Deep Learning for Natural Language Processing

Lecture 3 — Text classification 1: Log-linear models

Dr. Ivan Habernal

April 25, 2023

Trustworthy Human Language Technologies
Department of Computer Science
Technical University of Darmstadt

www.trusthlt.org

Feedback

Thank you for your feedback!

- Microphone
- Connection of lecture 2 to NLP
- Streaming
- Background knowledge (Lec 2 boring/hard)
- App / Live questions

Motivation

What are we going to achieve

Example task: Binary sentiment classification into positive and negative

Recall the IMDB dataset

We will learn a simple yet powerful supervised machine learning model

- Known as logistic regression, maximum entropy classifier
- In fact, it is a single-layer neural network
- An essential important building block of deep neural networks

The challenges of NLP

Ambiguity and variability of human language

Highly ambiguous

Example

Compare “I ate pizza with friends” to “I ate pizza with olives”

Highly variable

Example

The core message of “I ate pizza with friends” can be expressed as “friends and I shared some pizza”

Humans — great users of language, very poor at formally understanding and describing rules that govern language

Y. Goldberg (2017). *Neural Network Methods for Natural Language Processing*. Morgan & Claypool

Supervised machine learning to save us...

The best known set of methods for dealing with language data → supervised machine learning algorithms

- ML attempts to infer patterns and regularities from a set of pre-annotated input–output pairs
- ML excels at problem domains where a good set of rules is very hard to define but annotating the expected output for a given input is relatively simple

...but language is even more challenging

Natural language exhibits properties that make it even more challenging for ML

1. Discrete
2. Compositional
3. Sparse

Language is symbolic and discrete

Basic elements of written language: **characters**

Characters form **words** that denote objects, concepts, events, actions, and ideas

Characters and words are discrete symbols

- Words such as “hamburger” or “pizza” each evoke in us a certain mental representations
- But they are distinct symbols, whose meaning is external to them, to be interpreted in our heads
- No inherent relation between “hamburger” and “pizza” can be inferred from the symbols or letters themselves

Characters and words are discrete symbols

Compare that to concepts such as **color** (in machine vision), or acoustic signals — these concepts are **continuous**

- Colorful image to gray-scale image using a simple mathematical operation
- We can to compare two different colors based on inherent properties such as hue and intensity

This cannot be easily done with words

There is no simple operation to move from the word “red” to the word “pink” without using a large lookup table or a dictionary

Language is compositional

Letters $\rightarrow$ words $\rightarrow$ phrases $\rightarrow$ sentences

The meaning of a phrase can be larger than the meaning of the individual words, and follows a set of intricate rules

Example

Multi-word expressions (“New York”, “look something up”)
Idioms (“kick the bucket”, “blue chip”)

To interpret a text, we need to work beyond the level of letters and words, and look at long sequences of words such as sentences, or even complete documents.

Data sparseness

Combinations of words to form meanings $\rightarrow \infty$

- We could never enumerate all possible valid sentences

No clear way of generalizing from one sentence to another, or defining the similarity between sentences, that does not depend on their meaning which is unobserved to us

Challenging when learning from examples

Even with a huge example set we are very likely to observe events that never occurred in the example set and that are very different

Towards supervised machine learning on text data

Mathematical notation

Scalars, vectors, matrices

Lowercase letters represent scalars: x, y, b

Bold lowercase letters represent vectors: $\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{b}$

Bold uppercase letters represent matrices: $\boldsymbol{W}, \boldsymbol{X}$

Indexing

$[.]$ as the index operator of vectors and matrice

$\boldsymbol{b}_{[i]}$ is the i-th element of vector $\boldsymbol{b}$

$\boldsymbol{W}_{[i,j]}$ is the i-th row, j-th column of matrix $\boldsymbol{W}$

Notation

Sequences

$\boldsymbol{x}_{1:n}$ is a sequence of vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$

$[\boldsymbol{v}_1; \boldsymbol{v}_2]$ is vector concatenation

Note! We use vectors as *row* vectors

$$\boldsymbol{x} \in \mathbb{R}^d$$

Example $d = 5$:

$$\boldsymbol{x} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \end{pmatrix}$$

which is simply a list (1-d array) of numbers $(1, 2, 3, 4, 5)$

Multiplication example

$$\boldsymbol{x} \in \mathbb{R}^{d_{in}} \qquad \boldsymbol{W} \in \mathbb{R}^{d_{in} \times d_{out}} \qquad \boldsymbol{b} \in \mathbb{R}^{d_{out}}$$

