Pembentukan *Vector Space Model* Bahasa Indonesia Menggunakan Metode *Word to Vector*

Yulius Denny Prabowo[1], Tedi Lesmana Marselino[2], Meylisa Suryawiguna[3]
[1,2,3]Program Studi Informatika, Fakultas Industri Kreatif, Institut Teknologi dan Bisnis Kalbis
Jl. Pulomas Selatan Kav.22, Jakarta Timur 13210, DKI Jakarta, Indonesia
Email: [1]yulius.prabowo@kalbis.ac.id, [2]tedi.lesmana@kalbis.ac.id, [3]agathameylissa@gmail.com

Abstract. *Extracting information from a large amount of structured data requires expensive computing. The Vector Space Model method works by mapping words in continuous vector space where semantically similar words are mapped in adjacent vector spaces. The Vector Space Model model assumes words that appear in the same context, having the same semantic meaning. In the implementation, there are two different approaches: counting methods (eg: Latent Semantic Analysis) and predictive methods (eg Neural Probabilistic Language Model). This study aims to apply Word2Vec method using the Continuous Bag of Words approach in Indonesian language. Research data was obtained by crawling on several online news portals. The expected result of the research is the Indonesian words vector mapping based on the data used.*
Keywords: *vector space model, word to vector, Indonesian vector space model.*

Abstrak. *Ekstraksi informasi dari sekumpulan data terstruktur dalam jumlah yang besar membutuhkan komputasi yang mahal. Metode Vector Space Model bekerja dengan cara memetakan kata-kata dalam ruang vektor kontinu dimana kata-kata yang serupa secara semantis dipetakan dalam ruang vektor yang berdekatan. Metode Vector Space Model mengasumsikan kata-kata yang muncul pada konteks yang sama, memiliki makna semantik yang sama. Dalam penerapannya ada dua pendekatan yang berbeda yaitu: metode yang berbasis hitungan (misal: Latent Semantic Analysis) dan metode prediktif (misalnya Neural Probabilistic Language Model). Penelitian ini bertujuan untuk menerapkan metode Word2Vec menggunakan pendekatan Continuous Bag Of Words model dalam Bahasa Indonesia. Data penelitian yang digunakan didapatkan dengan cara crawling pada berberapa portal berita online. Hasil penelitian yang diharapkan adalah pemetaan vektor kata Bahasa Indonesia berdasarkan data yang digunakan.*
Kata Kunci: *vector space model, word to vector, vektor kata bahasa Indonesia.*

1. Pendahuluan

Peminatan dalam hal pemrosesan bahasa dangan menggunakan data teks dalam bidang ilmu komputer saat ini menunjukkan peningkatan, meskipun demikian pemrosesan bahasa ini memerlukan komputasi pengolahan data yang cukup berat dikarenakan data teks yang tidak terstruktur serta mempunyai informasi yang kaya, dalam arti mempunyai fitur yang banyak dan berdimensi tinggi. Sistem pemrograman bahasa alami secara tradisional memperlakukan kata-kata sebagai sebuah simbol atomik yang diskrit, kata "anjing" misalnya dapat digambarkan dalam ID765 dan kata "kucing" digambarkan dalam ID 647. Pemberian kode terhadap setiap kata ini dilakukan secara acak, akibatnya hubungan antar ID tersebut tidak dapat dipahami. Sehingga secara statistik sulit untuk mendapatkan informasi yang mungkin ada diantara kedua simbol individu (ID anjing dan ID kucing) seperti misalnya: keduanya adalah hewan, keduanya berkaki empat, keduanya hewan peliharaan dan lain sebagainya.

Pemberian kode yang unik dan diskrit ini juga menimbulkan permasalahan *data sparsity*. Karena itu untuk mendapatkan model statistik yang dapat menggambarkan hubungan antara simbol individu dibutuhkan data latih dan data uji yang amat banyak, konsekuensinya adalah komputasi yang amat boros memori diperlukan untuk mendapatkan model statistik dan

komputasi tersebut harus dijalankan untuk setiap ID yang ingin diperiksa. Representasi *vector* digunakan untuk mengatasi permasalahan ini.

Metode klasik yang banyak digunakan oleh para peneliti dalam untuk mengekstrak fitur dalam sebuah kalimat atau dokumen adalah metode *bag-of-word* (BOW) seperti TF-IDF atau *Count Vectorize.* Selain metode BOW peneliti juga banyak menggunakan metode *Latent Dirchlet Analytic* (LDA) atau *Latent Semantic Analysis* (LSA). Meskipun banyak digunakan metode ini mempunyai beberapa kerterbatasan misalnya memerlukan *vector* dalam dimensi tinggi atau fitur yang tersebar/jarang. Pada tahun 2013 Mikolov et all [1] mempopulerkan sebuah metode baru bernama *Word embedding*, yang menghasilkan fitur yang padat pada dimensi *vector* yang rendah. Eksperimen yang banyak dilakukan penelitian lain menunjukkan bahwa *word embedding* mampu menghasilkan fitur *vector* yang lebih baik.

Istilah *Word Embbeding* sendiri sebenarnya diperkenalkan pertama kali oleh Bengio et all [2]. Dua belas tahun sebelum Mikolov mempublikasikan penelitiannya, Bengio mempublikasikan sebuah makalah untuk menangani permodelan Bahasa, dalam paper inilah ide awal *word embedding* muncul. Saat itu Bengio menyebut proses ini sebagai *"learning a distributed representation for word"*. Pada tahun 2008, Ronan dan Jason [3] memperkenalkan konsep *pre-trained model* dan menunjukkan bahwa konsep ini merupakan pendekatan yang luar biasa untuk masalah NLP. Konsep *pre-trainned model* inilah yang nantinya digunakan Mikolov ketika menghasilkan sebuah p*re-trained model* yang diberi nama *Word2Vec*.

