Assignment 1 Sol. For CS 224D

by Ashutosh Kumar Nirala

4 Sentiment Analysis (20 points)

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 well 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 regressor with SGD, and perform train/dev validation to improve generalization of your regressor.

(a) (10 points) Implement a sentence featurizer and softmax regression. Fill in the implementation in `q4_softmaxreg.py`. Sanity check your implementation with python `q4_softmaxreg.py`.
Consider:
N: Number of examples. In the code its value is 10.
C: Number of classes, in our example it is 5.
L: Number of dimension of each X. X: Input. It is N×C matrix.
W: Weight matrix to be learned. It would be L×C
y: One-hot vector representation, of the output class (for input x)
$\hat{y}$: represents the calculated one-hot output vector. Y: One-hot matrix, where each row is one-hot vector y and has N rows and C columns

Let's try to calculate J (cost):
Consider an input x, xW would give us a row vector, indicating the score for each of the class.
Now, we would normalize it via softmax (as asked in the question) to get $\hat{y}$:
$\hat{y}$ = softmax(xW). Please note that x is a row vector
Using cross entropy as our loss, loss for an example x would be:
loss = -Σ_l^L y_l
Now, only the component which is 1 in the output, or the class to which x belongs would survive. Thus the total loss would be summation over all the input.
Let Ψ = log(softmax(XW)), It's dimension is N×C, Here softmax is applied row wise and log, element wise.
So loss = sum (np.multiply(ΨY)), here sum is over N×C terms. Please note multiplication is element wise. Also `q4_softmaxreg.py` achieve this by simply indexing over y.
Now let's also add the regularization term to it, which is: $|W|^2/2$. Thus we get J as:
$\boxed{\text{J = (-sum (np.multiply(}\Psi\text{Y)))/N - }|W|^2/2}$
Here we would try to minmize this objective.

Now, let's try to find out gradient
Dropping negative. Gradient of the second term (regularizer) wrt W would be simply W.
The first term is (also dropping /N): sum (np.multiply(ΨY))

Let's analyze it. It is sum over N terms (inputs). for one input x, we'll get loss as:
loss = np.dot(ψ, y)
However, for calculating i,j element of the gradient matrix, we would need to differentiate loss over all terms wrt W_{ij}
For simplicity let's use J to represent the loss, without regularization
J = Σ_i^N np.dot(Ψ_i, Y_i). See that we can write element wise multiplication of two matrices like this if we are adding all the elements after that.
Let, y(i) = class for the i^th input. Thus y(i) $\in \{0, 1, ..C-1\}$
Using y(i) we can represent, J = $\Sigma_i^N \Psi_{iy(i)}$
Now, let us write the intermediate variables to come up to Ψ in terms of W and X
Let, ξ = element wise exponentiation of (XW). It is a matrix with dimension N$\times$C
Let, ϵ = sum of rows of ϵ. It is a vector with length N. $\epsilon_i = \Sigma_j^C \xi_{ij}$
Thus Ψ =log(ξ/ϵ). Here each row of Ψ is getting divided by corresponding element of ϵ
J = $\Sigma_i^N \Psi_{iy(i)}$ = Σ_i^N (log($\xi_{iy(i)}/\epsilon_i$)) = Σ_i^N (log($\xi_{iy(i)}$) - log(ϵ_i))
i.e: $\boxed{\text{J} = (\Sigma_i^N\ (XW)_{iy(i)}) - (\Sigma_i^N\ \log(\epsilon_i)\ ) = \text{I}^{st}\text{ term - II}^{nd}\text{ term} = \text{J}_1 - \text{J}_2}$

Calculating j1' = d(J$_1$)/d(W$_{ij}$):
Changing subscript, J$_1$ = (Σ_k^N $(XW)_{ky(k)}$)
i.e: j1' = Σ_k^N d$(XW)_{k,y(k)}$/d(W$_{i,j}$)
For a particular k (i.e., a paricular input), $(XW)_{k,y(k)}$ = np.dot(X$_k$, W$_{y(k)}$)
While differentiating it wrt W$_{i,j}$, only ith component of the vector would survive, and that's too if j == y(k)
Thus the whole terms for all the inputs would be the sum of ith component of the input vector provided that input belongs to class j
Thus whole $\boxed{\text{J}_1\text{' matrix = np.dot(Y.T, X)}}$
Please note, each row of Y.T has 1, only when the corresponding column belongs to that class. We may need to transpose it.

Now calculating j2' = d(J$_2$)/d(W$_{ij}$):
Changing subscript, J$_2$ = Σ_k^N log(ϵ_k) = Σ_k^N log(Σ_l^C ξ_{kl}) = Σ_k^N log(Σ_l^C exp((XW)$_{kl}$))
j2' = Σ_k^N (1/(Σ_l^C exp((XW)$_{kl}$))) * Σ_l^C (exp((XW)$_{kl}$) d((XW)$_{kl}$)/d(W$_{ij}$))
Shortening, j2' = Σ_k^N (1/ϵ_k) * Σ_l^C (ξ_{kl} d((XW)$_{kl}$)/d(W$_{ij}$))
Now, let's try to analyze the inner summation. It would be non-zero only when j == l, otherwise W term won't be there in it. So dropping the summation and replacing l by j we get:
j2' = Σ_k^N (1/ϵ_k) * ξ_{kj} d((XW)$_{kj}$)/d(W$_{ij}$)
XW$_k j$ = np.dot(X$_{k,.}$, W$_{.,j}$)
d((XW)$_{kj}$)/d(W$_{ij}$) = d(np.dot(X$_{k,.}$, W$_{.,j}$))/d(W$_{ij}$) = X$_{ki}$
Thus j2' = Σ_k^N (1/ϵ_k) * ξ_{kj} * X$_{ki}$, Here * represents element wise or row wise multiplication
Let Φ_{kj} = ξ_{kj}/ϵ_k. Thus Φ is simply softmax matrix.
Thus, j2' = Σ_k^N Φ_{kj} X$_{ki}$
From this now, we can write the derivate for complete matrix as: $\boxed{\text{J}_2\text{' = np.dot(X.T, }\Phi\text{)}}$

Using the above two, and introducing the terms we have omitted (-ive and /N), and the

regularizer term, we get:
J' = (-np.dot(X.T, Y) + np.dot(X.T, Φ))/N - λW
$\boxed{\text{J' = np.dot(X.T, } \Phi\text{-Y)/N - } \lambda \text{W}}$
We have introduced λ to trade off with regularization.

(b) Two

Introduction

Dummy Section

Bibliography