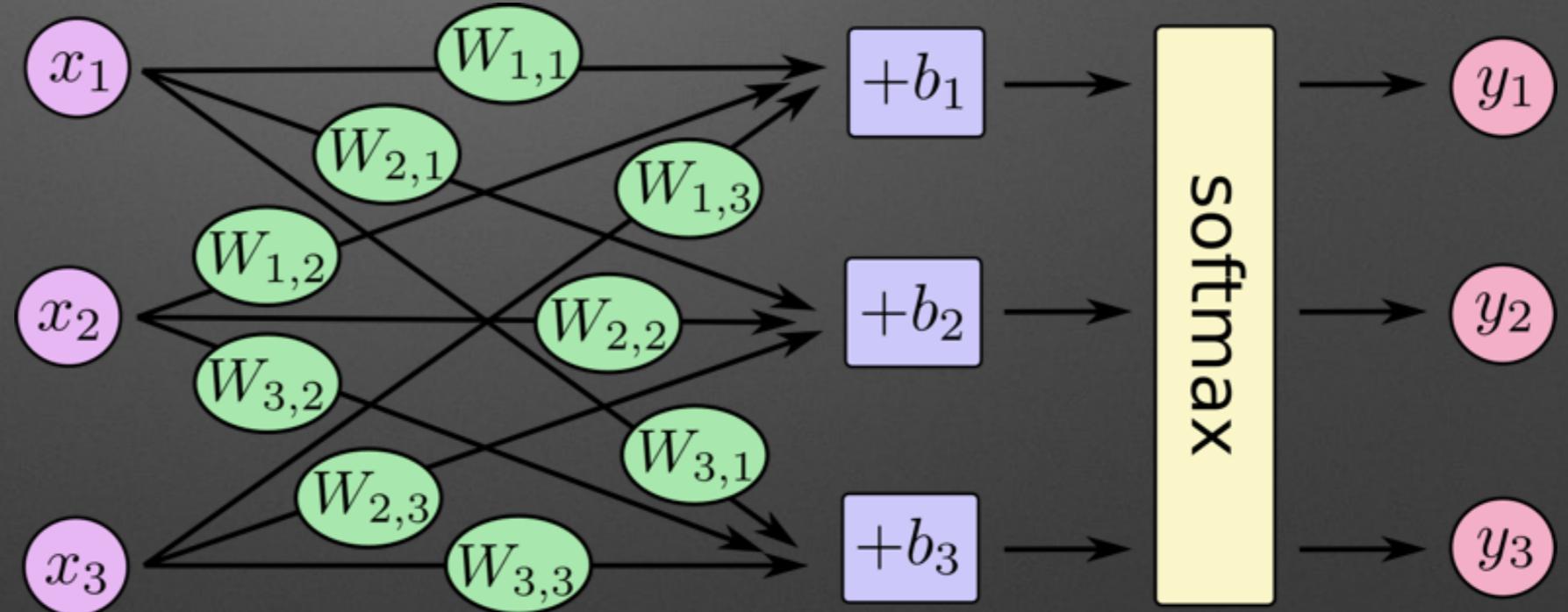


入力ノード数は学習時に決定

「This is a pen.」



テキストは可変長



(画像は <https://www.tensorflow.org/versions/r0.7/tutorials/mnist/beginners/index.html> より引用)

TensorFlowを使って  
テキストをクラス分類してみた

# 自然言語処理

- 単語の特徴化 / 文の特徴化
- 単語化
- 形態素解析
- N-gram言語モデル

# 特徴化

(特徴抽出, Feature extraction)



対象から**特徴量**を求めるこ

リンゴの場合:

丸い → 真球率

赤い → 色相・彩度

Tom Gill, CC BY-NC-ND

<https://www.flickr.com/photos/lapstrake/2899950676>

# 単語の特徴化 - One-hot表現

## (One-hot Representation)

# 辭書

「This is a pen.」

# This

→

That

# The

This

# Thou

1

0

0

1

0

□

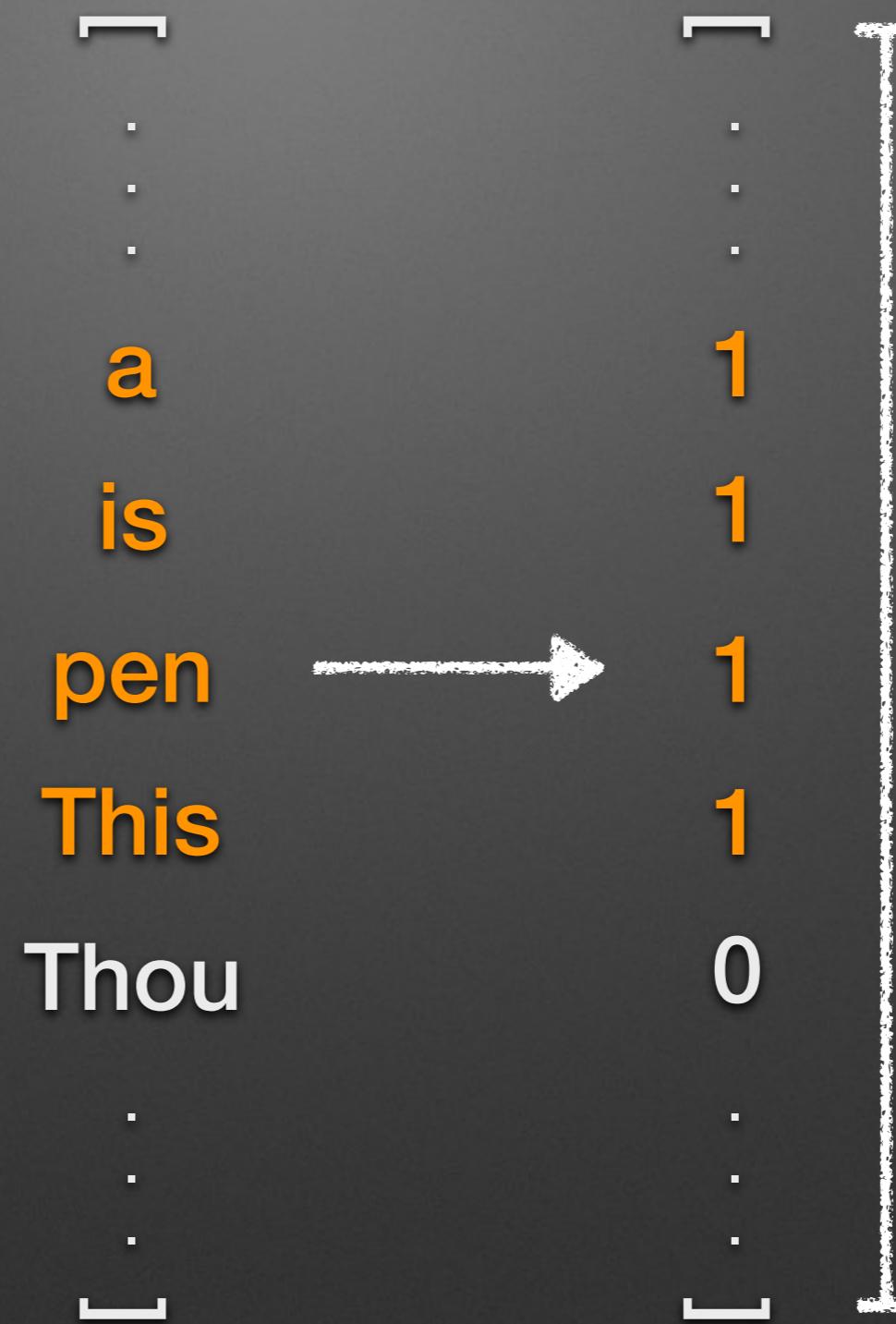
1

辞書と  
同サイズ  
の  
ベクトル

# 文の特徴化

「This is a pen.」

スペース (疎, スカスカ)  
順序が失われる  
頻度が失われる  
未知語が扱えない



辞書と  
同サイズ  
の  
ベクトル

手法の名称は不明...

