

Softmax (10 points)

task: prove $\text{softmax}(x) = \text{softmax}(x + c)$

(a) (5 points) Prove that softmax is invariant to constant offsets in the input, that is, for any input vector x and any constant c ,

$$\text{softmax}(x) = \text{softmax}(x + c)$$

where $x + c$ means adding the constant c to every dimension of x . Remember that

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

_Note: In practice, we make use of this property and choose $c = -\max_i x_i$ when computing softmax probabilities for numerical stability (i.e., subtracting its maximum element from all elements of x)._

solution

For a fixed i in equation (1) and words from $1, \dots, \dim(x)$:

$$\begin{aligned} \text{softmax}(x)_i &= \frac{\exp(x_i)}{\sum_{j=1}^{\dim(x)} \exp(x_j)} \quad \Bigg| \quad \cdot \frac{\exp(c)}{\exp(c)}, c \in \mathbb{R} \\ &= \frac{\exp(x_i + c)}{\sum_{j=1}^{\dim(x)} \exp(x_j + c) \exp(c)} = \text{softmax}(x + c)_i \quad \square \end{aligned}$$

task: implement softmax in python

(b) (5 points) Given an input matrix of N rows and D columns, compute the softmax prediction for each row using the optimization in part (a). Write your implementation in `q1_softmax.py`. You may test by executing `python q1_softmax.py`.

Note: The provided tests are not exhaustive. Later parts of the assignment will reference this code so it is important to have a correct implementation. Your implementation should also be efficient and vectorized whenever possible (i.e., use numpy matrix operations rather than for loops). A non-vectorized implementation will not receive full credit!

code `q1_softmax.py`

Neural Network Basics (30 points)

task: derive the gradients \rightarrow sigmoid function

(a) (3 points) Derive the gradients of the sigmoid function and show that it can be rewritten as a function of the function value (i.e., in some expression where only $\sigma(x)$, but not x , is present). Assume that the input x is a scalar for this question. Recall, the sigmoid function is

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \quad (2)$$

solution

$$\begin{aligned} f(x) &= \frac{u}{v} & f'(x) &= \frac{u'v - uv'}{v^2} \\ u &= 1 & u' &= 0 \\ v &= 1 + \exp(-x) & v' &= -\exp(-x) \end{aligned}$$

$$\begin{aligned} \sigma'(x) &= \frac{\partial}{\partial x} \frac{1}{1 + \exp(-x)} \\ &= \frac{\overbrace{0 \cdot (1 + \exp(-x))}^{u'v} - \overbrace{1 \cdot (-\exp(-x))}^{uv'}}{\underbrace{(1 + \exp(-x))^2}_{v^2}} \quad \left| \text{ simplify nominator} \right. \\ &= \frac{\exp(-x)}{(1 + \exp(-x))^2} \quad \left| \text{ split denominator, add and subtract 1 (nominator of second factor)} \right. \\ &= \frac{1}{1 + \exp(-x)} \frac{1 + \exp(-x) - 1}{1 + \exp(-x)} \quad \left| \text{ split second factor} \right. \\ &= \frac{1}{1 + \exp(-x)} \left(\frac{1 + \exp(-x)}{1 + \exp(-x)} - \frac{1}{1 + \exp(-x)} \right) \quad \left| \text{ reduce} \right. \\ &= \frac{1}{1 + \exp(-x)} \left(1 - \frac{1}{1 + \exp(-x)} \right) \quad \left| \text{ use sigmoid definition} \right. \\ &= \sigma(x)(1 - \sigma(x)) \end{aligned}$$

derive the gradient w.r.t. $x \rightarrow$ cross entropy using softmax function

(3 points) Derive the gradient with regard to the inputs of a softmax function when cross entropy loss is used for evaluation, i.e., find the gradients with respect to the softmax input vector θ , when the prediction is made by $\hat{y} = \text{softmax}(\theta)$. Remember the cross entropy function is

$$CE(y, \hat{y}) = - \sum_i y_i \log(\hat{y}_i) \quad (3)$$

where y is the one-hot label vector, and \hat{y} is the predicted probability vector for all classes. (Hint: you might want to consider the fact many elements of y are zeros, and assume that only the k -th dimension of y is one.)

