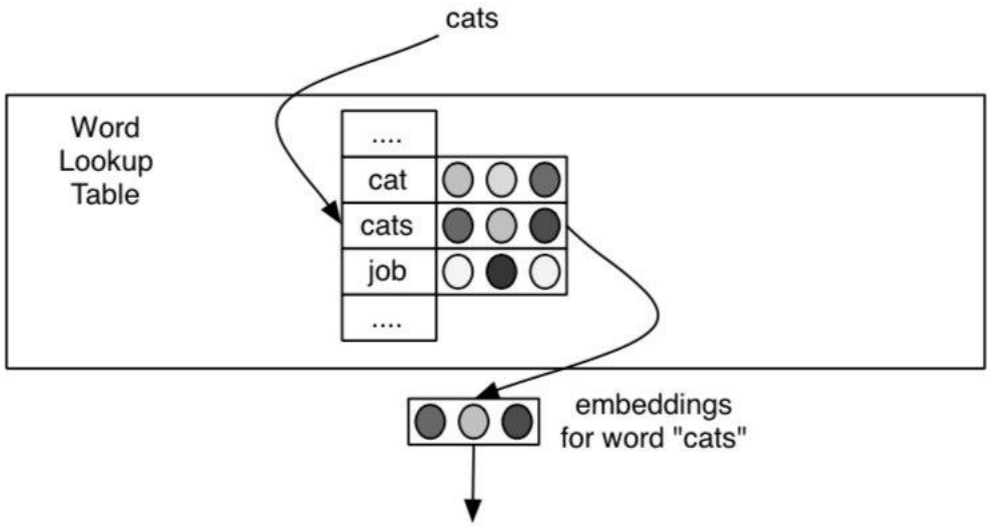
Text Classification Part 2

Assalamualaikum, halo selamat malam semuanya. Kulgram lagi dari saya ya malam ini.

Temanya tentang text classification with neural networks, lanjutan dari part 1 yg lalu. Mohon maaf dulu sebelumnya, saya ga bisa dapat banyak ilustrasi gambar2 utk tema ini tapi semoga penjelasannya tetap mudah dipahami ya. Oke, jadi langsung aja. Misalnya kita udah coba pakai non-neural methods (e.g. naive Bayes) utk text classification, dan perf-nya masih kurang memuaskan. Udah dikulik2 macem2 tetep kurang, langkah berikutnya, mungkin bisa coba pakai neural network. Salah satu keunggulan neural nets adalah representation learning-nya. Dalam artian, kalau pakai non-neural methods mungkin kita perlu pikirin mau merepresentasikan teks seperti apa. Apakah bag-of-words dgn one-hot encoding? Apakah TF-IDF? Atau lainnya? Sedangkan dgn neural networks, feature engineering seperti ini bisa ditekan sampai minimal (minimal, bukan nihil ya). Kekurangannya, tentu saja proses training yang lebih lama, kebutuhan data yg lebih besar, kemungkinan overfit yg lebih tinggi, dst. Misalnya kita udah memutuskan mau pakai neural nets, dan sudah aware dengan kelebihan dan kekurangannya. Gimana caranya? Ide besarnya adalah: representasikan teks input sebagai sebuah vektor, lalu lewatkan vektor ini ke sebuah output layer (biasanya cukup affine/linear layer dgn softmax layer di ujungnya).

Variasi2 yg ada biasanya hanya main2 di bagaimana cara merepresentasikan teks input sebagai sebuah vektor ini. Cara paling mudah adalah dengan menggunakan words/kata2 pada teks sebagai fitur, lalu meng-embed kata2 tsb menjadi vektor2, lalu meng-compose vektor2 ini menjadi sebuah vektor. Vektor terakhir inilah hasil representasi teks inputnya. Bahas satu2 ya, dari proses meng-embed dulu. Ini paling gampang. Intinya, tiap kata2 diubah jadi vektor dengan melihat ke sebuah lookup table. Lookup table ini menyimpan mapping [kata -> vektor] utk setiap kata yg ada di vocabulary/training data.