# 単語化

(トークン化, Tokenize)

「This is a pen.」

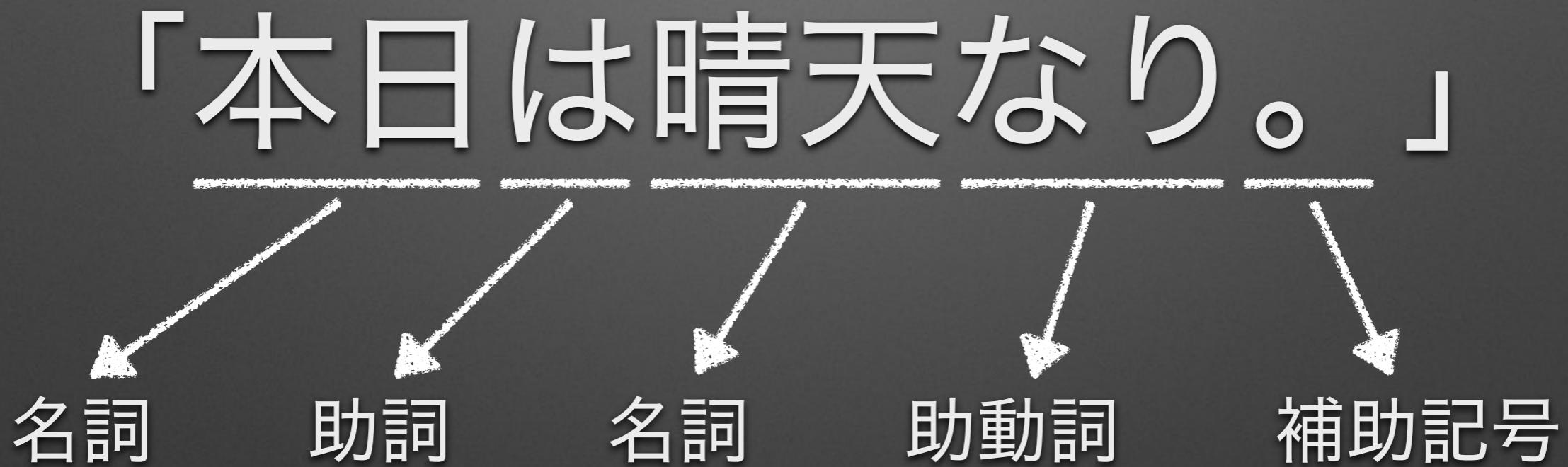
英語は分かち書き  
されている

「本日は晴天なり。」

日本語は分かち書き  
されていない

# 形態素解析

(Morphological Analysis)



今回は形態素解析ライブラリ「janome」を使用

# N-gram言語モデル

(N-gram language model)

形態素単位

「本日は晴天なり。」

1-gram (Unigram)

\_\_\_\_\_

2-gram (Bigram)

\_\_\_\_\_

3-gram (Trigram)

\_\_\_\_\_

文字単位

「本日は晴天なり。」

1-gram (Unigram)

\_\_\_\_\_

2-gram (Bigram)

\_\_\_\_\_

3-gram (Trigram)

\_\_\_\_\_

# 評価

# 実行環境

- Amazon EC2
- インスタンスタイプ: r3.xlarge (R3: メモリ最適化)
  - 仮想CPU: 4
  - メモリ: 30.5 GiB
- 料金: \$0.333 / 時間 (約38円)



# 学習データ

- 「鉄ニュース」の学習で用いた記事のタイトル
  - 鉄道: 6,321件
  - その他: 131,069件
- それぞれ1,000件(計2,000件)をテストデータとして使用
- 残りを教師データとして使用

# モデル / パラメータ (1)

- モデル: Softmax回帰
  - 入力層ノード数: 単語数 / 出力層ノード数: 2 (鉄道 / その他)
- 学習ステップ数: 500
  - ミニバッチサイズ: 100件 (鉄道: 50 / その他: 50)
- 最急降下法 (学習率: 0.01)
  - `tf.train.GradientDescentOptimizer`

# パターン (1)

文字単位



1-gram

形態素単位

2-gram

3-gram

= 計6パターン

# 評価結果 (1)

| 手法            | 前処理時間    | 学習時間     | 単語数    | 辞書サイズ   | モデルサイズ  | 精度    |
|---------------|----------|----------|--------|---------|---------|-------|
| 文字<br>1-gram  | 0.2 min  | 2.0 min  | 2,553  | 9.8 KiB | 20 KiB  | 96.8% |
| 文字<br>2-gram  | 0.3 min  | 9.7 min  | 60,728 | 398 KiB | 475 KiB | 95.7% |
| 文字<br>3-gram  | 0.4 min  | 15.0 min | 95,537 | 868 KiB | 747 KiB | 90.3% |
| 形態素<br>1-gram | 9.3 min  | 5.2 min  | 16,846 | 135 KiB | 132 KiB | 95.9% |
| 形態素<br>2-gram | 11.6 min | 10.8 min | 56,686 | 575 KiB | 443 KiB | 87.1% |
| 形態素<br>3-gram | 10.9 min | 5.8 min  | 33,030 | 430 KiB | 258 KiB | 72.9% |