short solution

$$\frac{\partial}{\partial x_k} CE'(y, \hat{y}) = \hat{y} - y = \begin{cases} \hat{y}_k - 1 & , \text{if } k \text{ is correct class} \\ \hat{y}_k & , \text{otherwise} \end{cases}$$

solution

first step: derive softmax

$$\text{softmax}(\theta) = \hat{y}_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)} = f(x)_i$$

$$\begin{aligned} f(x) &= \frac{u}{v} & f'(x) &= \frac{u'v - uv'}{v^2} \\ u &= \exp(x_i) & u' &= \begin{cases} \exp(x_i) & , i = k \\ 0 & , i \neq k \end{cases} \\ v &= \sum_j \exp(x_j) & v' &= \exp(x_k) \end{aligned}$$

case $i = k$:

$$\begin{aligned} \frac{\partial}{\partial x_k} f(x)_i &= \frac{\exp(x_i) \sum_j \exp(x_j) - \exp(x_i) \exp(x_k)}{(\sum_j \exp(x_j))^2} & \Bigg| & \text{factorise } \exp(x_i), \text{ use } i=k \\ &= \frac{\exp(x_i) (\sum_j \exp(x_j) - \exp(x_i))}{(\sum_j \exp(x_j))^2} & \Bigg| & \text{split denominator} \\ &= \frac{\exp(x_i)}{\sum_j \exp(x_j)} \frac{\sum_j \exp(x_j) - \exp(x_i)}{\sum_j \exp(x_j)} & \Bigg| & \text{split second term} \\ &= \frac{\exp(x_i)}{\sum_j \exp(x_j)} \left(\frac{\sum_j \exp(x_j)}{\sum_j \exp(x_j)} - \frac{\exp(x_i)}{\sum_j \exp(x_j)} \right) & \Bigg| & \text{reduce} \\ &= \frac{\exp(x_i)}{\sum_j \exp(x_j)} \left(1 - \frac{\exp(x_i)}{\sum_j \exp(x_j)} \right) & \Bigg| & \text{use softmax definition} \\ &= \hat{y}_i (1 - \hat{y}_i) \end{aligned}$$

$$f(x) = \frac{u}{v} \quad f'(x) = \frac{u'v - uv'}{v^2}$$

$$u = \exp(x_i) \quad u' = \begin{cases} \exp(x_i) & , i = k \\ 0 & , i \neq k \end{cases}$$

$$v = \sum_j \exp(x_j) \quad v' = \exp(x_k)$$

case $i \neq k$:

$$\begin{aligned} \frac{\partial}{\partial x_k} f(x)_i &= \frac{0 \cdot \sum_j \exp(x_j) - \exp(x_i) \exp(x_k)}{(\sum_j \exp(x_j))^2} \quad \left| \text{ simplify nominator} \right. \\ &= \frac{-\exp(x_i) \exp(x_k)}{(\sum_j \exp(x_j))^2} \quad \left| \text{ split denominator} \right. \\ &= \frac{-\exp(x_i)}{\sum_j \exp(x_j)} \frac{\exp(x_k)}{\sum_j \exp(x_j)} \quad \left| \text{ use softmax definition} \right. \\ &= -\hat{y}_i \hat{y}_k \end{aligned}$$

$$\frac{\partial}{\partial x_k} f(x)_i = \begin{cases} \hat{y}_i(1 - \hat{y}_i) & , i = k \\ -\hat{y}_i \hat{y}_k & , i \neq k \end{cases}$$