Representasi *vector* merupakan sebuah metode yang bekerja dengan cara memetakan kata-kata dalam ruang vektor kontinu, dimana kata-kata yang serupa secara *semantic*, akan dipetakan dalam ruang vektor yang berdekatan. *Word2Vec* sebagai metode terbaru *Vector Space Model* yang biasa digunakan untuk mengolah data masukan dalam jumlah yang besar merupakan model prediktif yang sangat efisien untuk mempelajari pola dari data mentah seperti yang dikatakan Faruqui [4], Kiros [5] dan Wolf [6] dalam penelitiannya. Dalam pembentukan metode *Word2Vec* ada dua pendekatan yang digunakan yaitu *Continuous Bag-of-Words* model (CBOW) dan *Skip-gram* model seperti yang dilakukan Mikolov et all [7]. Penelitian ini akan membahas tentang penerapan metode *Word2Vec* oleh Mikolov menggunakan data teks Bahasa Indonesia.

Tujuan penelitian ini adalah menghasilkan model vektor kata Bahasa Indonesia yang dapat digunakan untuk melihat persamaan kata atau melakukan pengelompokan kata yang serupa. Pembuatan *vector space* menggunakan data berita *online* ini diharapkan dapat mempermudah penelitian-penelitian selanjutnya dalam peringkasan berita atau deteksi kesamaan berita. Data yang digunakan dalam penelitian ini adalah artikel dari media daring yang diperoleh dengan cara mengunduh artikel tersebut melalui website media daring yang tersedia.

2. Tinjauan Pustaka

Word embedding adalah sebuah pendekatan yang digunakan untuk merepresentasikan *vector* kata. *Word embedding* merupakan pengembangan komputasi permodelan kata-kata yang sederhana seperti perhitungan menggunakan jumlah dan frekuensi kemunculan kata dalam sebuah dokumen. Hasil dari *word embedding* ini dapat digunakan untuk menggambarkan kedekatan sebuah kata atau sebuah dokumen namun harus dipahami kedekatan tersebut adalah kedekatan kontekstual sesuai dengan data latih yang digunakan dalam pembentukannya, sehingga seringkali kedekatan tersebut bukan merupakan makna sebuah kata.

Word embedding bekerja dengan cara melatih satu set *vector* yang mempunyai panjang tetap secara terus menerus. Secara visual, dalam *word embedding* dapat digambarkan bahwa setiap kata diwakili oleh sebuah titik di dalam luasan bidang tertentu, titik-titik ini kemudian akan dipelajari oleh perhitungan *word embedding* dan satu titik akan dipindahkan menjauh atau mendekati titik yang lainnya, berdasarkan kata-kata lain yang mengelilingi titik tersebut. Hal ini akan dilakukan terus menerus hingga sampai pada sebuah kondisi dimana semua titik tidak dapat dipindahkan lagi mendekati (atau menjauhi) titik lainnya. Sehingga hasil akhir dari iterasi ini dapat memberikan sebuah gambaran dimana kata-kata dengan makna yang serupa akan

cenderung berada dalam satu area yang sama dalam bidang tersebut atau dengan kata lain kata-kata yang ada dalam satu area pada bidang tersebut dan mempunyai jarak kedekatan yang kecil cenderung mempunyai kesamaan semantik. Penelitian ini menggunakan metode *word embedding* yang diberi nama *word to vector* yang dikembangkan oleh Mikolov [7].

WordToVec merupakan sebuah algoritma untuk mempelajari posisi kedekatan semantic antar kata dari sebuah teks masukan, metode word to vec ini terdiri dari dua buah algoritma utama *word embedding* didalamnya, yaitu: *continuous bag of wor*d (CBOW) dan *skip gram*. Algoritma CBOW digunakan untuk melihat panjang tertentu dari sebuah kata pada dokumen masukan. Sedangkan algoritma skip gram digunakan untuk memprediksi konteks kata dengan cara melihat kedekatan sebuah kata dengan kata lain yang posisinya sebelum atau sesudah kata tersebut. Secara arsitektur word to *vector* sebenarnya hanya sebuah jaringan syaraf tiruan yang tidak mempunyai banyak hidden layer, baik secara node dalam tiap layer maupun banyaknya layer.

Sebagai contoh sebuah kalimat "Hari ini cerah" dan "Hari ini oke" yang meskipun menggunakan kosa kata yang berbeda namun pendengarnya merasa ini sebuah Bahasa komunikasi yang bisa dipahami. Meskipun menggunakan kata yang berbeda namun makna kedua kata ini sebenarnya mirip atau sama. Jika kita membangun sebuah kosakata lengkap (notasi V) maka V akan memiliki anggota himpunan sebagai berikut: V={hari, ini, cerah, oke}. Sehingga sebuah *hot-encode vector* dari himpunan V akan mempunyai panjang/ukuran V (=4), penulisan *vector* untuk setiap kata dalam himpunan adalah sebagai berikut: hari=[1,0,0,0]; ini [0,1,0,0]; cerah=[0,0,1,0]; oke=[0,0,0,1]. Jika *vector* kata ini divisualisasikan pada ruang 5 dimensi maka setiap kata akan menempati salah satu dimensi dan tidak berhubungan dengan dimensi lainnya, dengan demikian akibatnya kata "cerah" dan kata "oke" berbeda, padahal secara makna sama. Untuk mengatasi itu kata-kata yang mempunyai makna sama harus menempati posisi spasial yang dekat, atau dalam Bahasa matematis, nilai cosinus sudut antara kedua *vector* kata yang bermakna sama harus mendekati 1, atau nilai sudutnya mendekati 0. Nilai jarak antar *vector* ini dapat dihitung menggunakan rumus:

$$\cos(\emptyset) = \frac{A.B}{||A||||B||} \quad (1)$$

Hal inilah yang menyebabkan munculnya gagasan untuk menghasilkan representasi terdistribusi dimana kata-kata yang mempunyai makna mirip/sama akan mendapat nilai ketergantungan yang lebih besar daripada kata-kata yang bejauhan maknanya. Dengan demikian mengatasi masalah yang muncul pada representasi hot-encoding *vector*

Hal yang harus diperhatikan dalam *word embedding* menggunakan metode *word to vector* ini adalah bagaimana kata dimuat dan disusun dalam sebuah kalimat sebelum dijadikan sebagai masukan pada konstruktor *word to vec*. Agar dapat melakukan hal ini maka data masukan harus tersimpan dalam bentuk kata-kata yang independent, karena itu sebelum masuk kedalam metode *word to vector*, data masukan harus diproses terlebih dahulu dalam sebuah proses yang dinamakan *pre-processing*. Penggunaan metode *word embedding* dan *word to vec* ini dapat dilihat pada penelitian Mikolov [7,8,9], penelitian yang dilakukan oleh Rong [10], penelitian yang Goldberg dan Levy [11], Baroni [12], Grave et all [13], Eren [14] dan Taylor [15].

Meskipun nampaknya perhitungan *word to vec* ini rumit namun pada praktiknya perhitungan teresebut sudah disimpan dalam sebuah fungsi dalam *library*. Penelitian ini menggunakan *library genism* untuk menjalankan *word to vector*. *Library genism* sendiri adalah sebuah *library opensource python* yang digunakan untuk pemrosesan Bahasa alami, *genism* pada awalnya dibangun sebagai library yang digunakan untuk permodelan topik oleh peneliti bernama Ceko Radim Řehůřek. *Genism* merupakan library yang cukup banyak digunakan karena efisiensinya, dalam penelitian ini library genism digunakan agar memudahkan WordToVec saat mempelajari *vector* kata yang baru dari sebuah teks. *Genism* juga digunakan

karena memudahkan untuk membuat hasil training WordToVec disimpan dalam beberapa format.

Tahap *pre-processing* adalah serangkaian proses untuk mempersiapkan teks masukan agar dapat diproses oleh algoritma *word to vec*. Dalam penelitian ini digunakan proses *pre-processing* dangan rangkaian tahapan sebagai berikut: l*ower case, stopword removal dan tokenizing.* Kata-kata hasil tahap *pre-processing* ini kemudian dimuat dalam sebuah memori atau iterator yang mampu memuat teks dalam jumlah yang sangat besar.

Secara matematis *Neural Probabilistic language models* secara tradisional menggunakan prinsip *Maximum Likelihood* pada saat proses pelatihan untuk memaksimalkan probabilitas kata berikutnya (notasi wt) yang diberikan kata-kata sebelumnya (notasi h) dalam fungsi softmax sebagai berikut:

$$P(w_t|h) = \text{softmax}(\text{score}(w_t, h)) = \frac{\exp\{\text{score}(w_t, h)\}}{\sum_{\text{Word w' in Vocab}} \exp\{\text{score}(w', h)\}} \quad (2)$$

Dimana skor (wt, h) menghitung kompatibilitas kata target wt dengan konteks h (biasanya menggunakan perkalian produk titik). Biasanya model ini dilatih dengan cara memaksimalkan *log-likelihood*nya pada data set pelatihan. Perhitungan tersebut dilakukan dengan memaksimalkan persamaan *loss-function* berikut:

$$J_{\text{ML}} = \log P(w_t|h) = \text{score}(w_t, h) - \log\left(\sum_{\text{Word w' in Vocab}} \exp\{\text{score}(w', h)\}\right). \quad (3)$$

Perhitungan ini akan menghasilkan model probabilistik yang dinormalisasi dengan baik untuk pemodelan bahasa.

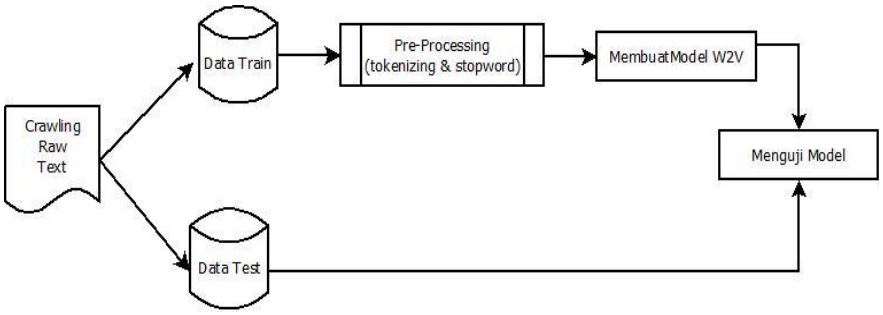
Model *skip-gram* dasar mendefinisikan probabilitas p melalui fungsi softmax. Jika kita menganggap wi sebagai satu *hot-encode vector* dengan dimensi N dan theta menjadi matriks *embedding* matriks N * K di mana kita memiliki kata-kata N dalam kosakata dan *embedding* yang kita pelajari memiliki dimensi K, maka kita dapat mendefinisikannya sebagai berikut:

$$p(w_i|w_t;\theta) = \frac{exp(\theta w_i)}{\sum_t exp(\theta w_t)} \quad (4)$$

Sebagai catatan,setelah melalui proses belajar maka theta matriks dapat dianggap sebagai *embedding lookup matrix*. Dalam Bahasa arsitektur jaringan syaraf tiruan, ini adalah jaringan saraf tiruan tiga lapis yang sederhana.