Ini ilustrasinya

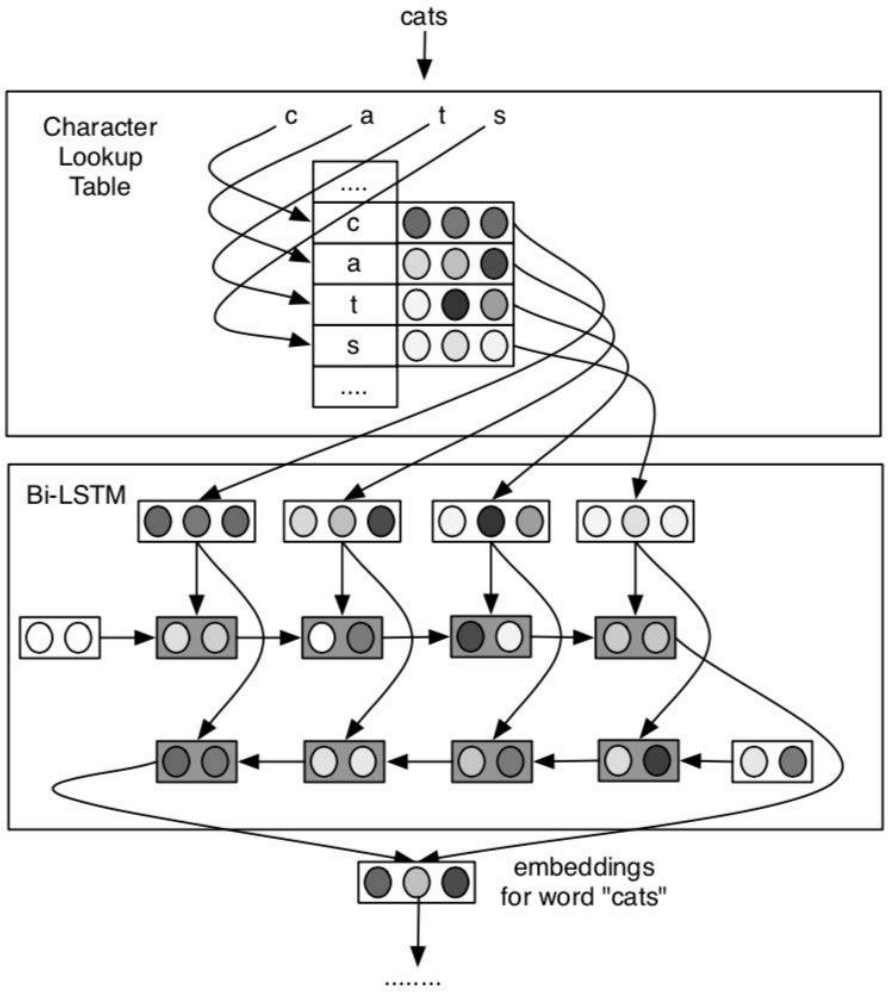
Di contoh ini, lookup table kita mengandung entri utk "cat", "cats", "job" dll. Mendapatkan representasi vektor utk "cats" adalah sesederhana melihat lookup table tsb dan mengembalikan vektor v yg sesuai dengan mapping ["cats" -> v]. Lookup table ini nanti jadi parameter neural networknya juga, jadi nilainya ikut di-learn pada saat training dgn backprop. Setelah semua kata pada teks input di-embed

jadi vektor2, langkah berikutnya adalah meng-compose vektor2 ini menjadi a single vector. Jadi ibaratnya kita perlu bikin fungsi `compose(List[Vector]) -> Vector`

Nah ini caranya bermacam2. Paling gampang bisa dijumlahin aja, atau dirata2.

Yang lebih fancy bisa dilewatin ke CNN atau LSTM. Vektor hasil `compose` ini adalah representasi vektor dari teks input. Terakhir, seperti yg udh dijelaskan di atas, vektor ini tinggal dilewatin ke linear layer + softmax aja utk dapet probability dist over classes. Loss function yg digunakan utk training biasanya cross-entropy loss. Trainingnya kyk neural network biasa aja pakai backprop. Oke jadi itu ide besarnya text classification dgn neural network (yg saya tau). Nah tapi biasanya ada lagi nih variasi2nya, terutama di bagian fitur yg digunakan. Di contoh ini tadi saya pakai fiturnya adalah kata2 dari teks input, jadi kata2 itulah yg di-embed, di-compose, dst. Bisa jadi juga kita gunakan fitur tambahan, misalnya n-gram, katakanlah bigram (2-gram). Jadi kalau ada teks "ayam bakar sinta", fitur2nya selain kata2nya "ayam", "bakar", dan "sinta", ada juga "<START>_ayam", "ayam_bakar", "bakar_sinta", "sinta_<END>". Kalau ndak salah, inilah yg digunakan oleh fastText (Joulin et al., 2016)