second step: derive cross-entropy

$$\begin{aligned} CE(y, \hat{y}) &= -\sum_i y_i \log(\hat{y}_i) \\ \frac{\partial}{\partial x_k} CE(y, \hat{y}) &= \frac{\partial}{\partial x_k} -\sum_i y_i \log(\hat{y}_i) \quad \left| \text{ chain rule, } f(x) = \log(x) \rightarrow f'(x) = \frac{1}{x} \right. \\ &= -\sum_i y_i \frac{1}{\hat{y}_i} \frac{\partial}{\partial x_k} \hat{y}_i \quad \left| \hat{y}_i = f(x)_i, \text{ use softmax derivation ib., split sum for bot} \right. \\ &= -\sum_{i=k} y_i \frac{1}{\hat{y}_i} \cdot \hat{y}_i(1 - \hat{y}_i) - \sum_{i \neq k} y_i \frac{1}{\hat{y}_i} \cdot (-\hat{y}_i \hat{y}_k) \quad \left| \text{ simplify, case } i=k \text{ is no sum a} \right. \\ &= -y_k(1 - \hat{y}_k) + \sum_{i \neq k} y_i \hat{y}_k \quad \left| \text{ expand} \right. \\ &= -y_k + y_k \hat{y}_k + \sum_{i \neq k} y_i \hat{y}_k \quad \left| \text{ combine 2nd term with the sum} \right. \\ &= \hat{y}_k \sum_i y_i - y_k \quad \left| \sum_i y_i \text{ is 1 because of one-hot-vector, simplify and ignore th} \right. \\ &= \hat{y}_k - y \end{aligned}$$

$$\frac{\partial}{\partial x_k} CE'(y, \hat{y}) = \hat{y}_k - y = \begin{cases} \hat{y}_k - 1 & , \text{ if } k \text{ is correct class} \\ \hat{y}_k & , \text{ otherwise} \end{cases}$$

derive the gradients → neural net

(6 points) Derive the gradients with respect to the inputs x to an one-hidden-layer neural network (that is, find $\frac{\partial J}{\partial x}$ where $J = CE(y, \hat{y})$ is the cost function for the neural network). The neural network employs sigmoid activation function for the hidden layer, and softmax for the output layer. Assume the one-hot label vector is y , and cross entropy cost is used. (Feel free to use $\sigma'(x)$ as the shorthand for sigmoid gradient, and feel free to define any variables whenever you see fit.)

 alt text

Recall that the forward propagation is as follows

$$h = \text{sigmoid}(xW_1 + b_1), \quad \hat{y} = \text{softmax}(hW_2 + b_2)$$

Note that here we're assuming that the input vector (thus the hidden variables and output probabilities) is a row vector to be consistent with the programming assignment. When we apply the sigmoid function to a vector, we are applying it to each of the elements of that vector. W_i and b_i ($i = 1, 2$) are the weights and biases, respectively, of the two layers.

solution

$$J = CE(y, \hat{y})$$

$$\frac{\partial J}{\partial x}$$

$$\begin{array}{lcl} \delta_1 = \frac{\partial CE}{\partial z_2} = \hat{y} - y & \Bigg| & \hat{y} = \text{softmax}(z_2) \\ \delta_2 = \frac{\partial CE}{\partial h} = \frac{\partial CE}{\partial z_2} \frac{\partial z_2}{\partial h} = \delta_1 W_2^T & \Bigg| & z_2 = hW_2 + b_2 \\ \delta_3 = \frac{\partial CE}{\partial z_1} = \frac{\partial CE}{\partial z_2} \frac{\partial z_2}{\partial h} \frac{\partial h}{\partial z_1} = \delta_2 \sigma'(z_1) & \Bigg| & h = \text{sigmoid}(z_1) = \sigma(z_1) \\ \delta_4 = \frac{\partial CE}{\partial x} = \frac{\partial CE}{\partial z_2} \frac{\partial z_2}{\partial h} \frac{\partial h}{\partial z_1} \frac{\partial z_1}{\partial x} = \delta_3 W_1^T & \Bigg| & z_1 = xW_1 + b_1 \end{array}$$

$$\frac{\partial J}{\partial x} = \frac{\partial CE}{\partial x} = (\hat{y} - y) W_2^T \sigma'(z_1) W_1^T$$

How many parameters has this neural network?