3. Metodologi Penelitian

Secara umum metodologi pelaksanaan penelitian ini dapat dilihat pada gambar 1 dibawah ini.



Gambar 1. Urutan Langkah Kerja

Data yang digunakan dalam penelitian ini didapatkan dengan cara mengambilnya langsung pada beberapa portal berita online, kategori berita yang diambil adalah berita politik, berita nasional dan ekonomi. Dalam proses ini crawler dibuat masing-masing untuk setiap situs berita, beberapa hal yang hasud diperhatikan dalam pembuatan crawler ini adalah struktur html dari situs yang menjadi target *crawl* dan jenis konten yang akan dimasukkan dalam algoritma perulangan.

Cara yang dilakukan ini sering disebut dengan nama *web scraping*, *web scraping* adalah proses ekstraksi data dari sebuah website. *Web scraping* bekerja menggunakan sebuah kode yang disebut dengan *crawler*, *crawler* ini bekerja dengan cara masuk ke halaman website, mengunduh konten dalam web tersebut, mengekstrak data dari konten kemudian menyimpan data tersebut ke dalam sebuah file atau database.

Setelah data berhasil diekstrak langkah selanjutnya adalah melakukan *pre-processing*, pada tahap *pre-processing* dilakukan beberapa langkah. pertama dilakukan proses *tokenizing* yaitu proses pemotongan teks input kedalam bentuk kata-kata yang menyusunnya, pemecahan ini dilakukan dengan cara menscan beberapa *delimiter whitespace* (pemisah), *delimiter* yang digunakan dalam penelitian ini adalah spasi, tab dan *newline*. Setelah proses *tokenizing* dilakukan, selanjutnya dilakukan proses penghilangan *stopword (stopword removal)*. Pada proses ini kata-kata yang tidak bermakna dan tanda baca namun frekuensinya sering muncul dihilangkan guna mendapatkan kata-kata kunci yang dapat digunakan pada proses selanjutnya. Setelah proses *stopword removal* dilakukan, hasilnya kemudian dijadikan sebagai masukan pada proses *stemming*. *Stemming* adalah sebuah proses dimana kata berimbuhan diubah menjadi kata dasarnya, pada penelitian ini digunakan modul *stemming* sastrawi yang banyak digunakan dalam penelitian teks Bahasa Indonesia.

Setelah tahapan *pre-processing* dilakukan, langkah selanjutnya adalah pembuatan model. Model dibuat dengan menggunakan metode *Word2Vec* menggunakan *library genism*, keluaran dari tahapan ini adalah sebuah model representasi kata dari data latih dalam *vector* kata, model ini nantinya dapat diakses melalui atribut "wv". Model ini kemudian dapat disimpan dalam sebuah file menggunakan fungsi save_*Word2Vec*_format() yang sudah disediakan *library genism*.

Model *vector* kata diperoleh dengan melatih algoritma melalui sekumpulan teks, sehingga pada setiap akhir pelatihan, setiap kata dalam data pelatihan yang digunakan menjadi *vector* numerik dengan panjang (p) yang ditentukan. Panjang (p) yang ditentukan ini sangat jauh lebih kecil dari panjang total kata-kata dalam data pelatihan (n) sehingga dapat ditulis p<<n. Vektor yang dihasilkan ini mewakili peta linier/proyeksi dari vektor (n) kedalam *vector* (p) dalam sebuah ruang kata imajiner p-dimensi. Penggunaan kata imajiner ini karena p-dimensi sebenarnya hanyalah konstruksi matematis sehingga tidak memiliki makna fisik yang dapat ditampilkan. Sehingga pada umumnya orang akan berfikir dengan meningkatkan nilai p akan didapatkan representasi kata dengan kualitas yang lebih baik.

Selanjutnya untuk melihat kemampuan model maka model yang dihasilkan kemudian diuji. Pengujian model ini sebenarnya mirip dengan pengujian klasifikasi. Klasifikasi bergantung pada gagasan kesamaan. Kesamaan ini dapat diperolah dari kesamaan yang sederhana dari sebuah nilai fitur kategorikal seperti warna atau bentuk dari sebuah objek, atau

berdasarkan fungsi yang lebih kompleks dari semua nilai fitur kategorikal yang dimiliki objek klasifikasi. Demikian juga dengan sebuah dokumen, dokumen dapat diklasifikasikan menggunakan atribut-atribut (fitur) yang dapat mudah diukur seperti ukuran file atau ekstensi file, hal ini dapat dengan mudah dilakukan namun biasanya manusia mengklasifikasikan sebuah dokumen berdasarkan kesamaan isi (makna) dokumen tersebut.

Skenario pengujian yang digunakan dalam penelitian ini adalah dimulai dari menampilkan kata-kata yang mendekati dengan kata masukan dan menampilkan kedekatannya dalam angka. Selanjutnya, menampilkan tiga kata yang mendekati kata masukan dan menampilkan kedekatannya dalam angka. Lalu dilanjutkan dengan menampilkan kedekatan antara dua kata yang dimasukkan. Tahap terakhir ialah mendeteksi kata yang paling jauh kedekatannya dari "n" masukan kata.

