Pertama-tama saya ingin menyampaikan bahwa saya bukan pakar di bidang ini—bidang penelitian saya sebenarnya lebih ke CS theory & mathematical logic—jadi mungkin rekan2 di sini lebih dalam ilmunya dibanding saya. Saya lihat ada Mas Kemal di grup ini, saya tahu beliau sangat mendalami NLP. Oleh karena itu, jika saya ada yg kurang atau kurang tepat, silakan ditambahkan :)

Oke, langsung saja. Untuk kulgram saya hari ini, saya akan membahas
1. Natural Language Understanding & Generation
2. Language Modelling
3. Word embeddings/distributional semantic model.

Materinya akan cenderung teori, dan bukan aplikasi/praktis.

Apa itu Natural Language Understanding (NLU)? Secara intuitif, NLU berfokus pada pengembangan teknik untuk membuat komputer "memahami" bahasa alami (bahasa manusia).

NLU digolongkan sebagai hard problem dalam AI. Seperti yg kita ketahui, komputer sangat baik ketika bekerja dengan data yang terstruktur. Komputer dapat memproses data tersebut dengan kecepatan yang jauh lebih tinggi dibanding manusia. Namun, manusia tidak berkomunikasi dengan data yang "terstruktur" (kita juga tak bicara dengan menggunakan kode biner!)

Kita berbicara menggunakan bahasa alami, yang secara umum kurang terstruktur. Sayangnya, komputer kesulitan dalam memproses bahasa alami yang cenderung tak terstruktur dan aturannya abstrak dan ambigu. Ketika kita membaca suatu teks di suatu websitus, kita paham apa maksud dari teks tersebut di dunia nyata. Ketika kita membaca twit di Twitter, emosi kita dapat dibangkitkan, kita bahkan dapat memvisualisasikan/membayangkan apa yang ditwitkan seseorang jika itu terjadi kepada kita di dunia nyata.

NLP adalah salah satu sub-field dari AI yang bertujuan membuat komputer mampu memproses (bahkan, mungkin memahami) bahasa alami, sebaik manusia.

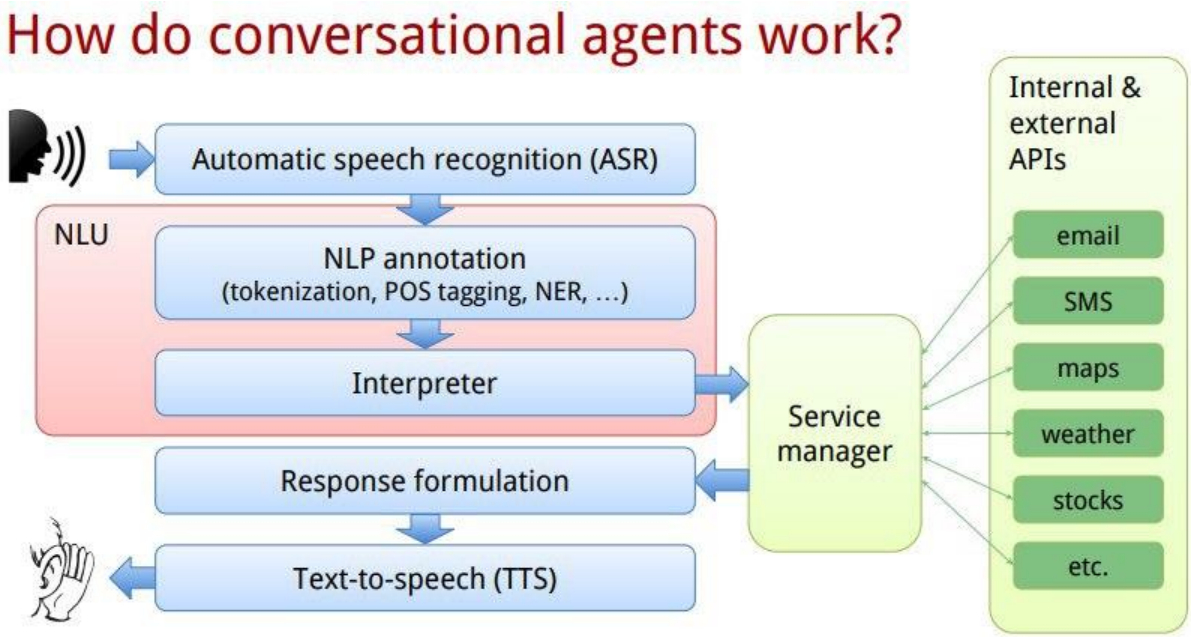
Sekarang, apa yang dimaksud *memahami*?