(2 points) How many parameters are there in this neural network, assuming the input is D_x -dimensional, the output is D_y -dimensional, and there are H hidden units?

solution

parameters are

- input dimension D_x
- output dimension D_y
- hidden units H
- bias $\rightarrow +1$ for input and output dimension

$$p = \underbrace{(D_x + 1) \cdot H}_{\text{layer 1+2}} + \underbrace{(H + 1) \cdot D_y}_{\text{layer 2+3}}$$

task: fill in implementation q2_sigmoid.py

(4 points) Fill in the implementation for the sigmoid activation function and its gradient in `q2_sigmoid.py`. Test your implementation using `python q2_sigmoid.py`. Again, thoroughly test your code as the provided tests may not be exhaustive.

solution q2_sigmoid.py

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \quad (2)$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

task: fill in implementation q2_gradcheck.py

(4 points) To make debugging easier, we will now implement a gradient checker. Fill in the implementation for `gradcheck_naive` in `q2_gradcheck.py`. Test your code using `python q2_gradcheck.py`.

solution q2_gradcheck.py

task: fill in implementation q2_neural.py (neural net with 1 hidden layer using sigmoid activation)

(8 points) Now, implement the forward and backward passes for a neural network with one sigmoid hidden layer. Fill in your implementation in `q2_neural.py`. Sanity check your implementation with `python q2_neural.py`.

solution q2_neural.py

word2vec (40 points + 2 bonus)

derive the gradients w.r.t. $v_c \rightarrow$ cross-entropy using softmax function

(3 points) Assume you are given a predicted word vector v_c corresponding to the center word c for skipgram, and word prediction is made with the softmax function found in word2vec models

$$\hat{y}_o = p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^W \exp(u_w^T v_c)} \quad (4)$$

where w denotes the w -th word and $u_w (w = 1, \dots, W)$ are the “output” word vectors for all words in the vocabulary. Assume cross entropy cost is applied to this prediction and word o is the expected word (the o -th element of the one-hot label vector is one), derive the gradients with respect to v_c .

Hint: It will be helpful to use notation from question 2. For instance, letting \hat{y} be the vector of softmax predictions for every word, y as the expected word vector, and the loss function

$$J_{softmax-CE}(o, v_c, U) = CE(y, \hat{y}) \quad (5)$$

where $U = [u_1, u_2, \dots, u_W]$ is the matrix of all the output vectors. Make sure you state the orientation of your vectors and matrices.

short solution

$$J_{softmax-CE} = - \sum_{i=1}^W y_i \log \left(\frac{\exp(z_i)}{\sum_{w=1}^W \exp(z_w)} \right)$$

$$\frac{\partial J_{softmax-CE}}{\partial v_c} = U^T (\hat{y} - y)$$

long solution 1

$$CE(y, \hat{y}) = - \sum_i y_i \log(\hat{y}_i)$$

Let $z_i = u_i^T v_c$,
then $z = U^T v_c$

$$J_{softmax-CE} = - \sum_{i=1}^W y_i \log \left(\frac{\exp(u_o^T v_c)}{\sum_{w=1}^W \exp(u_w^T v_c)} \right)$$

$$= - \sum_{i=1}^W y_i \log \left(\frac{\exp(z_i)}{\sum_{w=1}^W \exp(z_w)} \right)$$

$$\frac{\partial J_{softmax-CE}}{\partial v_c} = \frac{\partial J_{softmax-CE}}{\partial z} \frac{\partial z}{\partial v_c}$$

$$= \frac{\partial J_{softmax-CE}}{\partial z} \frac{\partial U^T v_c}{\partial v_c}$$

