EP 1: Word2Vec

Entrega: 25/09/2020 (14h, antes da aula)

Motivação

O objetivo[1] deste exercício-programa é aumentar a compreensão e a familiarização do mecanismo word2vec na versão *skipgram* (*salta-grama*).

Vamos relembrar rapidamente o algoritmo word2vec. O principal insight por trás do word2vec é que "diga me com quem a palavra anda e eu te direi quem a palavra é". O word2vec usa um truque: treinar uma rede neural simples com uma única camada oculta para realizar uma determinada tarefa, mas não vamos realmente usar essa rede neural para a tarefa em que a treinamos! Em vez disso, o objetivo é apenas aprender os pesos da camada oculta – veremos que esses pesos são na verdade os "vetores de palavras" que estamos tentando aprender.

Concretamente, suponha que temos uma palavra "central" c e uma janela contextual em torno de c. Devemos nos referir às palavras que se encontram nesta janela contextual como "palavras externas" Por exemplo, na Figura 1, vemos que a palavra central c é 'endereço'. Como o tamanho da janela de contexto é 2, as palavras externas estão 'correio', 'o', 'estava' e 'errado'.

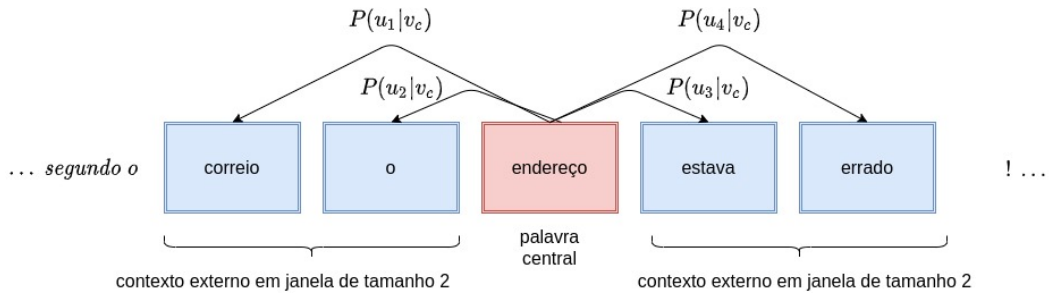

Fig. 1: O modelo de predição word2vec *skipgram* com janela de tamanho 2

No word2vec, a distribuição de probabilidade condicional de se observar a palavra o na janela contextual em torno da palavra central c é dada tomando o produto interno dos vetores u_o e v_c e aplicando a função softmax:

$$P(O = o|C = c) = \frac{\exp\left(u_o^\top \cdot v_c\right)}{\sum_{w \in Vocab} \exp(u_w^\top \cdot v_c)} \tag{1}$$

[1] Esse exercício foi baseado numa tarefa do curso de uma universidade de grande destaque internacional.

Neste caso, u_o é o vetor "externo" que representa a palavra externa o, e v_c é o vetor "central" que representa a palavra central c. Para armazenar esses parâmetros, temos duas matrizes, U e V. As colunas de U consistem dos vetores "externos" u_w. As colunas de V contêm todos os vetores "centrais" v_w. Ambos U e V contêm um vetor para cada $w \in$ *Vocabulário*; assumimos que cada palavra em nosso vocabulário corresponde a um número inteiro k, o índice da palavra no vocabulário, u_k representa tanto a k-ésima coluna de U quanto o vetor de palavras "externas" à palavra indexada por k. E v_k é tanto a k-ésima coluna de V quanto o vetor de palavra "central" para a palavra indexada por "k". Para simplificar a notação, usamos de maneira intercambiável k para referir à palavra e ao índice da palavra.

Para um único par de palavras c e o, o custo (*loss*) é dado por:

$$J_{\textit{naive-softmax}}(v_c, o, U) = -\log P(O = o|C = c). \tag{2}$$

Outra maneira de ver esse custo é como a *entropia cruzada*[2] entre a distribuição real y e a distribuição prevista $\hat{y}$. Tanto y quanto $\hat{y}$ são vetores com comprimento igual ao número de palavras no vocabulário. Além disso, a k-ésima entrada nesses vetores indica a probabilidade condicional de a k-ésima palavra ser uma palavra externa a c. A verdadeira distribuição empírica y é um vetor one-hot com 1 para a verdadeira palavra externa o e 0 em todos os outros lugares. A distribuição prevista $\hat{y}$ é a distribuição de probabilidade $P(O|C = c)$ dado pela equação (1).

> **Nota:** para refrescar a memória de vocês sobre os detalhes de cálculo, são fornecidos dois arquivos anexos ao enunciado no diretório *lembretes*:
>
> - *review-differential-calculus.pdf*, com os pontos básicos domcálculo diferencial parcial.
> - *derivatives*, com aplicações à retropopagação em redes neurais.