Example: $\boldsymbol{y} = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}$, $d_{in} = 3$, $d_{out} = 2$

$$\begin{pmatrix} \boldsymbol{x}_{[1]} & \boldsymbol{x}_{[2]} & \boldsymbol{x}_{[3]} \end{pmatrix} \begin{pmatrix} \boldsymbol{W}_{[1,1]} & \boldsymbol{W}_{[1,2]} \\ \boldsymbol{W}_{[2,1]} & \boldsymbol{W}_{[2,2]} \\ \boldsymbol{W}_{[3,1]} & \boldsymbol{W}_{[3,2]} \end{pmatrix} + \begin{pmatrix} \boldsymbol{b}_{[1]} & \boldsymbol{b}_{[2]} \end{pmatrix} = \begin{pmatrix} \boldsymbol{y}_{[1]} & \boldsymbol{y}_{[2]} \end{pmatrix}$$

Multiplication simplified with dot product $\boldsymbol{u} \cdot \boldsymbol{v} = \sum_i \boldsymbol{u}_{[i]} \boldsymbol{v}_{[i]}$

Example: $\boldsymbol{y} = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}$, $d_{in} = 3$, $d_{out} = 1$

$\boldsymbol{x} \in \mathbb{R}^{d_{in}}$ $\quad\boldsymbol{W} \in \mathbb{R}^{d_{in} \times d_{out}}$ $\quad\boldsymbol{b} \in \mathbb{R}^{d_{out}}$

$$\begin{pmatrix} \boldsymbol{x}_{[1]} & \boldsymbol{x}_{[2]} & \boldsymbol{x}_{[3]} \end{pmatrix} \begin{pmatrix} \boldsymbol{W}_{[1,1]} \\ \boldsymbol{W}_{[2,1]} \\ \boldsymbol{W}_{[3,1]} \end{pmatrix} + b = y$$

Equivalent dot product: $y = \boldsymbol{x} \cdot \boldsymbol{w} + b$, $d_{in} = 3$, $d_{out} = 1$

$\boldsymbol{x} \in \mathbb{R}^{d_{in}}$ $\quad\boldsymbol{w} \in \mathbb{R}^{d_{out}}$ $\quad b \in \mathbb{R}$

$$\begin{pmatrix} \boldsymbol{x}_{[1]} & \boldsymbol{x}_{[2]} & \boldsymbol{x}_{[3]} \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{w}_{[1]} & \boldsymbol{w}_{[2]} & \boldsymbol{w}_{[3]} \end{pmatrix} + b = y$$

Towards supervised classification setup

We often restrict ourselves to search over specific families of functions, e.g., the space of all linear functions with d_{in} inputs and d_{out} outputs

- By restricting to a specific hypothesis class, we are injecting the learner with **inductive bias** (a set of assumptions about the form of the desired solution)

High-dimensional linear functions

Function $f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R}^{d_{out}}$

$$f(\boldsymbol{x}) \text{ or } f(\boldsymbol{x}; \underbrace{\boldsymbol{W}, \boldsymbol{b}}_{\text{Explicit parameters}}) = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}$$

where $\boldsymbol{x} \in \mathbb{R}^{d_{in}}$ $\quad$ $\boldsymbol{W} \in \mathbb{R}^{d_{in} \times d_{out}}$ $\quad$ $\boldsymbol{b} \in \mathbb{R}^{d_{out}}$

Vector $\boldsymbol{x}$ is the **input**, matrix $\boldsymbol{W}$ and vector $\boldsymbol{b}$ are the **parameters** — typically denoted $\Theta = \boldsymbol{W}, \boldsymbol{b}$

Goal of learning

Set the values of the parameters $\boldsymbol{W}$ and $\boldsymbol{b}$ such that the function behaves as intended on a collection of input values $\boldsymbol{x}_{1:k} = \boldsymbol{x}_1, \dots, \boldsymbol{x}_k$ and the corresponding desired outputs $\boldsymbol{y}_{1:k} = \boldsymbol{y}_1, \dots, \boldsymbol{y}_k$