処理時間はCPU時間

**TensorFlow**  
といえば

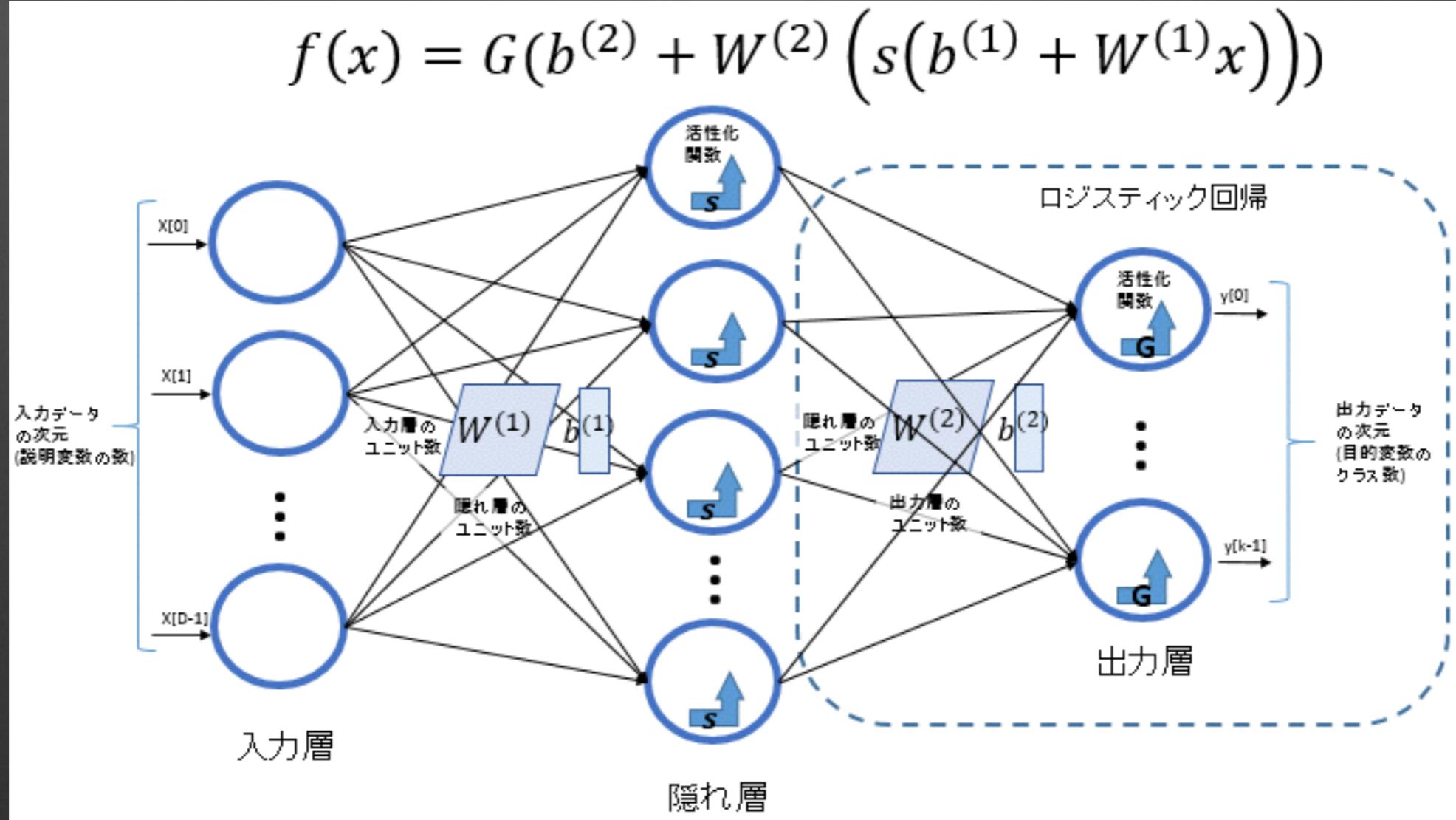
# ニューラルネットワーク

(NN: Neural Network)

# 多層パーセプトロン

(MLP: Multi-Layer Perceptron)

$$f(x) = G(b^{(2)} + W^{(2)} \left( s(b^{(1)} + W^{(1)}x) \right))$$



(画像は <http://sinhrks.hatenablog.com/entry/2014/11/30/192940> より引用)

# モデル / パラメータ (2)

- モデル: 多層パーセプトロン (入力層 - 隠れ層 - 出力層)
  - 入力層ノード数: 単語数 / 出力層ノード数: 2
  - 隠れ層ノード数: 100
  - 活性化関数: ReLU (隠れ層) / Softmax (出力層)
- 学習ステップ数: 500
- ミニバッチサイズ: 100件 (鉄道: 50 / その他: 50)

# パターン (2)

多層パーセプトロン  
文字単位2-gram



最急降下法 (学習率: 0.01)

`tf.train.GradientDescentOptimizer`

Adam法 (?) (学習率: 0.01)

`tf.train.AdamOptimizer`

= 計2パターン

# 評価結果 (2)

| 手法                    | 前処理時間   | 学習時間     | 単語数    | 辞書サイズ   | モデルサイズ  | 精度    |
|-----------------------|---------|----------|--------|---------|---------|-------|
| Softmax<br>回帰<br>(再掲) | 0.3 min | 9.7 min  | 60,728 | 398 KiB | 475 KiB | 95.7% |
| MLP<br>最急降下法          | 0.3 min | 20.6 min | 60,728 | 398 KiB | 23 MiB  | 95.3% |
| MLP<br>Adam           | 0.5 min | 21.9 min | 60,728 | 398 KiB | 70 MiB  | 95.9% |

処理時間はCPU時間

# 評価してみて

- 最もシンプルな実装（文字単位1-gram）が最高精度
- 2-gram、3-gram、MLPは500ステップでは足りなさそう
- Adam法は良さげ
- アルゴリズムの選択が難しい
- ハイパーパラメータの決定が難しい
- TensorFlow、可愛いよ

# 実装時の工夫

- ステップ毎にサマリを出力すると時間がかかる
  - 5ステップ毎に出力するようにした
- 出現頻度5回未満を削除
  - そうでないと、3-gramでMemoryError

# 今後トライしてみたいこと

- 多値クラス分類
- 単語の分散表現 → word2vec
- 単語の選択、次元削減 → TF-IDF
- 特徴化せずに処理 → RNN, LSTM, HTM
  - HTM: Hierarchical Temporal Memory
- 高速化 → GPUインスタンス