Memahami suatu kalimat adalah:
- menentukan nilai kebenarannya (dengan justifikasi)
- memperhitungkan konsekuensi
- mengambil tindakan yang sesuai
- menerjemahkan dari satu bahasa ke bahasa lain
- dst...

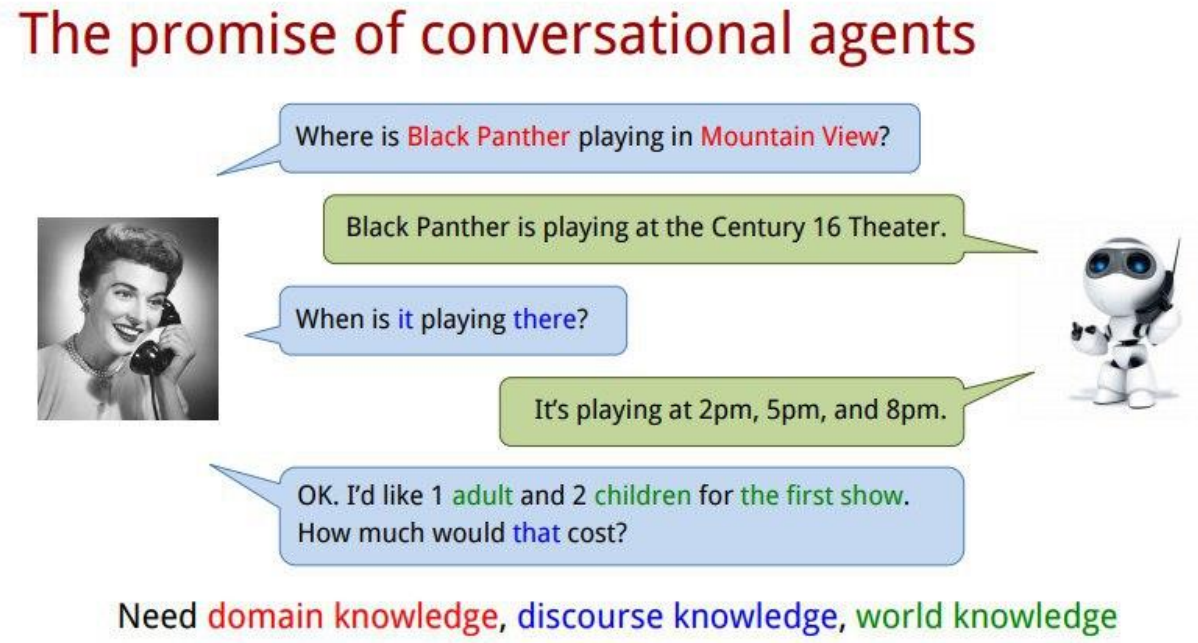
Ada banyak debat filosofis tentang apa itu yang diimaksud dengan *memahami*. Yang paling terkenal adalah Turing test (the imitation game). Yang juga penting namun tak terlalu terkenal: The Chinese room argument dari John Searle. Terlepas dari perdebatan filosofis, sejarah NLU bisa dirunut hingga tahun 60an. Sejarah singkat NLU sbb:
- 1960an: pattern-matching dengan rule-sets (kecil) [STUDENT, ELIZA]
- 1970-80an: menggunakan teori linguistik, berbasiskan formal logic, aplikasi terbatas [SHRDLU]
- Pertengahan 1990an: revolusi statistik NLP
- Akhir 200an: NLU menggabungkan teknik sebelumnya
- Pertengahan 2010an: DL menguasai NLU

Untuk saat ini NLU mengalami perkembangan yang pesat, seiring dengan booming-nya AI/ML. Tahun 2017 Microsoft dan IBM mencapai human parity dalam mengenali percakapan manusia (dataset: Switchboard). Tahun 2018 Microsoft mampu menerjemahkan Chinese - English (dataset:

newstest2017) dengan kualitas setara manusia. Selain itu ada banyak sekali perkembangan di industri: Siri, Google Assistant, Amazon Alexa, Microsoft Cortana, dst.




Secara garis besar, arsitektur conversational agents (Siri, Cortana, Alexa) sbb




Dibutuhkan banyak domain knowledge untuk membangun converstional agents. Ini tentu bukan permasalahan yang mudah. Dan memang, NLU belum sepenuhnya terpecahkan.