4. Hasil dan Diskusi

Total data yang digunakan dalam penelitian ini adalah 1200 berita dengan rincian: 400 data dari kategori berita politik, 400 kata dari kategori berita nasional dan 400 data dari kategori berita ekonomi. Data yang di*crawling* mempunyai beberapa metadata yang dijadikan sebagai *header* data yaitu: sumber/link dimana data tersebut diakses, tanggal artikel tersebut di*release* dan konten atau isi berita. Dalam penelitian ini data disimpan dalam format CSV dan hanya konten yang digunakan sebagai data latih untuk membuat *vector* kata.

Pada langkah pertama *pre-processing* data masukan berupa artikel dipotong kata per kata menggunakan acuan tanda baca atau spasi, langkah ini disebut dengan istilah *tokenizing.* Berikut adalah kode yang digunakan untuk memotong artikel menjadi sebuah kalimat:

```
def read_input(input_file):
        logging.info("membaca data {0}... ... ... ".format(input_file))
        with gzip.open(input_file, 'rb') as f:
            for i, line in enumerate(f):
                if (i % 10000 == 0):
                    logging.info("membaca {0} data".format(i))
                yield gensim.utils.simple_preprocess(line)
```

Fungsi diatas dibuat menggunakan *library genism*, proses *tokenizing* dilakukan oleh *library genism*, perintah yang digunakan adalah *gensim.utils.simple_preprocess(line). Library genism* sekaligus melakukan beberapa pra-pemrosesan dasar seperti *tokenization*, mengubah semua kata menjadi huruf kecil, dan menampilkan hasilnya sebagai daftar token (kata-kata). Berikut adalah penjelasan perintah *genism.utils.simple_preprocess* dari *library genism*. Secara umum perintah ditulis dengan perintah:

```
gensim.utils.simple_preprocess(doc, deacc=False, min_len=2, max_len=12)
```

Perintah ini berfungsi untuk mengubah dokumen kedalam daftar token yang semuanya ditulis menggunakan huruf kecil dan mengabaikan token yang terlalu pendek atau terlalu panjang. Dalam penelitian ini batas token terpendek yang digunakan adalah 2 huruf dan batas token yang terpanjang adalah 12 huruf. Berikut adalah Parameter yang digunakan:

- doc (str) – digunakan sebagai parameter untuk Input dokumen.
- deacc (bool) – digunakan untuk menghapus tanda aksen dari token menggunakan deaccent ()?
- min_len (int) – digunakan untuk menentukan panjang token minimum (inklusif). Token yang lebih pendek dibuang.
- max_len (int) – digunakan untuk menentukan panjang token maksimum dalam hasil (inklusif). Token yang lebih panjang dibuang.
- return: Merupakan hasil keluaran fungsi ini, berupa token (kata) diekstraksi dari doc.

- Jenis return: merupakan jenis keluaran dari fungsi ini, keluaran ditampilkan dalam daftar str

Langkah awal pembuatan model adalah dengan men*training*/melatih model. Proses pelatihan model sebenarnya cukup mudah dilakukan menggunakan *library Word2Vec genism*, kita hanya perlu menginisiasi penggunaan fungsi *Word2Vec* dan memasukkan daftar token yang telah dibuat pada langkah sebelumnya kedalam fungsi ini. *Word2Vec* akan menggunakan semua token yang dimasukkan untuk membuat kosa kata dalam prosesnya, dalam pembuatan kosakata ini kata-kata yang sama akan dihitung satu. Berikut adalah kode dari proses training.

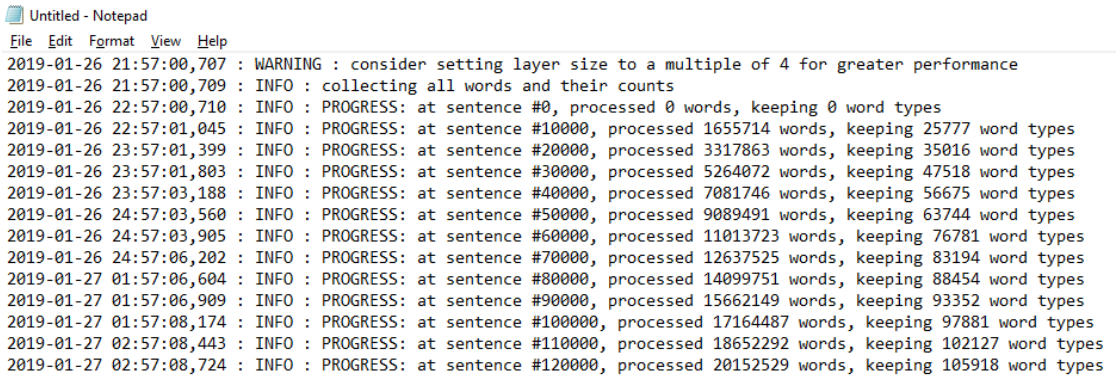
```
model = genism.models.Word2Vec(document, size=1200, window=5, min_count=2, workers=8)
model.train(documents, total_examples=len(documents), epochs=10)
```

Parameter yang digunakan:

- size (int) - Dimensi dari vektor kata yang digunakan, dalam penelitian ini adalah 1200.
- window (int) - Jarak maksimum antara kata saat ini dan prediksi dalam sebuah kalimat, semakin kecil nilainya semakin baik hasilnya, dalam penelitian ini digunakan sebesar 5.
- min_count (int, opsional) – Digunakan sebagai batasan untuk mengabaikan semua kata dengan frekuensi total lebih rendah dari ini, dalam penelitian ini digunakan batas terkecil adalah 2.
- workers (int) – digunakan untuk menentukan berapa banyak core yang digunakan untuk melatih model, semakin banyak core digunakan semakin cepat pemrosesan, dalam penelitian ini karana laptop menggunakan cuda core sebanyak 8 core maka workers yang digunakan sebanyak 8.