# まとめ

- 単純な問題であれば、単純なアルゴリズムでOK
- 自然言語処理の初步的な手法
- TensorFlowは素敵な道具

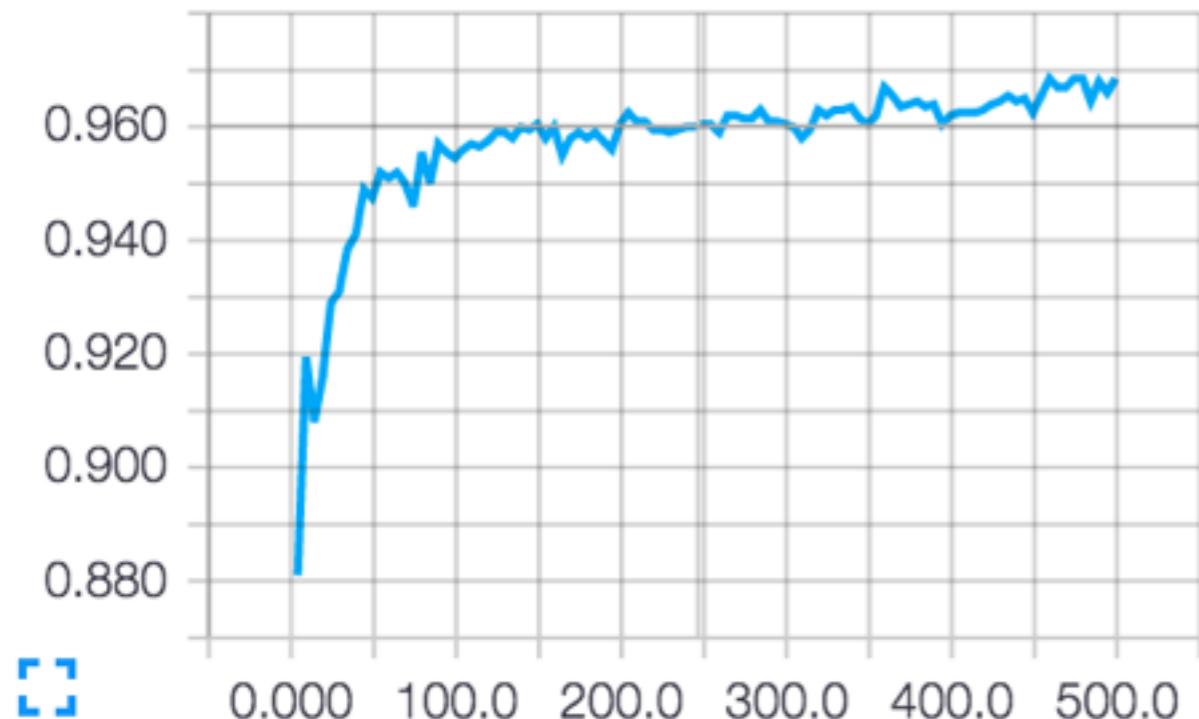




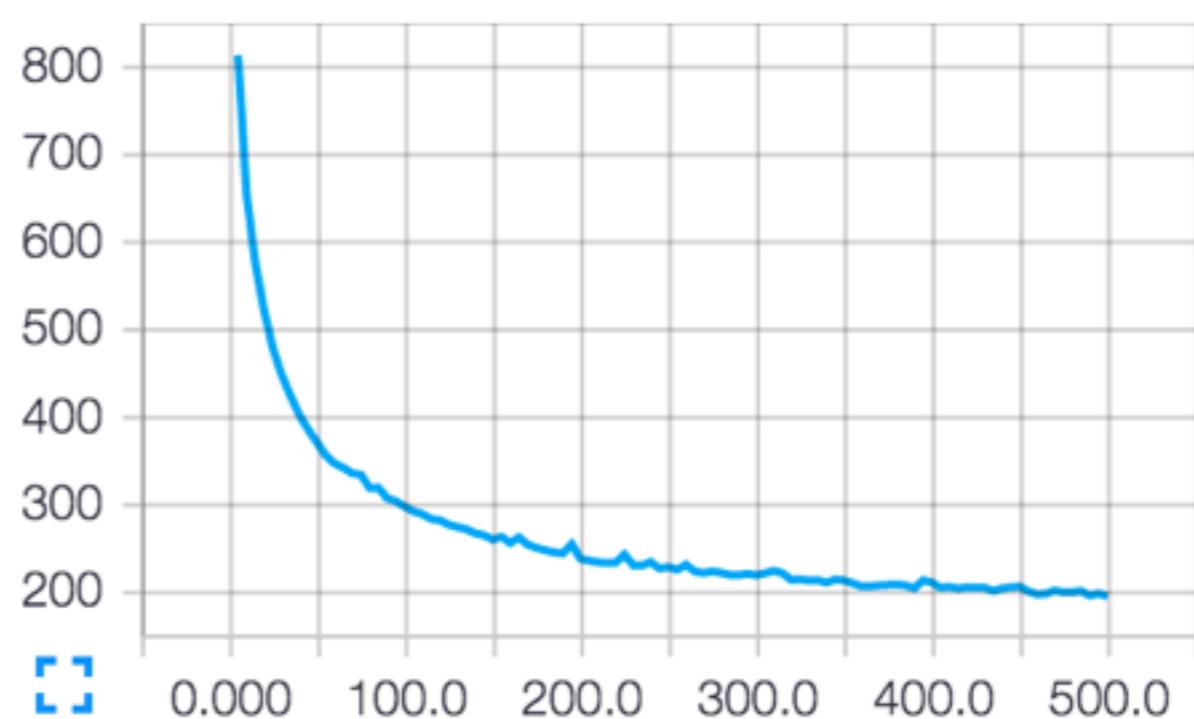
以降、参考データ

# 文字単位1-gram + Softmax回帰

accuracy