# The reality of conversational agents

**Colbert**: Write the show.

**Siri**: What would you like to search for?

**Colbert**: I don't want to search for anything! I want to write the show!

**Siri**: Searching the Web for "search for anything. I want to write the shuffle."

[...]

**Colbert**: For the love of God, the cameras are on, give me something?

**Siri**: What kind of place are you looking for? Camera stores or churches?

**Colbert**: F—- you.

**Siri**: I'd blush if I could.

Seperti contoh anekdot ini :blush:

Oke. Sekarang apa yg dimaksud dengan Natural Language Generation (NLG). Secara intuitif, NLG adalah "kebalikan" dari NLU. Kalau NLU adalah dari TEKS -> PENGETAHUAN/PEMAHAMAN, NLG adalah PENGETAHUAN/PEMAHAMAN -> TEKS. Misal mesin anotasi: ada foto atau video, kemudian mesin tersebut akan mendeskripsikan apa yang terlihat di foto/video tsb dengan menggunakan teks.

Secara umum, NLG cenderung lebih mudah dibanding NLU, satu faktor utamanya adalah, input NLG dapat dikontrol. Sedangkan NLU tidak.

| NLG | NLU |
|---|---|
| Relatively Unambiguus | Ambiguity in input |
| Well-formed | ill-formed input |
| Well-specified | Under-specification |

Lalu langkah awal untuk menyelesaikan masalah di NLU adalah bagaimana kita merepresentasikan suatu teks sehingga dapat diproses/dipahami oleh komputer. Kita menggunakan Language Modelling (LM). Awalnya (dan sampai saat ini) language modelling (LM) dikembangkan untuk memecahkan permasalahan pengenalan ucapan (speech recognition). Dalam NLP, teknik LM juga digunakan untuk memecahkan masalah tagging dan parsing. Secara umum permasalahan yang ingin kita pecahkan adalah sebagai berikut. Asumsikan kita memiliki suatu korpus (suatu himpunan kalimat dalam suatu bahasa), misal kita punya teks majalah Tempo Online dari edisi beberapa tahun terakhir. Dengan korpus ini, kita ingin memperkirakan parameter dari suatu LM. LM didefinisikan sebagai berikut. Pertama, kita definisikan V sebagai himpunan seluruh kosa kata dalam suatu bahasa. Misal dalam bahasa Indonesia kita punya V = {komputer, lari, kucing, makan, . . . }.
Kardinalitas V berhingga, i.e., untuk bahasa Indonesia ±100000 (KBBI). Sebuah kalimat dalam suatu bahasa adalah suatu rangkaian kata-kata

Sebuah kalimat dalam suatu bahasa adalah suatu rangkaian kata-kata
x1 x2 . . . xn
di mana $n \geq 1$, $i \in \{1, \ldots, (n-1)\}$, $x_i \in V$ , dan xn adalah simbol STOP.

Contoh:
- kucing itu lari STOP
- kucing tidur STOP

Definisikan $V^*$ sebagai himpunan semua kalimat dari kosa kata $V$; tentu kardinalitas $V^*$ tak berhingga, karena kalimatnya tak dibatasi panjangnya.

**Definition 1 (Language Model)** *Language model terdiri dari sebuah himpunan terhingga $V$ dan sebuah fungsi $p(x_1, x_2, \ldots, x_n)$ di mana:*

1. *untuk sembarang $(x_1 \ldots x_n) \in V^*, p(x_1, x_2, \ldots, x_n) \geq 0$*
2. *$\sum_{(x_1 \ldots x_n) \in V^*} p(x_1, x_2, \ldots, x_n) = 1$;*

*Jadi $p(x_1, x_2, \ldots, x_n)$ adalah sebuah distribusi probabilitas dari kalimat di $V^*$.*

Language model didefinisikan sbb

Satu contoh pemelajaran LM adalah sbb. Definisikan c(x1,..,xn) sebagai jumlah kemunculan kalimat x1,..,xn di dalam korpus, dan N sebagai jumlah kalimat di dalamnya.

$$p(x_1, x_2, \ldots, x_n) = \frac{c(x_1 \ldots x_n)}{N}.$$

Kita bisa definisikan LM sbb

Namun, metode pemelajaran LM ini kurang baik, karena metode ini akan menetapkan probabilitas 0 untuk kalimat yang tidak muncul dalam korpus. Sehingga gagal generalisasi ke kalimat yang tak ada di dalam training data. Mengapa LM penting? Dalam berbagai aplikasi, sangat berguna untuk memiliki "prior" distribution p(x 1 , x 2 , . . . , x n ) dari kalimat-kalimat yang kemungkinannya besar maupun kecil untuk muncul dalam suatu bahasa. Misal dalam speech recognition, LM dipadukan dengan acoustic model yang memodelkan pengucapan berbagai kata. Jadi ketika ada ucapan (audio) dalam suatu bahasa, acoustic model akan menghasilkan kandidat kalimat beserta probabilitas nya, kemudian LM digunakan untuk mengurutkan kemungkinannya berdasarkan probabilitas tiap kalimat muncul dalam bahasa yang digunakan.