Binary classification

Function $f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \to \mathbb{R}$

$$f(\boldsymbol{x}) \text{ or } f(\boldsymbol{x}; \underbrace{\boldsymbol{w}, \boldsymbol{b}}_{\text{Explicit parameters}}) = \boldsymbol{x} \cdot \boldsymbol{w} + b$$

However, for binary text classification

- Our input is in the form of a natural language text
- Our labels are two categories, e.g., positive and negative

Let's start with the labels

Very easy: Just arbitrarily map the categories into 0 and 1 (e.g., negative = 0, positive = 1)

Numerical representation of natural language text

Goal: Transform text into a fixed-size vector of real numbers

What's our setup:

$$f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \rightarrow \mathbb{R} \qquad f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{w} + b$$

What we need:

$$\boldsymbol{x} \in \mathbb{R}^{d_{in}}$$

What we have:

One of my favorite movies ever,The Shawshank Redemption is a modern day classic as it tells the story of two inmates who become friends and find solace over the years in which this movie takes place.Based on a Stephen King novel, ...

What is a “word”?

A matter of debate among linguists, answer not always clear

Very simplistic definition: words are sequences of letters separated by whitespace

But: `dog`, `dog?`, `dog.`, and `dog)` would be different words

Better: words separated by whitespace or punctuation

A process called **tokenization** splits text into tokens based on whitespace and punctuation

- English: the job of the tokenizer is quite simple
- Hebrew, Arabic: sometimes without whitespace
- Chinese: no whitespaces at all

Y. Goldberg (2017). *Neural Network Methods for Natural Language Processing*. Morgan & Claypool

Tokens

Symbols `cat` and `Cat` have the same meaning, but are they the same word?

Something like `New York`, is it two words, or one?

- We distinguish between words and tokens
- We refer to the output of a tokenizer as a token, and to the meaning-bearing units as words

Keep in mind

We use the term **word** very loosely, and take it to be interchangeable with **token**.

In reality, the story is more complex than that.

Vocabulary

We build a fix-sized static **vocabulary** (e.g., by tokenizing training data)

- Typical sizes: 20,000 – 100,000 words

Each word has a unique fixed index

$$V = \left(\text{a}_1 \quad \text{abandon}_2 \quad \ldots \quad \text{cat}_{852} \quad \ldots \quad \text{zone}_{2,999} \quad \text{zoo}_{3,000}\right)$$

(Averaged) Bag-of-words

$$\boldsymbol{x} = \frac{1}{|D|} \sum_{i=1}^{|D|} \boldsymbol{x}^{D_{[i]}}$$

$D_{[i]}$ – word in doc D at position i, $\boldsymbol{x}^{D_{[i]}}$ – one-hot vector

Example: a cat sat $\rightarrow$ **a, cat, sat**

$$V = \begin{pmatrix} \text{a}_1 & \text{abandon}_2 & \ldots & \text{cat}_{852} & \ldots & \text{zone}_{2,999} & \text{zoo}_{3,000} \end{pmatrix}$$

$$\text{a} = \boldsymbol{x}^{D_{[1]}} = \begin{pmatrix} 1_1 & 0_2 & 0_3 & \ldots & 0_{2,999} & 0_{3,000} \end{pmatrix}$$

$$\text{cat} = \boldsymbol{x}^{D_{[2]}} = \begin{pmatrix} 0_1 & \ldots & 1_{852} & \ldots & 0_{2,999} & 0_{3,000} \end{pmatrix}$$

$$\text{sat} = \boldsymbol{x}^{D_{[3]}} = \begin{pmatrix} 0_1 & \ldots & 1_{2,179} & \ldots & 0_{2,999} & 0_{3,000} \end{pmatrix}$$