$$= U^T (\hat{y} - y)$$

long solution 2

$ W $..	all words	
y, \hat{y}	..	column vector	$ W \times 1$
u_i, v_j	..	column vector	$D \times 1$
y	..	one-hot-column vector	$ W \times 1$
\hat{y}	..	softmax output	$ W \times 1$
\hat{y}_i	$= p(i c) = \frac{\exp(u_i^T v_c)}{\sum_{w=1}^W \exp(u_w^T v_c)}$		
J	$= - \sum_{i=1}^W y_i \log(\hat{y}_i)$		
U	$= [u_1, u_2, \dots, u_k, \dots, u_W] \quad \dots \quad u_k \text{ column vectors}$		

$$\begin{aligned}
 J_{softmax-CE} &= - \sum_{i=1}^W y_i \log \left(\frac{\exp(u_i^T v_c)}{\sum_{w=1}^W \exp(u_w^T v_c)} \right) \\
 &= - \sum_{i=1}^W y_i [\log(\exp(u_i^T v_c)) - \log(\sum_{w=1}^W \exp(u_w^T v_c))] \\
 &= - \sum_{i=1}^W y_i [u_i^T v_c - \log(\sum_{w=1}^W \exp(u_w^T v_c))] \quad \left| \quad \text{one-hot, } \sum_{i=1}^W y_i = 1 \right. \\
 &= - [u_k^T v_c - \log(\sum_{w=1}^W \exp(u_w^T v_c))] \\
 \frac{\partial J_{softmax-CE}}{\partial v_c} &= - [u_k^T - \frac{\partial}{\partial v_c} \log(\sum_{w=1}^W \exp(u_w^T v_c))] \quad \left| \quad f(x) = \log(x), f'(x) = \frac{1}{x} \right. \\
 &= - [u_k - \frac{1}{(\sum_{x=1}^W \exp(u_x^T v_c))} \frac{\partial}{\partial v_c} (\sum_{w=1}^W \exp(u_w^T v_c))] \\
 &= - [u_k - \frac{\sum_{w=1}^W \exp(u_w^T v_c) u_w}{(\sum_{x=1}^W \exp(u_x^T v_c))}] \\
 &= \sum_{w=1}^W \frac{\exp(u_w^T v_c)}{(\sum_{x=1}^W \exp(u_x^T v_c))} u_w - u_k \quad \left| \quad \text{recognize softmax} \right. \\
 &= \sum_{w=1}^W \hat{y}_w u_w - u_k
 \end{aligned}$$

derive the gradients w.r.t. U , or $u_w \rightarrow$ cross-entropy using softmax function

(3 points) As in the previous part, derive gradients for the “output” word vectors u_w ’s (including u_o).

short

$$\frac{\partial J_{\text{softmax-CE}}}{\partial U} = v_c(\hat{y} - y)^T$$

long version

$$\begin{aligned} \frac{\partial J_{\text{softmax-CE}}}{\partial U} &= \frac{\partial J_{\text{softmax-CE}}}{\partial z} \frac{\partial z}{\partial U} \\ &= \frac{\partial J_{\text{softmax-CE}}}{\partial z} \frac{\partial U^T v_c}{\partial U} \\ &= v_c(\hat{y} - y)^T \end{aligned}$$

derive the gradients w.r.t. $v_c, u_o, u_k \rightarrow$ negative sampling

(6 points) Repeat part (a) and (b) assuming we are using the negative sampling loss for the predicted vector v_c , and the expected output word is o . Assume that K negative samples (words) are drawn, and they are $1, \dots, K$, respectively for simplicity of notation ($o \notin \{1, \dots, K\}$). Again, for a given word, o , denote its output vector as u_o . The negative sampling loss function in this case is

$$J_{\text{neg_sample}}(o, v_c, U) = -\log(\sigma(u_o^T v_c)) - \sum_{k=1}^K \log(\sigma(-u_k^T v_c))$$

where $\sigma(\cdot)$ is the sigmoid function.