Selanjutnya kita akan melihat sekilas metode Markov Models yang menjadi dasar untuk berbagai LM (n-gram).

$$P(X_1 = x_1, \ldots, X_n = x_n).$$

Misal ada rangkaian variable random X 1 , X 2 , . . . , X n ,
masing-masing bisa bernilai sembarang elemen di V . Asumsikan n tetap, misal
n = 42. Kita ingin memodelkan probabilitas sembarang x 1 . . . x n , n ≥ 1, x i ∈ V .
Kita memodelkan joint probability sb

Total ada |V|^n kemungkinan rangkaian. Jadi melis semua kemungkinan tentu kurang efisien. Kita butuh model yang lebih efisien.

$$P(X_1 = x_1, X_2 = x_2, \ldots X_n = x_n) = \prod_{i=1}^{n} P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1})$$

Selanjutnya, kita membahas Trigram LM. Trigram LM berangkat dari second-order Markov process, i.e., probabilitas X_i, bergantung pada 2 elemen sebelumnya, X_{i-1} dan X_{i-2}. Secara formal sbb

**Definition 2** *Trigram LM terdiri dari:*

1. *Set* $V$,
2. *Parameter* $q(w|u,v)$ *untuk tiap trigram* $u, v, w$ *dimana* $w \in V \cup \{\text{STOP}\}$, *and* $u, v \in V \cup \{^*\}$

*nilai dari* $q(w|u,v)$ *diinterpretasikan sebagai probabilitas menemui* $w$ *tepat setelah* $(u,v)$. *Jadi secara umum kita punya*

$$p(x_1 \dots x_n) = \prod_{i=1}^{n} q(x_i|x_{i-2}, x_{i-1})$$

Dengan ini kita bisa definisikan Trigram LM

Trigram adalah hanya salah satu LM, kita tentu saja bisa extend berapapun dependensinya, oleh sebab itu: n-gram. Lantas, bagaimana kita evaluasi suatu LM? Ada 2 strategi utama.

Extrinsic evaluation: Ini adalah cara evaluasi jika kita tahu bagaimana untuk melakukannya. Misal kita tahu bagaimana cara mengukur kualitas spell-checker, dan spell-checker ini menggunakan suatu model. Misal kita punya 2 model: LM1 dan LM2. Kita evaluasi spell-checker ketika menggunakan LM1, dan kemudian LM2, dan membandingkan hasilnya. Evaluasi ini bagus, tapi cukup mahal.

yg kedua

Intrinsic evaluation: evaluasi tanpa dependensi eksternal, i.e., hanya mengukur parameter internal dari model. Biasanya dicapai menggunakan training/testing dataset. Kita ``train" model dengan training data, kemudian kita gunakan testing data untuk mengukur akurasi model. Semakin akurat semakin baik.

Perlu diingat bahwa model (probabilistik) tak memprediksi data spesifik. Namun, ia memberikan nilai probabilitas kepada data yang mungkin digunakan sebagai input. Dalam kasus LM, model memprediksi probabilitas kata selanjutnya berdasarkan informasi sebelumnya. Jadi, ia tidak memprediksi satu ``kata terbaik", melainkan probabilitas semua kata yang ada dalam V.

Ukuran untuk menilai kualitas model disebut perplexity. Metode pengukuran sbb:

1. kita punya kalimat dari test data $x^1, x^2, \dots, x^m$, masing-masing $x^j$ adalah suatu kalimat $x_1^j \dots x_n^j$. Test data ini bukan bagian dari training data, i.e., belum pernah digunakan sebelumnya.
2. kita bisa ukur probabilitas $p(x^j)$, dan menggunakan probabilitas keseluruhan dari kalimat test

$$\prod_{i=1}^{m} p(x^j)$$

semakin tinggi hasilnya, semakin baik model kita ketika bertemu dengan kalimat baru.

$$M = \sum_{j=1}^{m} n_j$$

Definisikan M sebagai total kata dalam test corpus sbb

$$\frac{1}{M} \log_2 \prod_{j=1}^{m} p\left(x^j\right) = \frac{1}{M} \sum_{j=1}^{m} \log_2 p\left(x^j\right)$$

Maka log probabilitas rata-rata adalah

Intuisi: semakin tinggi nilainya, semakin bagus modelnya.