Averaged bag-of-words example: $\boldsymbol{x} \in \mathbb{R}^{3,000}$

Example: a cat sat → `a, cat, sat`

$$\text{a} = \boldsymbol{x}^{D[1]} = \begin{pmatrix} 1_1 & 0_2 & 0_3 & \ldots & 0_{2,999} & 0_{3,000} \end{pmatrix}$$

$$\text{cat} = \boldsymbol{x}^{D[2]} = \begin{pmatrix} 0_1 & \ldots & 1_{852} & \ldots & 0_{2,999} & 0_{3,000} \end{pmatrix}$$

$$\text{sat} = \boldsymbol{x}^{D[3]} = \begin{pmatrix} 0_1 & \ldots & 1_{2,179} & \ldots & 0_{2,999} & 0_{3,000} \end{pmatrix}$$

$$\boldsymbol{x} = \frac{1}{|D|} \sum_{i=1}^{|D|} \boldsymbol{x}^{D[i]}$$

$$= \begin{pmatrix} 0.33_1 & 0_2 & \ldots & 0_{851} & 0.33_{852} & 0_{853} & \ldots & 0.33_{2,179} & \ldots & 0_{3,000} \end{pmatrix}$$

Out-of-vocabulary (UNK) tokens

Words in a language are very unevenly distributed (Zipf's law)

- There is always a large 'tail' of rare words

When building the vocabulary, use the most frequent words, all others represented by an unknown token (**UNK** or **OOV**)

Example vocabulary, most common 3,000 words and UNK

$$V = \left(\text{a}_1 \quad \text{abandon}_2 \quad \ldots \quad \text{zone}_{2,999} \quad \text{zoo}_{3,000} \quad \text{UNK}_{3,001} \right)$$

- In machine translation, how to translate the **UNK** word?

P. Koehn (2020). *Neural Machine Translation.* (not freely available). Cambridge University Press

Subword units: Byte-pair encoding

1. The words in the corpus are split into characters (marking original spaces with a special space character) — this is the initial vocabulary V
2. The most frequent pair of characters is merged and added to V
3. Repeat 2 for a fixed given number of times
4. Each of these steps increases V by one, beyond the original inventory of single characters

When done over large corpora with multiple languages and writing systems, BPE prevents OOV!

Byte-pair encoding example on a toy corpus (part 1)

```
t h i s _ f a t _ c a t _ w i t h _ t h e _
h a t _ i s _ i n _ t h e _ c a v e _ o f _
t h e _ t h i n _ b a t
```

Most frequent: `t h` (6 times), merge into a single token

```
th i s _ f a t _ c a t _ w i th _ th e _ h a
t _ i s _ i n _ th e _ c a v e _ o f _ th e
_ th i n _ b a t
```

Most frequent: `a t` (4 times), merge into a single token

```
th i s _ f at _ c at _ w i th _ th e _ h at
_ i s _ i n _ th e _ c a v e _ o f _ th e _
th i n _ b at
```

Byte-pair encoding example on a toy corpus (part 2)

`th i s _ f at _ c at _ w i th _ th e _ h at _ i s _ i n _ th e _ c a v e _ o f _ th e _ th i n _ b at`

Most frequent: `th e` (3 times), merge into a single token

`th i s _ f at _ c at _ w i th _ the _ h at _ i s _ i n _ the _ c a v e _ o f _ the _ th i n _ b at`

$V =$ {t, h, i, s, _, f, a, c, w, e, n, v, o, f, b, th, at, the}

At the end of this process, the most frequent words will emerge as single tokens, while rare words consist of still unmerged subwords

SentencePiece: A variant of byte pair encoding

Byte-pair example. Word splits indicated with @@.

```
[the] [relationship] [between] [Obama]
[and] [Net@@] [any@@] [ahu] [is] [not]
[exactly] [friendly] [.]
```

SentencePiece escapes the whitespace with _ and tokenizes the input into an arbitrary subword sequence

SentencePiece example of ”Hello world.”

```
[Hello] [_wor] [ld] [.]
```

Lossless tokenization — all the information to reproduce the normalized text is preserved

T. Kudo and J. Richardson (2018). “SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Brussels, Belgium: Association for Computational Linguistics, pp. 66–71

Recap: Transform text into a fixed-size vector of real numbers

What's our setup:

$$f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \to \mathbb{R} \qquad f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{w} + b$$

What we need:

$$\boldsymbol{x} \in \mathbb{R}^{d_{in}}$$

What we have:

One of my favorite movies ever,The Shawshank Redemption is a modern day classic ...