After you've done this, describe with one sentence why this cost function is much more efficient to compute than the softmax-CE loss (you could provide a speed-up ratio, i.e., the runtime of the softmax-CE loss divided by the runtime of the negative sampling loss).

Note: the cost function here is the negative of what Mikolov et al had in their original paper, because we are doing a minimization instead of maximization in our code.

solution

$$\begin{aligned} \frac{\partial J_{\text{neg_sample}}}{\partial v_c} &= (\sigma(u_o^T v_c) - 1) \cdot u_o - \sum_{k=1}^K (\sigma(-u_k^T v_c) - 1) \cdot u_k \\ \frac{\partial J_{\text{neg_sample}}}{\partial u_o} &= (\sigma(u_o^T v_c) - 1) \cdot v_c \\ \frac{\partial J_{\text{neg_sample}}}{\partial u_k} &= -(\sigma(-u_k^T v_c) - 1) \cdot (v_c) \end{aligned}$$

$$\begin{aligned}
\frac{\partial J_{neg_sample}}{\partial v_c} &= \frac{\partial}{\partial v_c} \left(-\log(\sigma(u_o^T v_c)) - \sum_{k=1}^K \log(\sigma(-u_k^T v_c)) \right) \\
&= -\frac{1}{\sigma(u_o^T v_c)} \cdot \overbrace{\sigma(u_o^T v_c)(1 - \sigma(u_o^T v_c))}^{\sigma'(\cdot)} \cdot u_o - \sum_{k=1}^K \frac{1}{\sigma(-u_k^T v_c)} \cdot \overbrace{(\sigma(-u_k^T v_c))(1 - \sigma(-u_k^T v_c))}^{\sigma'(\cdot)} \\
&= -(1 - \sigma(u_o^T v_c)) \cdot u_o - \sum_{k=1}^K (1 - \sigma(-u_k^T v_c)) \cdot (-u_k) \\
&= (\sigma(u_o^T v_c) - 1) \cdot u_o - \sum_{k=1}^K (\sigma(-u_k^T v_c) - 1) \cdot u_k \\
\frac{\partial J_{neg_sample}}{\partial u_o} &= (\sigma(u_o^T v_c) - 1) \cdot v_c - 0 \\
&= (\sigma(u_o^T v_c) - 1) \cdot v_c \\
\frac{\partial J_{neg_sample}}{\partial u_k} &= -0 - \frac{1}{\sigma(-u_k^T v_c)} \cdot \overbrace{(\sigma(-u_k^T v_c))(1 - \sigma(-u_k^T v_c))}^{\sigma'(\cdot)} \cdot (-v_c) \\
&= -(\sigma(-u_k^T v_c) - 1) \cdot (v_c)
\end{aligned}$$



derive the gradients w.r.t. all word vectors → skip-gram, cbow

(8 points) Derive gradients for all of the word vectors for skip-gram and CBOW given the previous parts and given a set of context words $[\text{word}_{c-m}, \dots, \text{word}_{c-1}, \text{word}_c, \text{word}_{c+1}, \dots, \text{word}_{c+m}]$, where m is the context size. Denote the “input” and “output” word vectors for word_k as v_k and u_k respectively.

Hint: feel free to use $F(o, v_c)$ (where o is the expected word) as a placeholder for the $J_{softmax-CE}(o, v_c, \dots)$ or $J_{neg-sample}(o, v_c, \dots)$ cost functions in this part — you’ll see that this is a useful abstraction for the coding part. That is, your solution may contain terms of the form $\frac{\partial F(o, v_c)}{\partial \dots}$.