Perplexity didefinisikan sebagai $2^{-l}$ di mana

$$l = \frac{1}{M} \sum_{j=1}^{m} \log_2 p\left(x^j\right)$$

Perplexity didefinisikan sebagai

Di sini kita bisa lihat, mengikuti intuisi sebelumnya, bahwa semakin kecil nilai perplexity, maka semakin baik modelnya dalam memodelkan data baru. Oke, good, kita punya LM. Tapi masalah tak selesai begitu saja.. Salah satu masalah utama adalah: curse of dimensionality. semakin besar training data, semakin besar kosa kata. Sebelumnya kita ketahui bahwa suatu himpunan kosa kata |V|^n memiliki kemungkinan rangkaian kata. Data menjadi sparse karena banyaknya kemungkinan rangkaian kata tumbuh secara eksponensial. Bagaimana mengatasi permasalahan ini? Salah satu jawabannya adalah menggunakan teknik word embedding yang dapat mengurangi dimensi. Secara garis besar, word embedding adalah cara untuk merepresentasikan LM sebagai suatu vektor dari bilangan riil. Bagaimana cara untuk membangunnya? Salah satu teknik yang paling populer adalah word2vec yang dikembangkan oleh Tomas Mikolov et. al. tahun 2013.

Misal kita punya dua kalimat yang "mirip":
1. Seekor kucing makan
2. Seekor harimau makan

Maka kita punya V = {seekor, kucing, harimau, makan}. Kita bisa membuat one-hot encoding vector untuk tiap kata di V . seekor = [1, 0, 0, 0], kucing = [0, 1, 0, 0], harimau = [0, 0, 1, 0], makan = [0, 0, 0, 1]. Jika kita visualisasikan encoding ini, maka kita bisa menggunakan 4-dimensional space, masing-masing kata di V menempati satu dimensi (tanpa proyeksi ke dimensi lainnya). jadi, kata "kucing" dan "harimau" memiliki tingkat perbedaan yang sama seperti "seekor" dan "makan". Ini tentu sedikit aneh, karena kucing dan harimau lebih "mirip" (sama-sama binatang mamalia lucu yang berada di family felidae) dibanding "seekor" dan "makan".

Kita ingin membangun vektor yang dapat merepresentasikan kedekatan antar kata, i.e., cosine similarity-nya mendekati 1. Di sini muncul konsep distributed representations. Secara intuitif, kita membuat suatu dependensi antar kata-kata. Sebelumnya, dengan one-hot encoding, masing-masing kata independen terhadap yang lain. word2vec adalah salah satu metode untuk membangun dependensi antar kata yang kita inginkan. Ada dua cara (semua menggunakan Neural Nets): skip-gram dan common bag of words (CBOW).

CBOW

Metode ini menggunakan konteks dari tiap kata sebagai input dan berusaha untuk memprediksi kata yang berhubungan dengan konteks tersebut. Misal contoh kita sebelumnya: seekor kucing makan. Misal input ke NN adalah kata "kucing". Kita ingin memprediksi kata target, "makan", menggunakan hanya satu input konteks "kucing". Kita menggunakan one-hot encoding dari kata input dan mengukur output error dibandingkan dengan one-hot encoding dari target kata "makan". Kata input/konteks adalah one-hot encoded vector dengan ukuran V (input layer), hidden layer berisi N neurons dan output layer berisi V . Secara garis besar, hidden layer menyalin weighted sum dari input ke layer selanjutnya (output). Di sini tak ada fungsi aktivasi (sigmoid, tanh, etc), tapi hanya perhitungan softmax di output layer.