Simple solution:

- Bag-of-words (tokenized), $d_{in} = |V|$

Binary text classification

Binary text classification

Binary classification as a function

Linear function and its derivatives

We have this linear function

$$f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \to \mathbb{R} \qquad f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{w} + b = \boldsymbol{x}_{[1]} \boldsymbol{w}_{[1]} + \ldots + \boldsymbol{x}_{[d_{in}]} \boldsymbol{w}_{[d_{in}]} + b$$

Derivatives wrt. parameters w and b

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{w}_{[i]}} = \boldsymbol{x}_{[i]} \qquad \frac{\mathrm{d}f}{\mathrm{d}b} = 1$$

Non-linear mapping to $[0, 1]$

We have this linear function

$$f(\boldsymbol{x}) : \mathbb{R}^{d_{in}} \to \mathbb{R} \qquad f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{w} + b$$

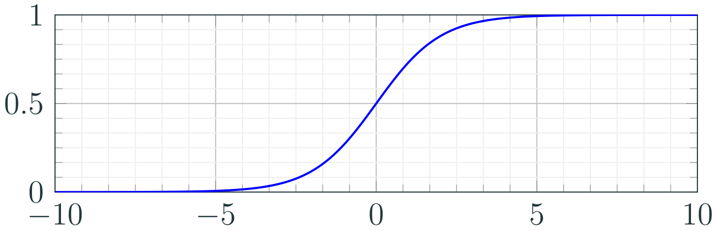
which has an unbounded range $(-\infty, +\infty)$

However, each example's label is $y \in \{0, 1\}$

Sigmoid (logistic) function

Sigmoid function $\sigma(t) : \mathbb{R} \to \mathbb{R}$

$$\sigma(t) = \frac{\exp(t)}{\exp(t) + 1} = \frac{1}{1 + \exp(-t)}$$



Odd function, range of $\sigma(t) \in [0, 1]$,

Sigmoid $\sigma(t) = \frac{1}{1+\exp(-t)}$

Derivative of sigmoid wrt. its input

$$\begin{aligned}\frac{\mathrm{d}\sigma}{\mathrm{d}t} &= \frac{\exp(t) \cdot (1 + \exp(t)) - \exp(t) \cdot \exp(t)}{(1 + \exp(t))^2} \\ &= \ldots \\ &= \sigma(t) \cdot (1 - \sigma(t))\end{aligned}$$

Our binary text classification function

Linear function through sigmoid — log-linear model

$$\hat{y} = \sigma(f(\boldsymbol{x})) = \frac{1}{1 + \exp(-(\boldsymbol{x} \cdot \boldsymbol{w} + b))}$$

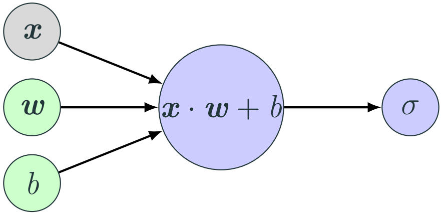


Figure 1: Computational graph; green circles are trainable parameters, gray are inputs

Decision rule of log-linear model

Log-linear model $\hat{y} = \sigma(f(\boldsymbol{x})) = \frac{1}{1+\exp(-(\boldsymbol{x} \cdot \boldsymbol{w} + b))}$

- Prediction = 1 if $\hat{y} > 0.5$
- Prediction = 0 if $\hat{y} < 0.5$

Natural interpretation: Conditional probability of prediction = 1 given the input $\boldsymbol{x}$

$$\begin{aligned} \sigma(f(\boldsymbol{x})) &= \Pr(\text{prediction} = 1 | \boldsymbol{x}) \\ 1 - \sigma(f(\boldsymbol{x})) &= \Pr(\text{prediction} = 0 | \boldsymbol{x}) \end{aligned}$$

Binary text classification

Finding the best model’s parameters

The loss function

Loss function: Quantifies the loss suffered when predicting $\hat{y}$ while the true label is y for a single example. In binary classification:

$$L(\hat{y}, y) : \mathbb{R}^2 \to \mathbb{R}$$

Given a labeled training set $(\boldsymbol{x}_{1:n}, \boldsymbol{y}_{1:n})$, a per-instance loss function L and a parameterized function $f(\boldsymbol{x}; \Theta)$ we define the corpus-wide loss with respect to the parameters Θ as the average loss over all training examples

$$\mathcal{L}(\Theta) = \frac{1}{n} \sum_{i=1}^{n} L(f(\boldsymbol{x}_i; \Theta), y_i)$$