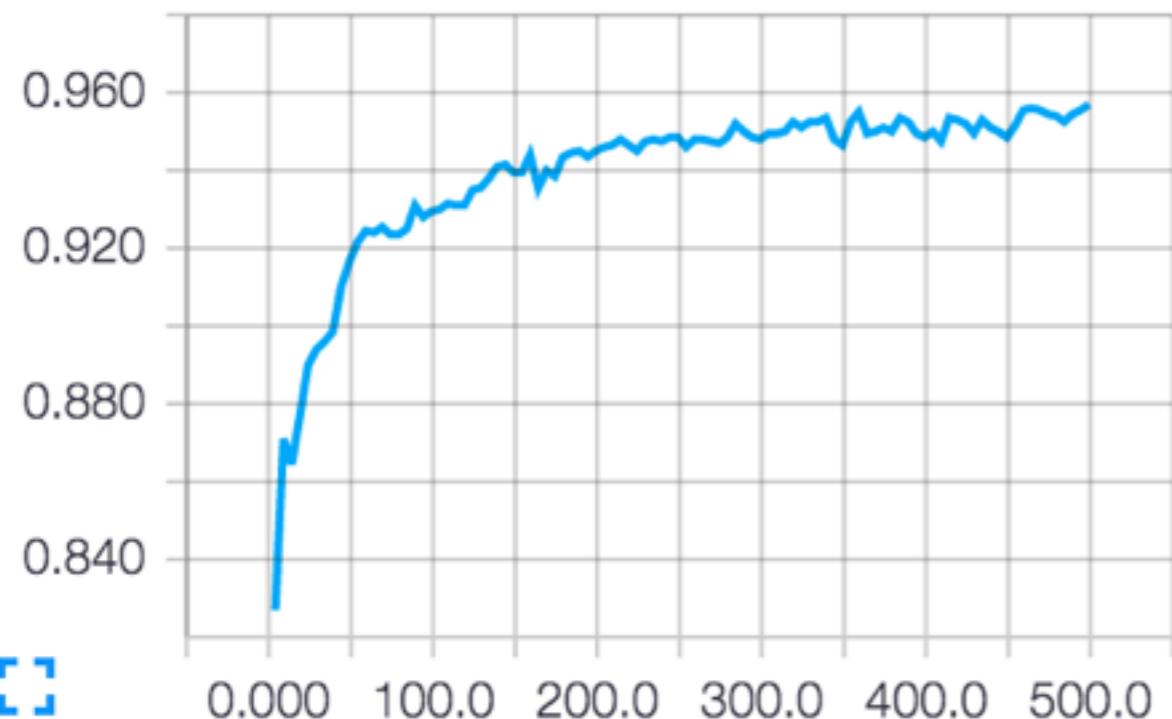


loss

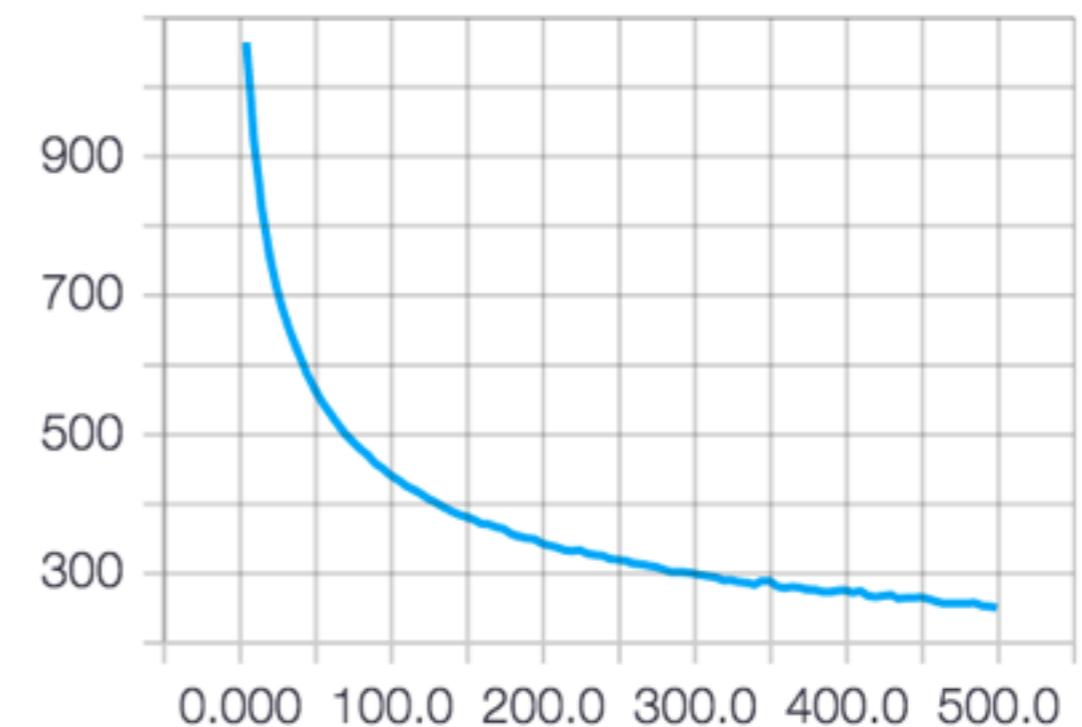


# 文字単位2-gram + Softmax回帰

accuracy

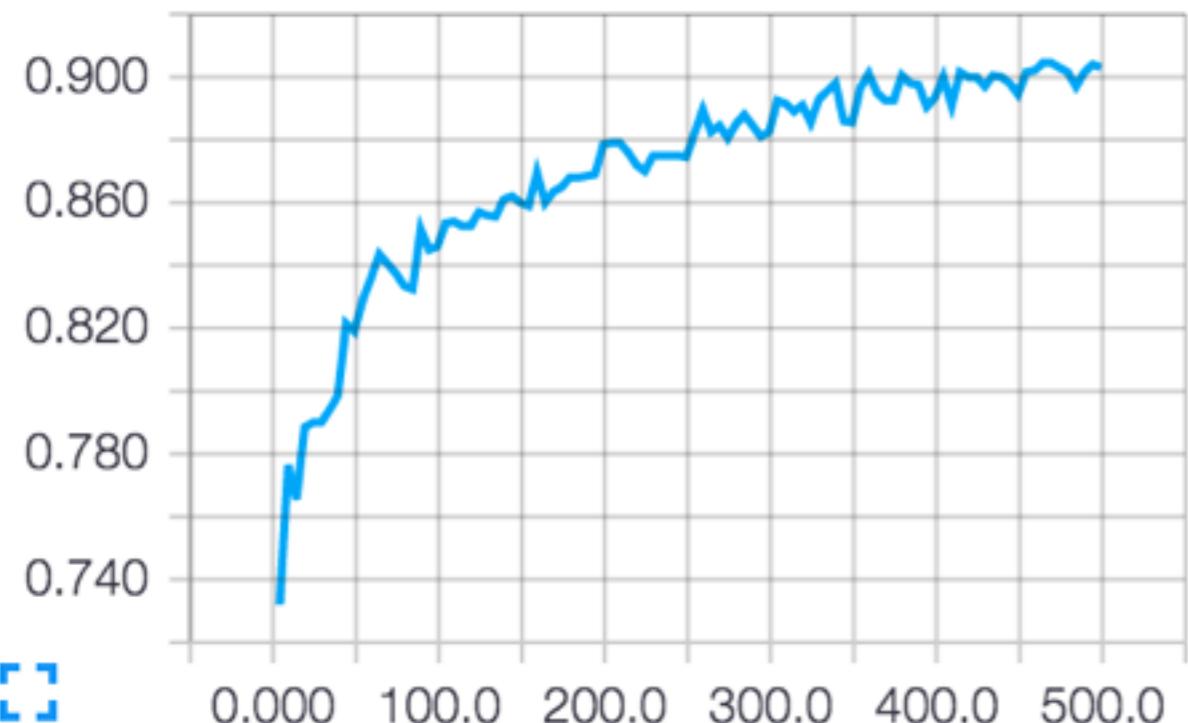


loss

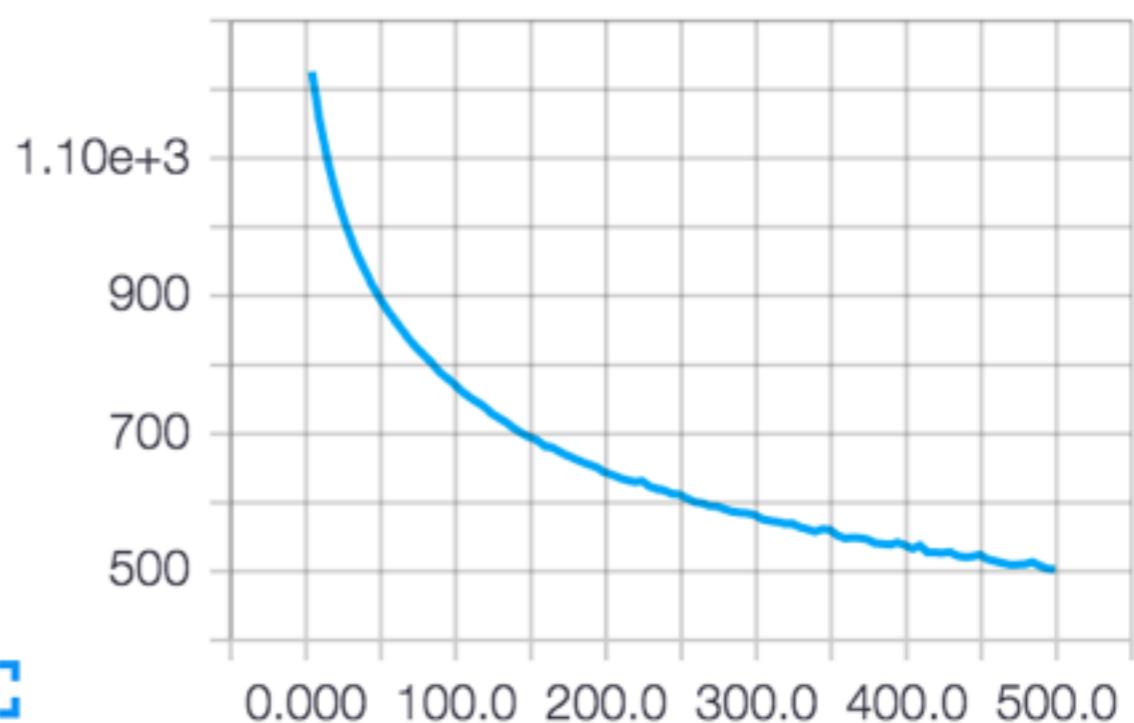