Recall that for skip-gram, the cost for a context centered around c is

$$J_{skip-gram}(\text{word}_{c-m\dots c+m}) = \sum_{-m \leq j \leq m, j \neq 0} F(w_{c+j}, v_c)$$

where w_{c+j} refers to the word at the j -th index from the center. CBOW is slightly different. Instead of using v_c as the predicted vector, we use \hat{v} defined below. For (a simpler variant of) CBOW, we sum up the input word vectors in the context

$$\hat{v} = \sum_{-m \leq j \leq m, j \neq 0} v_{c+j}$$

then the CBOW cost is

$$J_{CBOW}(\text{word}_{c-m\dots c+m}) = F(w_c, \hat{v})$$

_Note: To be consistent with the \hat{v} notation such as for the code portion, for skip-gram $\hat{v} = v_c$._

solution

For the sake of clarity, we will denote U as the collection of all output vectors for all words in the vocabulary. Given a cost function F , we already know how to obtain the following derivatives

$$\frac{\partial F(w_i, \hat{v})}{\partial U} \quad \text{and} \quad \frac{\partial F(w_i, \hat{v})}{\partial \hat{v}}$$

therefore for skip-gram, the gradients for the cost of one context window are

$$\begin{aligned} \frac{\partial J_{skip-gram}(\text{word}_{c-m\dots c+m}, \hat{v})}{\partial U} &= \sum_{-m \leq j \leq m, j \neq 0} \frac{\partial F(w_{c+j}, v_c)}{\partial U} \\ \frac{\partial J_{skip-gram}(\text{word}_{c-m\dots c+m}, \hat{v})}{\partial v_c} &= \sum_{-m \leq j \leq m, j \neq 0} \frac{\partial F(w_{c+j}, v_c)}{\partial v_c} \\ \frac{\partial J_{skip-gram}(\text{word}_{c-m\dots c+m}, \hat{v})}{\partial v_j} &= 0, \text{ for all } j \neq c. \end{aligned}$$

Similarly for CBOW, we have

$$\begin{aligned} \frac{\partial J_{CBOW}(\text{word}_{c-m\dots c+m})}{\partial U} &= \frac{\partial F(w_c, \hat{v})}{\partial U}, \text{ (using the definition of } \hat{v} \text{ in the problem)} \\ \frac{\partial J_{CBOW}(\text{word}_{c-m\dots c+m})}{\partial v_c} &= \frac{\partial F(w_c, \hat{v})}{\partial \hat{v}} \\ \frac{\partial J_{CBOW}(\text{word}_{c-m\dots c+m})}{\partial v_j} &= 0, \text{ for all } j \notin \{c-m, \dots, c-1, c+1, \dots, c+m\}. \end{aligned}$$

q3 word2vec.py

(12 points) In this part you will implement the word2vec models and train your own word vectors with stochastic gradient descent (SGD). First, write a helper function to normalize rows of a matrix in `q3_word2vec.py`. In the same file, fill in the implementation for the softmax and negative sampling cost and gradient functions. Then, fill in the implementation of the cost and gradient functions for the skip-gram model. When you are done, test your implementation by running `python q3_word2vec.py`.

Note: If you choose not to implement CBOW (part h), simply remove the `NotImplementedError` so that your tests will complete.

q3_sgd.py

q3_run.py

cbow in q3_word2vec.py

Sentiment Analysis

Now, with the word vectors you trained, we are going to perform a simple sentiment analysis. For each sentence in the Stanford Sentiment Treebank dataset, we are going to use the average of all the word vectors in that sentence as its feature, and try to predict the sentiment level of the said sentence. The sentiment level of the phrases are represented as real values in the original dataset, here we'll just use five classes:

"very negative (--)", "negative (-)", "neutral", "positive (+)", "very positive (++)"

which are represented by 0 to 4 in the code, respectively. For this part, you will learn to train a softmax classifier, and perform train/dev validation to improve generalization.

q4_sentiment.py

Explain...

... (1 points) Explain in at most two sentences why we want to introduce regularization when doing classification (in fact, most machine learning tasks)

solution

Regularization generally helps to maintain some control over the weights and thus prevent overfitting. This then helps the model in the generalization aspect for new data.