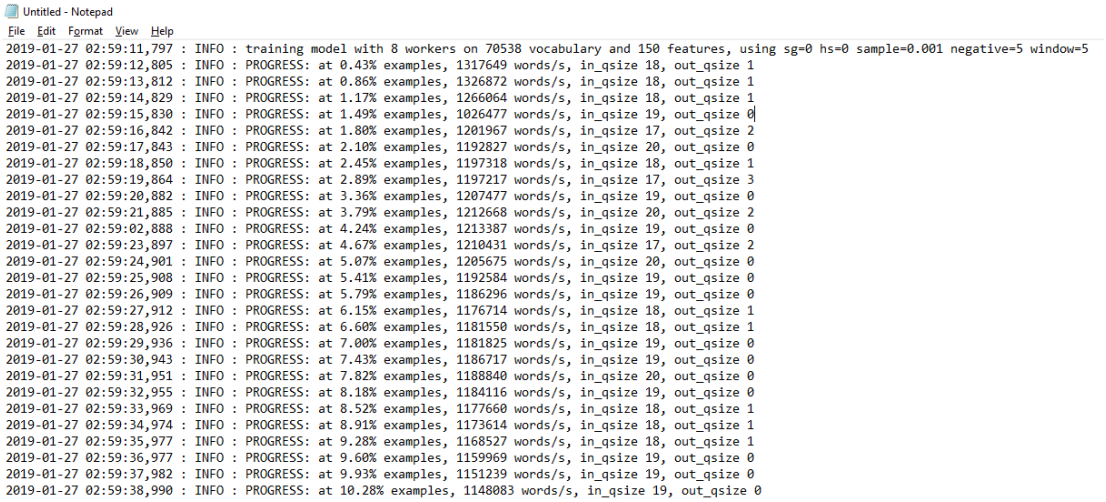
Secara teoritis penggunaan nilai yang lebih kecil dalam parameter window akan menghasilkan hasil yang lebih baik, dalam eksperimen digunakan *window* bernilai 2-3-4-5-6-7, berlawanan dengan teorinya eksperimen menunjukkan *window* dengan nilai 5 lebih baik hasilnya dibandingkan dengan *window* yang bernilai lebih kecil. Meskipun demikian peneliti tidak mempunyai ujicoba yang cukup untuk menyimpulkan penyebab hal ini. Sedangkan untuk parameter *min_count* peneliti menggunakan frekuensi kemunculan kata lebih dari dua kali dalam penelitian ini dan tidak mencoba nilai frekuensi yang lain.

Log dari pemrosesan data untuk langkah *load* data dapat dilihat pada gambar 2 dibawah ini:



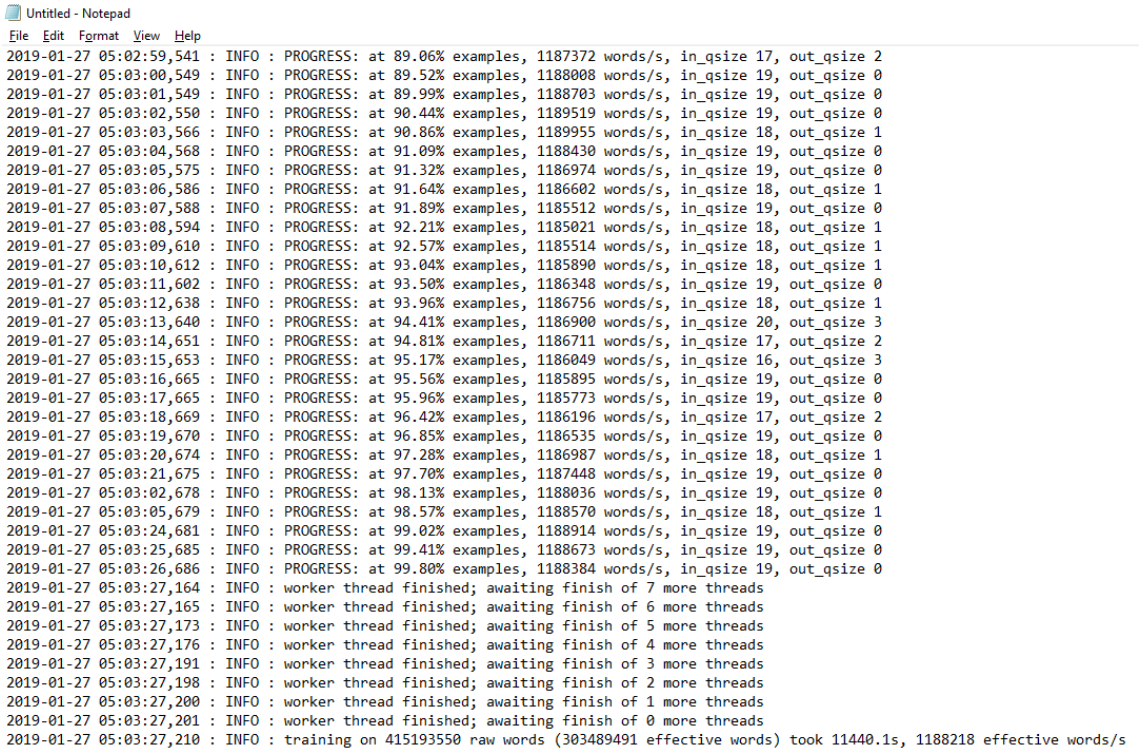
Gambar 2. Log data

Setelah data diproses ke memory, langkah selanjutnya adalah proses training. Berikut adalah log saat mulai dan saat selesai training, pemotongan log dilakukan karena proses training yang memakan waktu terlalu lama. Log data saat proses training dimulai dapat dilihat pada gambar 3.