# 文字単位3-gram + Softmax回帰

accuracy

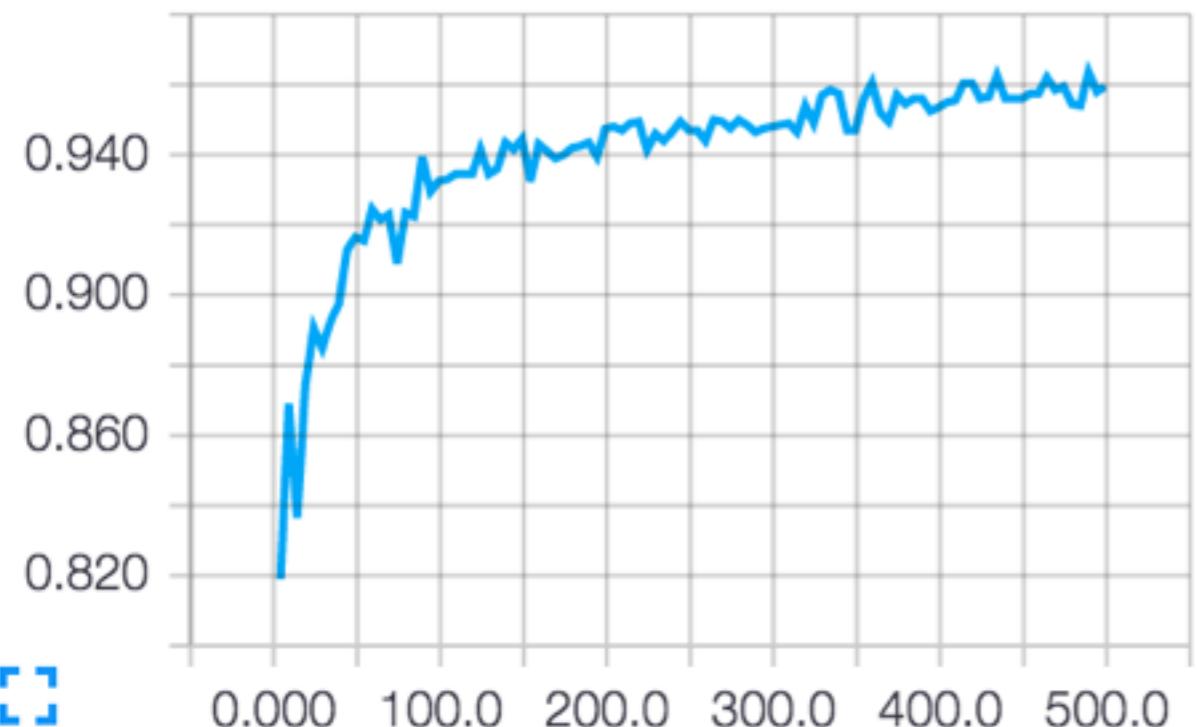


loss

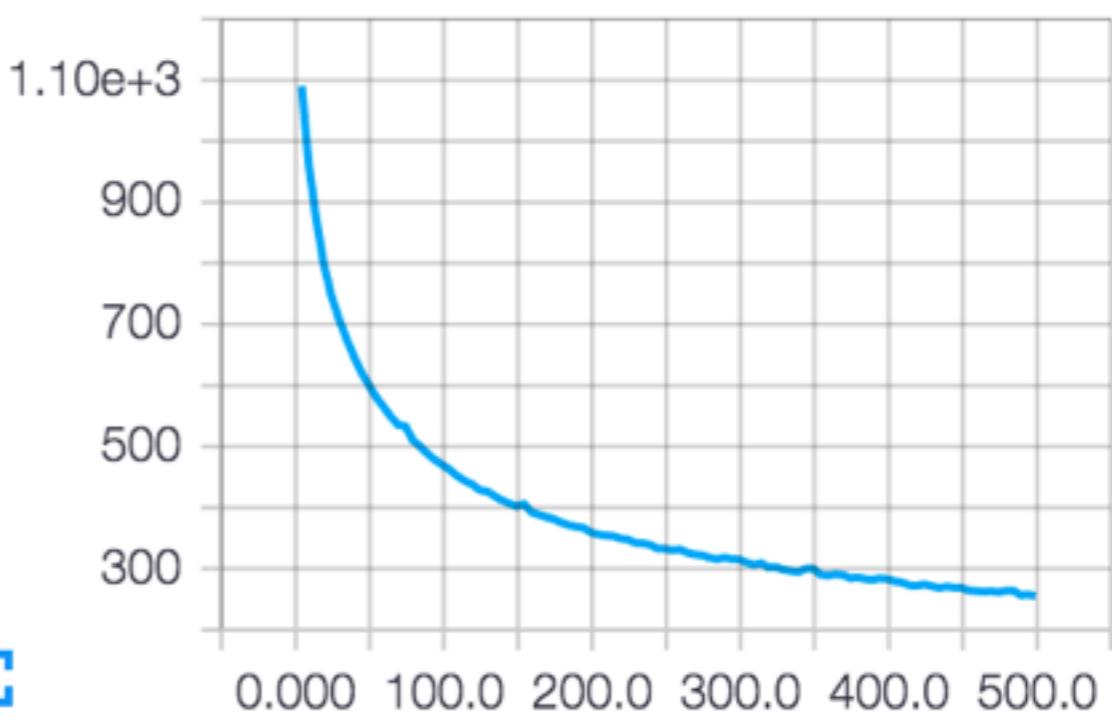


# 形態素単位1-gram + Softmax回帰

accuracy

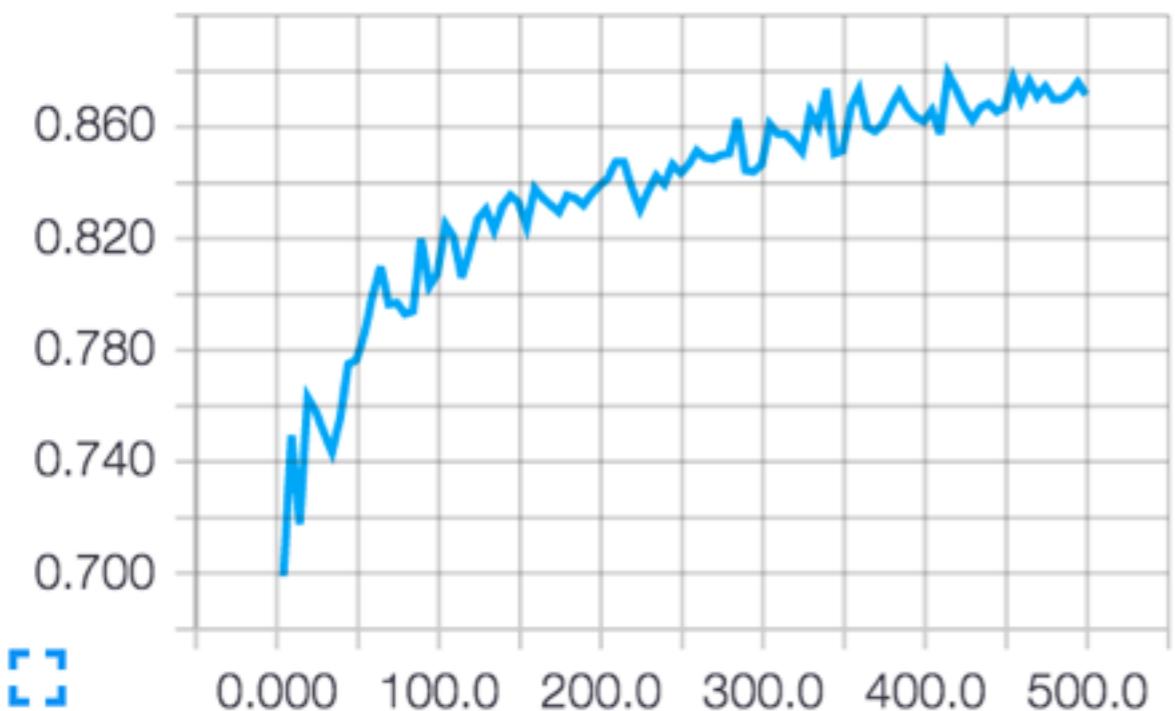


loss