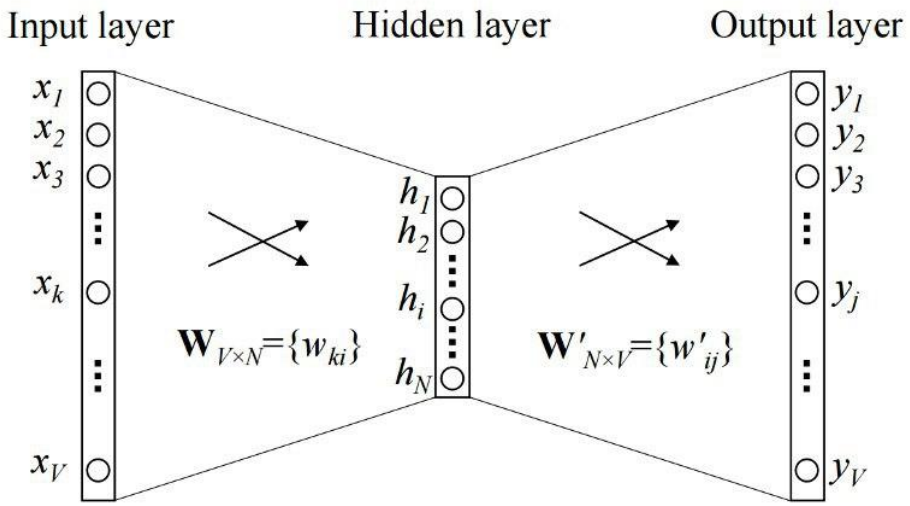
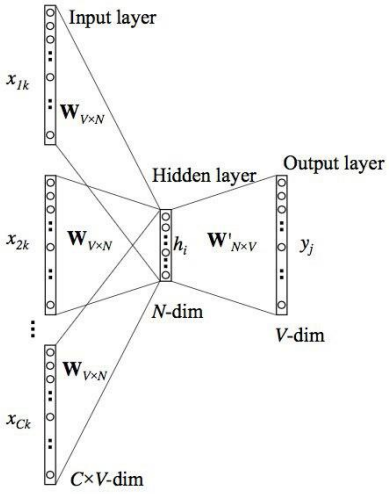
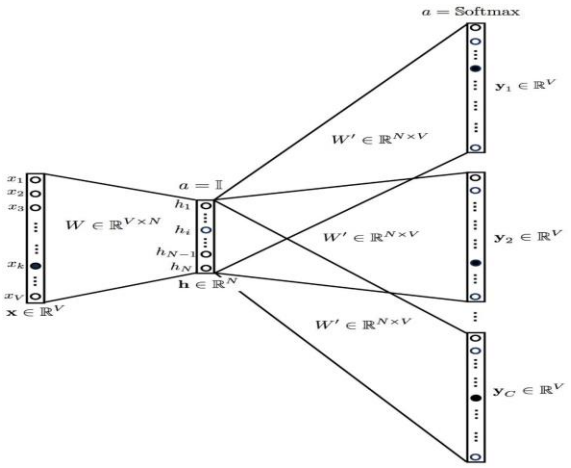



Figure 1: A simple CBOW model with only one word in the context

Arsitektur sederhana CBOW




Kita juga bisa menggunakan lebih dari 1 kata input/konteks.




skip-gram

Skip-gram adalah "kebalikan" dari CBOW. Seperti yang terlihat di gambar berikut. One-hot encoding vector dari kata target berada di input layer, dan kata konteks berada di output layer. Jadi, output dari arsitektur skip-gram adalah distribusi probabilitas dari kata-kata konteks.

$$\frac{1}{n}\sum_{t=1}^{n}\sum_{-c\leq j\leq c, j\neq 0}\log p\left(x_{t+j}|x_t\right)$$

Secara formal, kita diberikan training words x 1 , x 2 , . . . , x n , skip-gram model memiliki objektif untuk memaksimalkan rata-rata log probabilitas, di mana c adalah ukuran dari konteks. Kalo kita lihat formula di atas, kita ingat dengan penurunan perplexity, bukan? Semakin besar c menghasilkan akurasi yang semakin tinggi, dan membutuhkan training time yang lebih banyak.

Sekarang, mana yang lebih baik kita gunakan di antara keduanya?

## 3.2 CBOW vs skip-gram

Skip-gram bekerja lebih baik dibanding CBOW dengan ukuran data yang lebih kecil dan cenderung dapat merepresentasikan kata-kata yang jarang muncul dengan lebih baik. Sedangkan CBOW lebih unggul untuk merepresentasikan kata-kata yang sering muncul. Ada dua optimisasi untuk teknik ini

1. Hierarchical softmax: secara garis besar, ini digunakan untuk meningkatkan efisiensi dalam perhitungan softmax.

2. Negative sampling: untuk mengurangi banyaknya output vectors yang harus di-update setiap iterasi, maka hanya update sample dari output vectors tersebut.