Gambar 3. Log data latih (mulai)

Gambar 4 dibawah menunjukkan log saat proses training selesai, dapat dilihat bahwa pemrosesan training ini menggunakan 8 *thread* pada GPU CUDA laptop dengan waktu pemrosesan kurang lebih sebesar 11440,1 detik dan menghasilkan daftar kata sebanyak 303489491 kata.



Gambar 4. Log data latih (selesai)

Pengujian dilakukan dengan cara memasukkan satu kata kedalam model dan melihat kata – kata yang sama. Pengujian dilakukan langsung di *python*, pengujian pertama dilakukan menggunakan kata "target" berikut adalah kode untuk pengujian pertama:

w1 = "target"

model.wv.most_similar (positive=w1)

Kode tersebut menghasilkan keluaran sebagai berikut:

2019-01-29 20:03:42 INFO : precomputing L2-norms of word weight *vector*s

[('sasaran', 0.871721625328064), ('batas', 0.7922376990318298), ('ketentuan', 0.7915753126144409), ('pendapatan', 0.7772612571716309), ('melampaui', 0.7618112564086914), ('mendorong', 0.7483716011047363), ('tumbuh', 0.7330487966537476), ('investasi', 0.7239381074905396), ('kinerja', 0.7228356599807739), ('penerimaan', 0.7213647365570068)]

Dari hasil pengujian didapatkan nilai kedekatan sebesar 0,8 hingga 0,7, sehingga dapat disimpulkan model cukup dapat mengenali kata yang sama atau mempunyai *vector* yang cukup dekat dengan masukan.

Untuk menampilkan 3 kata yang mendekati dengan kata "target" digunakan perintah sebagai berikut:

```
w1 = "target"
model.wv.most_similar (positive=w1, topn=3)
```

Kode tersebut menghasilkan keluaran sebagai berikut:

2019-01-29 20:03:50 INFO : precomputing L2-norms of word weight *vector*s

[('sasaran', 0.871721625328064), ('batas', 0.7922376990318298), ('ketentuan', 0.7915753126144409)]

Hasil pengujian menunjukkan 3 tingkat kedekatan kata dengan nilai antara 0,80 hingga 0,79.

Selanjutnya diadakan pengujian untuk mencari kata yang sama menggunakan masukan lebih dari satu kata, berikut adalah kode yang digunakan untuk melakukan hal tersebut:

```
w1 = ['modal', 'uang', 'pendapatan']

w2 = ['kesempatan']

model.wv.most_similar (positive=w1, negative=w2, topn=8)
```

Koding diatas akan menampilkan delapan hasil positif (artinya dekat) dengan kata-kata yang dimasukkan pada w1 dan sekaligus memberikan keluaran kata-kata yang negatif (artinya jauh) dengan masukan w2. Berikut adalah keluaran dari koding diatas:

[('apbn', 0.7086508274078369), ('laba', 0.7016597390174866), ('membaik', 0.7002605199813843), ('positif', 0.6868821978569031), ('tumbuh', 0.6777950525283813), ('merata', 0.6413239240646362), ('adil', 0.6777950525283813), ('sejahtera', 0.6413239240646362)]]

Model juga dapat menguji kesamaan dua buah kata, sejauh kata tersebut ada dalam data latih yang digunakan, berikut adalah kode untuk menguji kedekatan dua buah kata berdasarkan data latih yang digunakan:

```
model.wv.most_similar (w1='uang', w2='kaos')
```

Berikut adalah hasil keluaran koding diatas:

0.35355593501920781

Semakin kecil angka yang dihasilkan menunjukkan kedua kata mempunyai vektor kata yang jauh. Untuk kata yang sama maka nilai *vector* akan sebesar 1.0 karena rentang skor *cosinus* selalu berada antara [0.0 – 1.0].

Model yang dihasilkan juga dapat digunakan untuk mendeteksi kata dari sekelompok kata yang paling tidak terkait dengan kata lainnya, berikut adalah kode yang dapat digunakan untuk melakukan hal tersebut:

model.wv.doesnt_match([“pendapatan”, “pemasukan”, “kucing”]

Kode diatas menghasilkan “kucing” yang paling tidak sama dengan kedua kata lainnya, ketika diuji menggunakan kata: “pendapatan”, “pemasukan” dan “neraca” koding diatas menghasilkan “neraca” sebagai kata yang mempunyai nilai *vector* kecil atau jarak lebih jauh dibanding jarak relatif ketiga vektor kata tersebut.