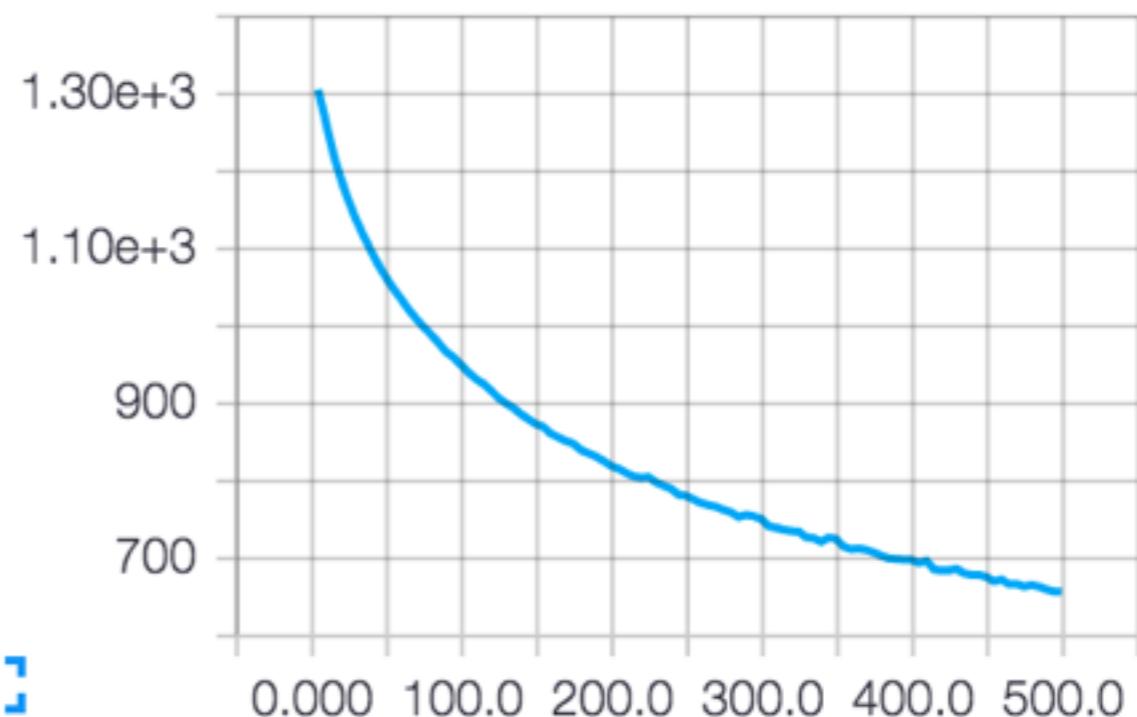


# 形態素単位2-gram + Softmax回帰

accuracy

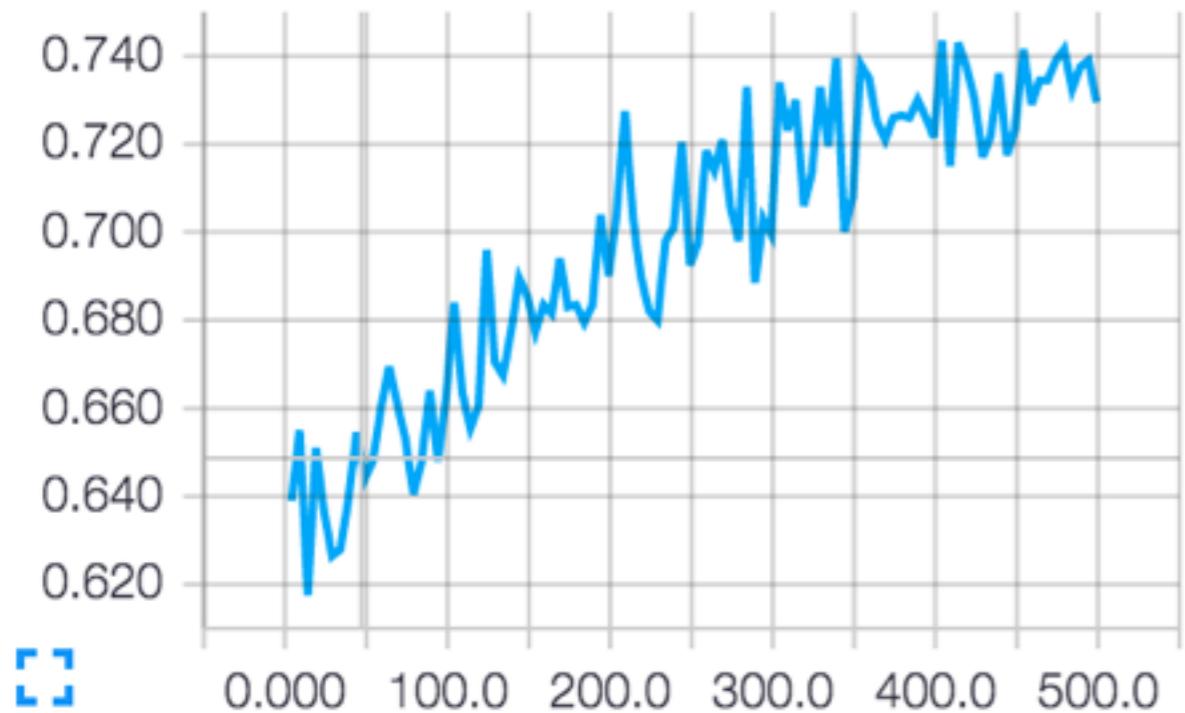


loss

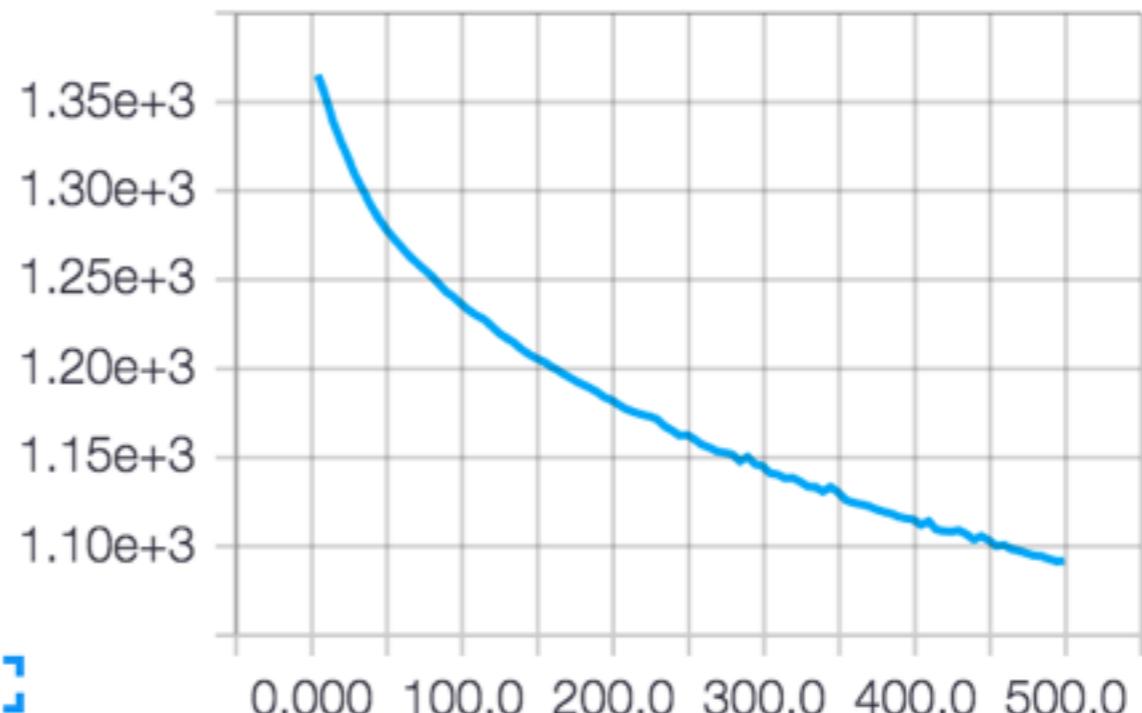


# 形態素単位3-gram + Softmax回帰

accuracy



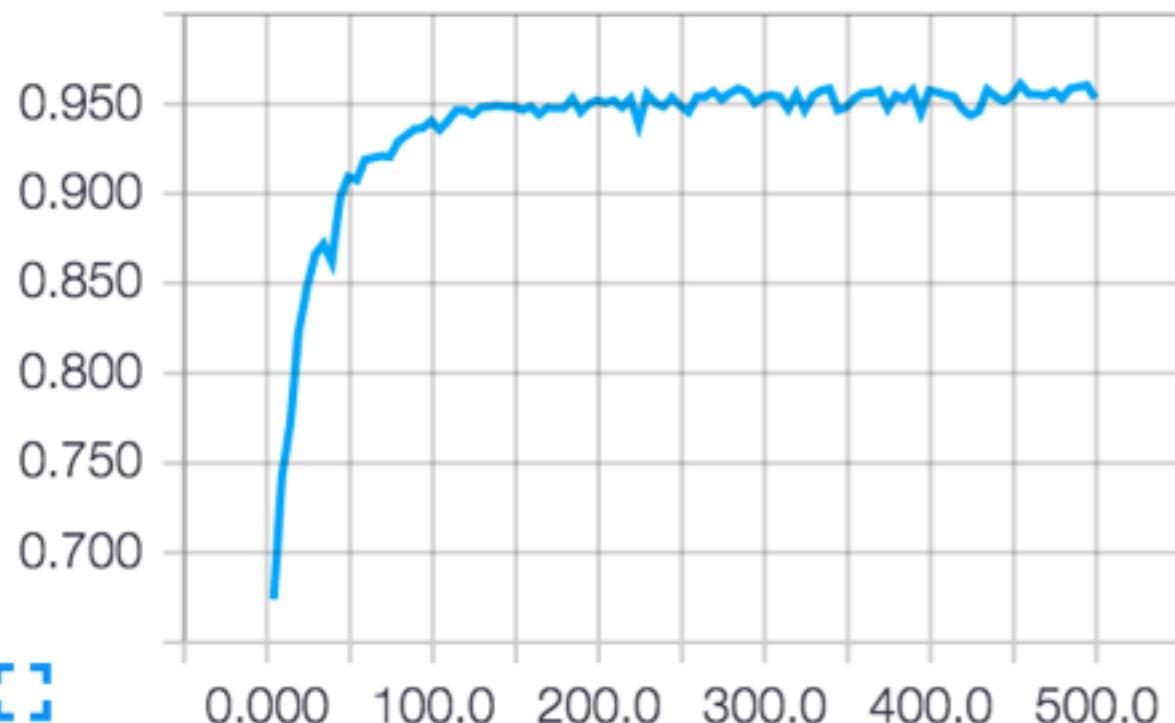
