

Convolutional neural networks for text classification

September 6, 2018; due September 20, 2018 (11:59pm)

In this homework, you will be implementing the forward pass, backpropagation, and gradient checking for a convolutional neural network with sparse inputs for text classification. The python notebook to be completed and the training data, can be found on bCourses in `Files/HW2/`. Turn in your homework on gradescope to be evaluated.

1 Preliminaries

(Simply state the derivative; no need to derive it.)

1. $\frac{dcx}{dx} =$
2. $\frac{d \log(x)}{dx} =$
3. $\frac{d\sigma(x)}{dx} =$
4. $\frac{d \tanh(x)}{dx} =$
5. $\frac{d(2x)^2}{dx} =$

2 Forward

Let S = vocab size, k = window width, F = number of filters; w = one-hot vector $\in \mathbb{R}^S$ (all zeros except a single 1 identifying its position in the vocabulary). A text of N words is the sequence of such vectors: $[w_0, \dots, w_N]$. Using the notation from Goldberg 2017 (p. 151), define $x_i = \oplus(w_{i:i+k-1})$ to be the concatenation of vectors from position i to position $i+k-1$; $x_i \in \mathbb{R}^{Sk}$. A convolutional *filter* $u_f \in \mathbb{R}^{Sk}$ is applied in a sliding window to each position i in the sentence, followed by a tanh nonlinearity, yielding:

$$p_{i,f} = \tanh(x_i^\top u_f) \quad (1)$$

We will build a CNN with *narrow convolution* (Goldberg 154); for a single filter, $p_f = [p_{1,f}, \dots, p_{N-k+1,f}] \in \mathbb{R}^{N-k+1}$. For max-pooling, we then take the max element in p_f across all of those $N-k+1$ positions; ($h_f = \max_i p_f \in \mathbb{R}$). $h = [h_1, \dots, h_F] \in \mathbb{R}^F$ is the vector of such values across all F filters. The output then is $o = \sigma(h^\top V)$.

The model is comprised of two sets of parameters:

$$U = [u_1, \dots, u_F] \quad (2)$$

$$V \in \mathbb{R}^F \quad (3)$$

For simplicity, we omit all bias terms here.

Given the parameters and the definition of the model, complete the `forward` function in the python notebook to calculate o (the probability of the positive class) for an input text. You may not import any additional libraries or modify the utility functions but you are free to write additional functions if you need.