Parte 1: Fundamentos matemáticos do word2vec

Entregar um arquivo pdf com as contas relativas aos seguintes itens pertinentes aos fundamentos do método word2vec , e não esquecer de numerar as respostas de acordo com :

(a) Mostre que a perda/custo naive-softmax dada na Equação (2) é a mesma que a perda de entropia cruzada entre y e $\hat{y}$; ou seja, mostre que

$$-\sum_{w \in Vocab} y_w \log(\hat{y}_w) = -\log(\hat{y}_o). \tag{3}$$

Sua resposta não deve exceder uma linha.

(b) Calcule a derivada parcial de $J_{\textit{naive-softmax}}(v_c, o, U)$ em relação a v_c. Por favor escreva a resposta em termos de y, $\hat{y}$ e U.

(c) Calcule as derivadas parciais de $J_{\textit{naive-softmax}}(v_c, o, U)$ em relação a cada um dos vetores de palavras "externas", u_w's. Há dois casos: quando $w = o$, o verdadeiro vetor de palavras "externas" e $w \neq o$, para todas as outras palavras. Escreva a sua resposta em termos de y, $\hat{y}$ e v_c.

[2] O custo por entropia cruzada entre a distribuição (discreta) de probabilidade verdadeira p e outra distribuição q é: $-\sum_i \log(q_i)$.

(d) A função sigmóide é dada pela Equação (4):

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}. \tag{4}$$

Calcule a derivada de $\sigma(x)$ em relação a x, onde x é um escalar. Dica: dê sua resposta em termos $\sigma(x)$.

Amostragem Negativa

(e) Vamos considerar algumas modificações adicionais ao modelo básico de skip-gram que são importantes para realmente tornar viável o treinamento, que consiste da amostragem de palavras frequentes para diminuir o número de exemplos de treinamento. Assim, vamos modificar o objetivo da otimização com uma técnica chamada "Amostragem negativa", o que faz com que cada amostra de treinamento atualize apenas uma pequena porcentagem dos pesos do modelo.

É importante notar que a amostragem de palavras frequentes e a aplicação de Amostragem Negativa não apenas reduzem a carga computacional do processo de treinamento, mas também melhoraram a qualidade de seus vetores de palavras resultantes.

Vamos considerar a perda de Amostragem Negativa, que é uma alternativa para a Perda Naive-Softmax. Suponha que K amostras negativas (palavras) sejam retiradas do vocabulário. Por simplicidade de notação, devemos nos referir a eles como $w_1, w_2, \ldots, w_K$ e seus vetores externos como $u_1, \ldots, u_K$. Observe que $o \notin \{w_1, \ldots, w_K\}$. Para uma palavra central c e uma palavra externa o, a função de custo de amostragem negativa é dada pela soma da entropia cruzada do exemplo positivo e das entropias cruzadas das amostras negativas:

$$J_{\textit{amostra negativa}}(v_c, o, U) = -\log\Big(\sigma(u_o^\top v_c)\Big) - \sum_{k=1}^{K} \log\Big(\sigma(-u_k^\top v_c)\Big) \tag{5}$$

para uma amostra $w_1, \ldots, w_K$, onde $\sigma(\cdot)$ é a função sigmóide.

Repita as partes (b) e (c), calculando as derivadas parciais de $J_{\textit{neg-sample}}$ em relação a v_c, em relação a u_o, e em relação a uma amostra negativa u_k. Dê suas respostas em termos dos vetores u_o, v_c e u_k, onde $k \in [1, K]$. Descreva com uma frase por que esta função de custo é muito mais eficiente de calcular do que o cuato do naive softmax. Você deve ser capaz de usar sua solução da parte (d) para ajudar a calcular os gradientes necessários aqui.

(f) Suponha que a palavra central seja $c = w_t$ e a janela de contexto seja $[w_{t-m}, \ldots, w_{t-1}, w_t, w_{t+1}, \ldots, w_{t+m}]$, onde m é o tamanho da janela de contexto. Lembre-se de que, para a versão skip-gram do word2vec , o custo total da janela de contexto é:

$$J_{\textit{skip-gram}}(v_c, w_{t-m}, \ldots, w_{t+m}, U) = \sum_{\substack{-m \le j \le m \\ j \neq 0}} J(v_c, w_{t+j}, U) \tag{6}$$

onde $J(v_c, w_{t+j}, U)$ representa um termo de custo arbitrário para a palavra central $c = w_t$ e palavra externa w_{t+j}. Este custo pode ser $J_{naive\text{-}softmax}(v_c, w_{t+j}, U)$ ou $J_{neg\text{-}sample}(v_c, w_{t+j}, U)$, dependendo da sua implementação. Escreva três derivados parciais:

(i) $\dfrac{\partial J_{skip\text{-}gram}(v_c, w_{t-m}, \ldots, w_{t+m}, U)}{\partial U}$

(ii) $\dfrac{\partial J_{skip\text{-}gram}(v_c, w_{t-m}, \ldots, w_{t+m}, U)}{\partial v_c}$

(iii) $\dfrac{\partial J_{skip\text{-}gram}(v_c, w_{t-m}, \ldots, w_{t+m}, U)}{\partial v_w}$, para $w \neq c$.