Training as optimization

$$\mathcal{L}(\Theta) = \frac{1}{n} \sum_{i=1}^{n} L(f(\boldsymbol{x}_i; \Theta), y_i)$$

The training examples are fixed, and the values of the parameters determine the loss

The goal of the training algorithm is to set the values of the parameters Θ, such that the value of $\mathcal{L}$ is minimized

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmin}} \, \mathcal{L}(\Theta) = \underset{\Theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L(f(\boldsymbol{x}_i; \Theta), y_i)$$

Binary cross-entropy loss (logistic loss)

$$L_{\text{logistic}} = -y \log \hat{y} - (1-y)\log(1-\hat{y})$$

Partial derivative wrt. input $\hat{y}$

$$\frac{\mathrm{d}L_{\text{Logistic}}}{\mathrm{d}\hat{y}} = -\left(\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right) = -\frac{y-\hat{y}}{\hat{y}(1-\hat{y})}$$

Full computational graph

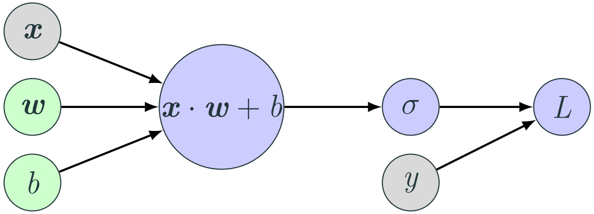


Figure 2: Computational graph; green circles are trainable parameters, gray are constant inputs

How can we minimize this function?

- Recall Lecture 2: (a) Gradient descent and (b) backpropagation

(Online) Stochastic Gradient Descent

```
1: function SGD($f(\boldsymbol{x}; \Theta)$, $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, $(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$, $L$)
2:     while stopping criteria not met do
3:         Sample a training example $\boldsymbol{x}_i$, $\boldsymbol{y}_i$
4:         Compute the loss $L(f(\boldsymbol{x}_i; \Theta), \boldsymbol{y}_i)$
5:         $\hat{\boldsymbol{g}} \leftarrow$ gradient of $L(f(\boldsymbol{x}_i; \Theta), \boldsymbol{y}_i)$ wrt. $\Theta$
6:         $\Theta \leftarrow \Theta - \eta_t \hat{\boldsymbol{g}}$
7:     return $\Theta$
```

Loss in line 4 is based on a **single training example** $\rightarrow$ a rough estimate of the corpus loss $\mathcal{L}$ we aim to minimize

The noise in the loss computation may result in inaccurate gradients

Minibatch Stochastic Gradient Descent

```
1: function MBSGD($f(\boldsymbol{x}; \Theta)$, $(\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$, $(\boldsymbol{y}_1, \dots, \boldsymbol{y}_n)$, $L$)
2:     while stopping criteria not met do
3:         Sample $m$ examples $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), \dots (\boldsymbol{x}_m, \boldsymbol{y}_m)\}$
4:         $\hat{\boldsymbol{g}} \leftarrow 0$
5:         for $i = 1$ to $m$ do
6:             Compute the loss $L(f(\boldsymbol{x}_i; \Theta), \boldsymbol{y}_i)$
7:             $\hat{\boldsymbol{g}} \leftarrow \hat{\boldsymbol{g}} +$ gradient of $\frac{1}{m} L(f(\boldsymbol{x}_i; \Theta), \boldsymbol{y}_i)$ wrt. $\Theta$
8:         $\Theta \leftarrow \Theta - \eta_t \hat{\boldsymbol{g}}$
9:     return $\Theta$
```

Properties of Minibatch Stochastic Gradient Descent

The minibatch size can vary in size from $m = 1$ to $m = n$

Higher values provide better estimates of the corpus-wide gradients, while smaller values allow more updates and in turn faster convergence

Lines 6+7: May be easily parallelized

Recap

Take aways

- Tokenization is tricky
- Simplest representation of text as bag-of-word features
- Binary classification as a linear function of words and a sigmoid
- Binary cross-entropy (logistic) loss
- Training as minimizing the loss using minibatch SGD and backpropagation

License and credits

Credits

Ivan Habernal