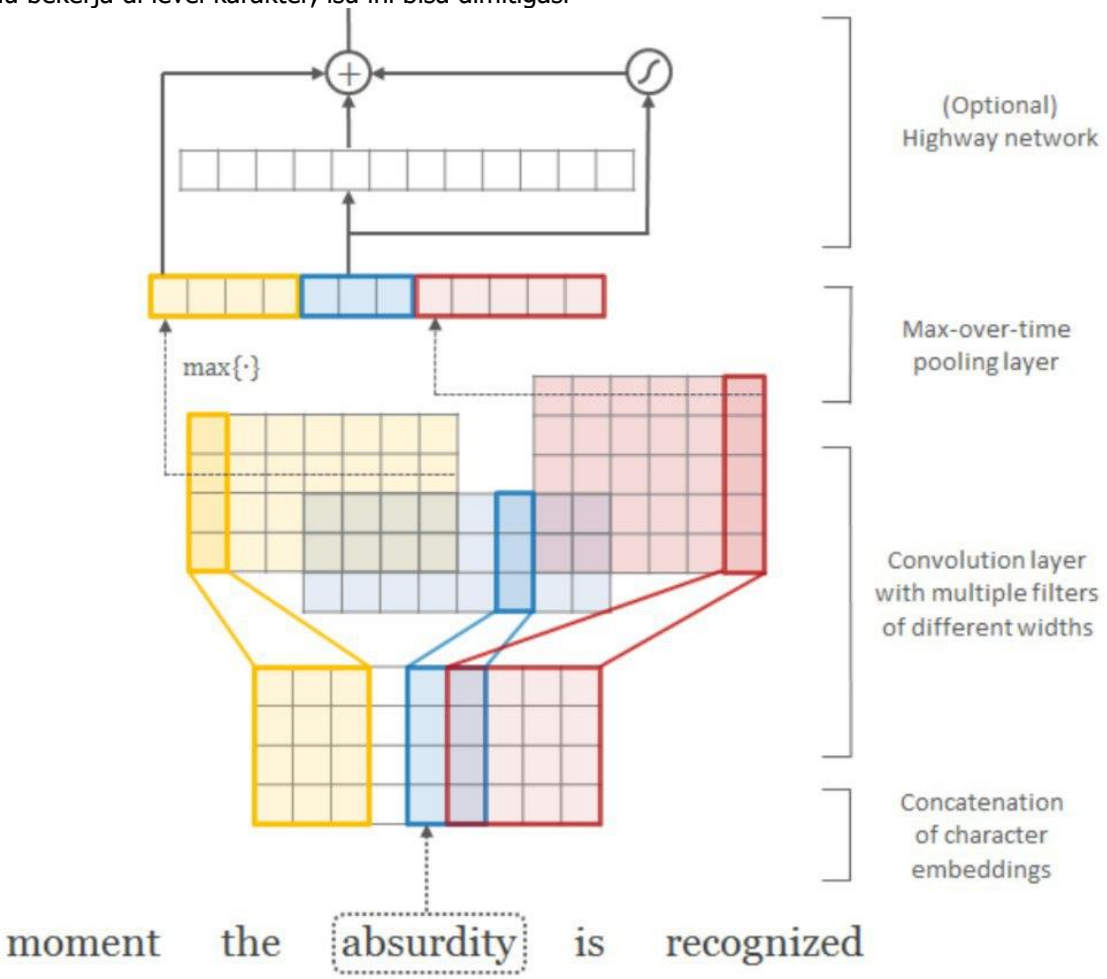
Bisa juga fitur karakter dipakai. Jadi selain tiap kata kita dapatkan representasi vektornya lewat lookup table, kita juga compose karakter2 penyusun kata tersebut jadi a single vector, lalu hasilnya katakanlah di-concat.