Dê suas respostas em termos de $\dfrac{\partial J_{skip\text{-}gram}(v_c, w_{t+j}, U)}{\partial U}$ e $\dfrac{\partial J_{skip\text{-}gram}(v_c, w_{t+j}, U)}{\partial v_c}$.

Isso deve ser muito simples – cada solução deve ser uma linha.

Depois de fazer isso: Dado que você calculou as derivadas de $J_{skip\text{-}gram}(v_c, w_{t+j}, U)$ em relação a todos os parâmetros do modelo U e V nas partes (a) a (c), você agora calculou as derivadas da função de custo total $J_{skip\text{-}gram}$ em relação a todos os parâmetros. Você está pronto para implementar o word2vec!

Parte 2: Codificação do word2vec

Nesta parte você irá implementar o modelo word2vec e treinar seus próprios vetores de palavras com gradiente estocástico descendente (SGD). Para cada um dos métodos que você precisa implementar, incluímos aproximadamente quantas linhas de código nosso solução tem nos comentários do código. Esses números são incluídos para orientá-lo. Você não tem que se limitar a eles, você pode escrever um código mais curto ou mais longo como desejar. Se você acha que sua implementação é significativa mais longa do que nossa sugestão, é um sinal de que existem alguns métodos de biblioteca (e.g. `numpy`) que você pode utilizar para tornar seu código mais curto e mais rápido. Os loops `for` em Python demoram muito para serem concluídos quando usados em vetores e matrizes grandes, então esperamos que você utilize métodos `numpy`.

Pede-se:

(a) Primeiro, implemente o método `sigmoid`, que recebe um vetor e aplica a função sigmóide a ele. Em seguida, implemente o custo e o gradiente softmax no método `naiveSoftmaxLossAndGradient` e o custo da amostragem negativa e gradiente no método `negSamplingLossAndGradient`. Finalmente, preencha a implementação para o modelo skip-gram no método `skipgram`. Quando terminar, teste sua implementação executando `python word2vec.py`.

(b) Conclua a implementação para seu otimizador SGD no método `sgd` do arquivo *sgd.py*. Teste sua implementação executando `python sgd.py`.

(c) Hora da verdade! Agora vamos carregar alguns dados reais e treinar vetores de palavras com tudo você acabou de implementar! Vamos usar o conjunto de dados do córpus da B2W para treinar vetores word2vec. Você precisará baixar os conjuntos de dados nesse endereço https://github.com/b2wdigital/b2w-reviews01/blob/

master/B2W-Reviews01.csv para a pasta `dados`. Não há código adicional a ser escrito para esta parte; somente execute `python run.py`.

Observação: *o processo de treinamento pode demorar muito, dependendo da eficiência de sua implementação e o poder de computação de sua máquina (uma implementação eficiente leva de uma a duas horas). Planeje de acordo!*

(d) Após o treinamento (40.000 iterações), o script será concluído e uma visualização para alguns vetores de palavras por nós selecionados aparecerá. Também será salvo como *vectores_de_palavras.png* no diretório do projeto. Inclua o ARQUIVO PNG em sua entrega. Explique resumidamente em no máximo três frases o que você vê na imagem.

Utilizando as máquinas do IME

Para realizar o treinamento em um computador laptop normal, é possível que o tempo em que esse treinamento leve várias horas.

No IME temos um conjunto de máquinas poderosas a disposição dos alunos:

https://wiki.ime.usp.br/servicos:processamento

Como acessar a rede ime, ssh e permissões de conta:

https://wiki.ime.usp.br/tutoriais:como_acessar_a_rede_ime_de_fora

Instruções entrega

Entregar um zip contendo 5 arquivos:

1. Um pdf com as respostas da parte 1.
2. O arquivo `wordvec.py` acrescido com seu código
3. O arquivo `sgd.py` acrescido com seu código
4. A saída do treinamento, o arquivo *vetores_de_palavras.png*
5. Um outro arquivo de comentários gerais que achar pertinente, incluindo explicação resumida em no máximo três frases sobre o que você vê na imagem png gerada.