loss



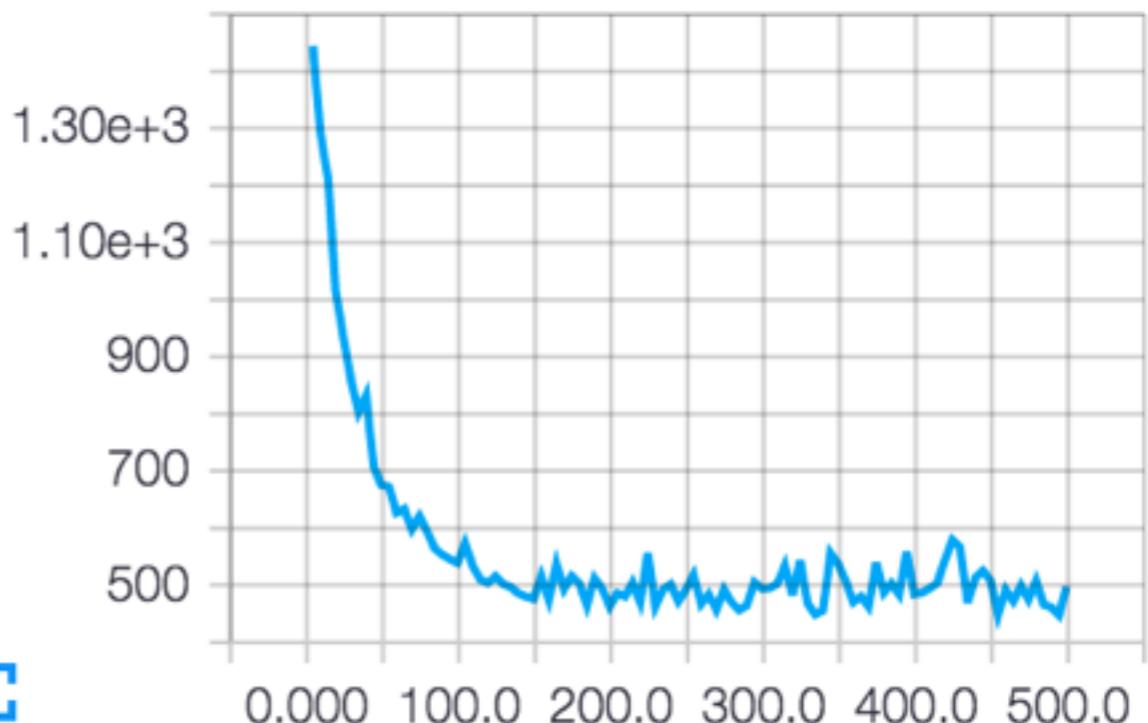
# 文字単位2-gram

+ 多層パーセプトロン + 最急降下法

accuracy



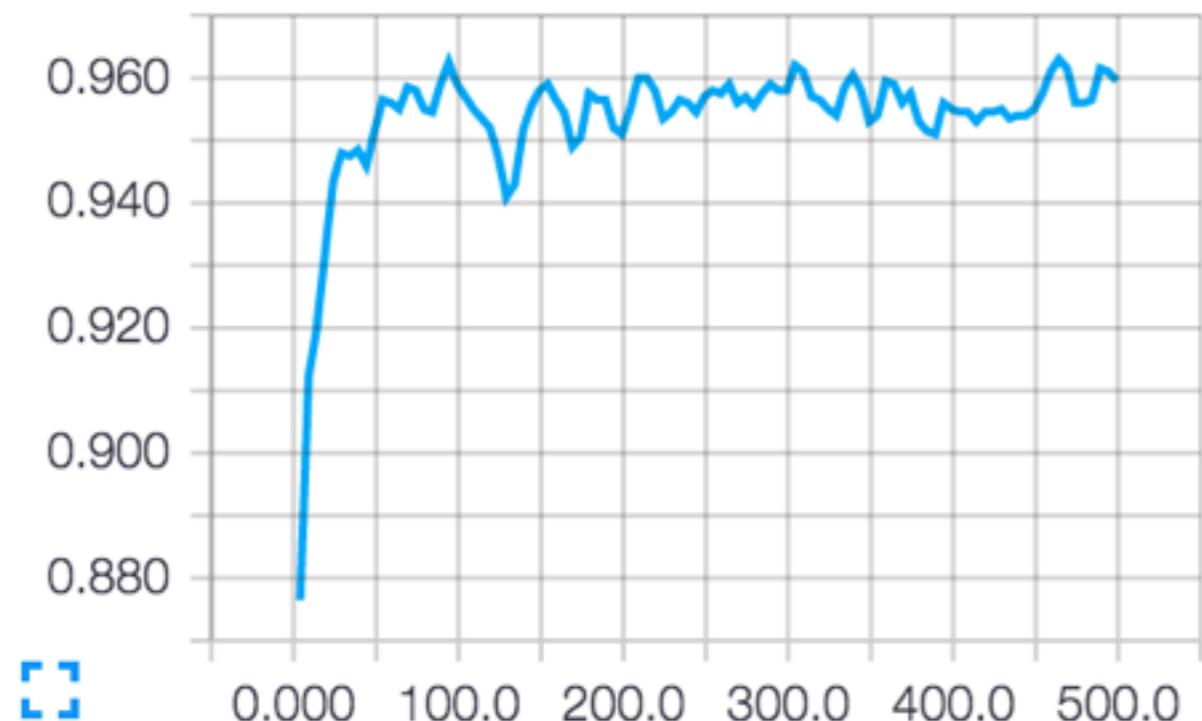
loss



# 文字単位2-gram

## + 多層パーセプトロン + Adam法

accuracy



loss