5. Kesimpulan dan Saran

Dari eksperimen yang dilakukan dalam penelitian ini didapatkan lima poin kesimpulan. Pertama, makna dan keterbatasan dari *Word embedding* ini harus dipahami dengan jelas terlebih dahulu sehingga dalam penerapannya tidak menghasilkan klaim yang berlebihan. Kedua, algoritma *Word2Vec* mempelajari *vector* kata dengan cara yang tidak diawasi manusia (unsupervised) karena tidak dilakukan pelabelan terhadap data latih terlebih dahulu. Ketiga, vektor numerik yang diperoleh untuk sebuah kata dari dokumen pelatihan adalah hasil dari sebuah fungsi algoritma yang diterapkan algoritma tersebut pada data pelatihan untuk menurunkan *vector* data pelatihan tersebut. Keempat, tidak relevan mencampur *vector* hasil model dan *vector* data latih meskipun kedua *vector* tersebut berasal dari dokumen (atau korpus) yang sama. Kedua *vector* ini berbeda karena tidak mempunyai hubungan dan cara mereka mempelajarinya juga berbeda. Kelima, sebanyak apapun data latih yang digunakan, vektor kata yang dihasilkan akan kontekstual, artinya *vector* yang menggunakan data latih review film tidak tepat diterapkan misalnya pada data politik. Atau menganalisis sebuah dokumen menggunakan dua model yang dibangun dengan data latih yang berbeda konteks.

Saran bagi penelitian selanjutnya ialah pertama, model *Word2Vec* yang dibangun meskipun menunjukkan performa yang baik dalam mendeteksi kesamaan kata namun perlu dibangun ulang menggunakan data latih/korpora yang lebih besar jumlahnya dan variasi konteksnya sehingga menghasilkan model yang universal. Kedua, proses komputasi membangun model *Word2Vec* ini masih memakan waktu yang cukup lama meskipun diproses menggunakan computer dengan GPU, maka algoritma yang digunakan perlu dioptimalisasikan sehingga diharapkan dapat mempercepat waktu pemrosesan. Ketiga, model penelitian ini dibangun menggunakan data tidak terstruktur namun sebaiknya untuk keperluan tertentu *Word2Vec* dibangun menggunakan data terstruktur misal: data pertanyaan dan jawaban pada sebuah forum diskusi.

Referensi

[1] Tomas Mikolov, Greg Corrado, Kai Chen & Jeffrey Dean. *“Efficient Estimation of Word Representations in Vector Space”, 2013*. https://arxiv.org/pdf/1301.3781.pdf

[2] Yoshua Bengio, Ducharme Rejean, Vincent Pascal & Janvin Christian. *“A Neural Probabilistic Language Model”*, 2003. http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf

[3] Collobert Ronan, & Weston Jason. *“A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning”*. 2008. https://ronan.collobert.com/pub/matos/2008_nlp_icml.pdf

[4] M. Faruqui and C. Dyer, *"Improving Vector Space Word Representations Using Multilingual Correlation"*, Carnegie Mellon University, 2014 hal.236-244.

[5] R. Kiros, Y. Zhu, R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torallba and S. Fidler, *"Skip-Tought Vector"*, Advances in Neural Information Processing System, 2015. [diakses tanggal 07 Januari 2018]

[6] L. Wolf, Y. Hanani, K. Bar, and N.Dershowitz, *"Joint Word2Vec Networks for Bilingual Semantic Representations"*, International Journal of Computation Linguistics and Applications, Vol.5, 2014. [diakses tanggal 07 Januari 2018]

[7] T. Mikolov, K. Chen, G.Corrado, and J.Dean, *"Efficient Estimation of Word Representations in Vector Space"*, Cornell 2013. [Online] https://arxiv.org/pdf/1301.3781.pdf [diakses tanggal 07 Januari 2018]

[8] T. Mikolov, I. Sutskever, K. Chen, G.Corrado and J.Dean, *"Distributed Representations of Words and Phrases and their Compositionality"*, Neural Information Processing System Conference, 2013. [Online] https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf [diakses tanggal 07 Januari 2018]

[9] T.Mikolov, Y.Wen-Tau, and G.Zweig, *"Linguistic Regularities in Continuous Space Word Representations"*, Association for Computational Linguistic 2013. [Online] http://www.aclweb.org/anthology/N13-1090 [diakses tanggal 07 Januari 2018]

[10] X.Rong, *"Word2Vec Parameter Learning Explained"*, Cornell 2016. [Online] https://arxiv.org/pdf/1411.2738v3.pdf [diakses tanggal 07 Januari 2018]

[11] Y.Goldberg and O.Levy, *"Word2Vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method"*, Cornell 2014. [Online] https://arxiv.org/pdf/1402.3722v1.pdf [diakses tanggal 07 Januari 2018]

[12] M. Baroni and A. Lenci, "*Distributional Memory: A General Framework for Corpus-based Semantics"*, Computational Linguist, 2010. [Online] https://arxiv.org/pdf/1301.3781.pdf [diakses tanggal 07 Januari 2018]

[13] E. Grave, P. Bojanowski, P. Gupta, A. Joulin and T. Mikolov, *"Learning Word Vectors for 157 Languages"*, International Conference on Language Resources and Evaluation, [online] https://arxiv.org/abs/1802.06893, [diakses juli 2018]

[14] L.T. Eren and K.Metin, *"Vector Space Models in Detection of Semantically Non-compositional Word Combinations in Turkish"*, Analysis of Images, Social Networks and Texts (AIST) 2018. Lecture Notes in Computer Science, vol 11179. Springer

[15] S.Taylor and T.Brychcín, *"The representation of some phrases in Arabic word semantic vector spaces"*, Open Computer Science. 8(1): 182-193

(Halaman ini sengaja dikosongkan.)