Misalnya begini

Perhatikan bahwa idenya masih sama: ada fitur2 yg di-embed (karakter), vektor2 hasil embedding process ini di-compose (pakai bi-LSTM). vektor hasil terakhirnya ini adalah vektor representasi kata "cats" yg disusun dari karakter2nya. vektor ini bisa digabung (di-concat/dijumlah/dll) dgn vektor representasi "cats" yg didapat dari lookup table sebelumnya utk mendapatkan representasi final dari kata "cats". Salah satu alasan kenapa fitur karakter digunakan seperti ini adalah utk mengatasi masalah out-of-vocabulary (OOV): bagaimana merepresentasikan kata2 yg tidak pernah muncul di training data? Kalau hanya mengandalkan lookup table saja, semua kata2 OOV ini jadi dikasih satu representasi yg sama, misalnya <UNK> (unknown token). Kalau kata2 pada teks input (saat test time misalnya) semuanya tdk pernah muncul di training data, modelnya jadi "buta" krn yg dia lihat hanya untaian <UNK> saja

Kalau bekerja di level karakter, isu ini bisa dimitigasi



Selain bi-LSTM, bisa juga digunakan CNN utk meng-compose, contohnya begini

Agak beda dgn di computer vision, kalau di NLP, kernel CNN-nya height-nya selalu sama dgn dimensi dari word embedding-nya (representasi vektor dari suatu word tadi). Width-nya aja yg beda2. Oiya itu di akhir setelah CNN ada highway layer ya. Ini sah2 aja. Hasil composition mau dilewatin layer apa dulu terserah. Tapi biasanya nanti ya ujungnya linear + softmax. Atau bisa juga gak pakai fitur kata2 sama sekali, anggap aja teks input itu sebagai untaian karakter. Keunggulannya, ide ini bisa diterapkan pada bahasa2 yang segmentasi kata2nya kadang sulit/tidak begitu jelas (mis. Japanese, Chinese) Ini yg dilakukan Zhang et al. (2015). Mereka pakai CNN utk meng-compose character embeddings ini.

Hasilnya juga masih oke (utk data yg besar). http://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.pdf

Sesi pertanyaan

Dari Raden Muhammad Hadi :

Kang @kmkurn , metode/algoritma apa ya yang bisa dipakai untuk mengklasifikasikan dokumen dengan target lebih dari 1? Saya ada kasus dimana sebuah dokumen diklasifikasikan dalam 4 kategori dimana 1 dokumen ini bisa aja memuat 4 kategori ini dalam suatu nilai confidence tertentu, misalnya kategori A 20%, B 30%, C 80%, D 10%.

Jawaban :

Hmm kalo saya, kalo kategorinya ga terlalu banyak, saya akan bikin binary classifier sebanyak kategorinya. Kalau ada dokumen masuk, semua binary classifier ini dijalanin utk dapet angka2 confidence itu. Binary classifiernya bisa bener2 independen utk tiap kategori, atau ada parameter sharing, bisa dicoba ke validation set mana yg lebih bagus.

Dari Yoga Yudistira:

Mas @kmkurn mau nanya, kan biasanya kita bisa ngetrain language model dulu semacam word2vec yang kemudian hasil layernya bakal dipake buat downstream task kaya text classification, apa cara kaya gini hasilnya emang pasti lebih bagus, atau terkadang ada case mending inialisasi weight embeddingnya dari awal?

Jawaban :

Hmm ini belum ada jawaban pastinya sih. Memang dari hasil2 yg ada biasanya pakai pretrained word embedding biasanya lebih bagus, tapi ga menutup kemungkinan train dari awal yg lebih bagus (salah satu paper saya begini). saran saya dicoba keduanya aja mana yg lebih bagus di validation set. atau kalau mau digabung keduanya, final word embedding-nya adalah hasil concat dari pretrained word embedding dengan learned word embedding. Dyer et al. (2016) melakukan ini. http://www.aclweb.org/anthology/N16-1024

Paper yang bersangkutan https://arxiv.org/abs/1810.05334

Dari :

Hervind Phil :

Kak mau tanya @kmkurn terkait OOV.
Ada cara untuk menambah vocab baru tanpa harus traning dari awal lagi?
Kalau transfer learning kan kasus nya untuk kategory (y) nya beda tapi cmmiw ya

Jawaban :

Pertanyaan bagus, saya juga belum tau jawabannya. Kalau boleh nebak2, transfer learning mungkin bisa digunakan. Tambahin vocabnya (i.e. lookup table word embedding-nya ditambah), dan train dari awal lagi, _tapi_ parameternya diinisialisasi dari parameter sebelumnya (i.e. ketika vocabnya belum ditambah). Harapannya meski train dari awal tp konvergennya lebih cepat dicapai.

Dari