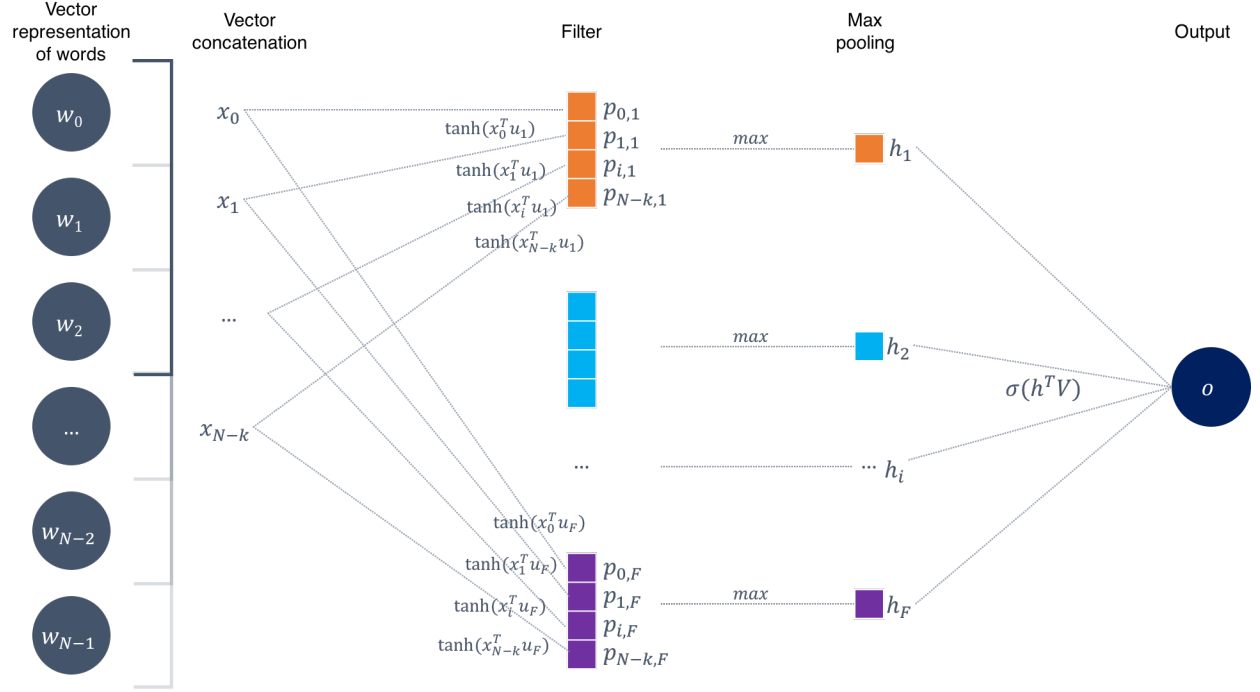


Figure 1: Variables and their relations

3 Backward

Given training data in the form of $\langle x, y \rangle$ pairs, we can use backpropagation to learn the values of U and V by taking the derivative of a loss function with respect to those parameters, and using stochastic gradient descent to minimize that loss. We'll use the negative log likelihood as our loss function (alternatively, we seek the maximize the log likelihood). For a single training example, the log likelihood is the following:

$$L(U, V) = \underbrace{y \log o}_A + \underbrace{(1 - y) \log(1 - o)}_B \quad (4)$$

Your task is to calculate the gradient update for U . To make it easier, here's the process for calculating the gradient update for V : since the loss function is the outcome of a sequence of function applications, we can use the chain rule to simplify the process of finding the derivative.

$$\frac{\partial L(U, V)}{\partial V} = \frac{\partial(A + B)}{\partial V} = \frac{\partial A}{\partial V} + \frac{\partial B}{\partial V} \quad (5)$$

A

$$\frac{\partial A}{\partial V} = \frac{\partial y \log(\sigma(Vh))}{\partial \sigma(Vh)} \times \frac{\partial \sigma(Vh)}{\partial Vh} \times \frac{\partial Vh}{\partial V} \quad (6)$$

$$= \frac{y}{\sigma(Vh)} \times \sigma(Vh) (1 - \sigma(Vh)) \times h \quad (7)$$

$$= y (1 - \sigma(Vh)) h \quad (8)$$

B

$$\frac{\partial B}{\partial V} = \frac{\partial(1-y) \log(1-\sigma(Vh))}{\partial(1-\sigma(Vh))} \times \frac{\partial(1-\sigma(Vh))}{\partial Vh} \times \frac{\partial Vh}{\partial V} \quad (9)$$

$$= \frac{1-y}{1-\sigma(Vh)} \times -\sigma(Vh)(1-\sigma(Vh)) \times h \quad (10)$$

$$= -(1-y)(\sigma(Vh))h \quad (11)$$

$$\frac{\partial(A+B)}{\partial V} = \frac{\partial A}{\partial V} + \frac{\partial B}{\partial V} \quad (12)$$

$$= y(1-\sigma(Vh))h - (1-y)(\sigma(Vh))h \quad (13)$$

$$= (y - \sigma(Vh))h \quad (14)$$

The gradient update for V is $(y - \sigma(Vh))h$.

3.1 Gradient

What is the gradient update for U ?

3.2 Implementation

Implement both updates (for V and U) by completing `backward` function in the python notebook. Do not import any additional libraries or modify the utility functions. The learning rate, convolution window size, number of filters, random seed, and training regime are all defined and should not be modified.

4 Gradient checking

It's easy to get the gradient wrong; one way of checking its correctness is to use the definition the derivative. For any function $J(\theta)$ parameterized by a single parameter θ , the derivative is equal to:

$$\frac{\partial}{\partial \theta} J(\theta) = \lim_{\epsilon \rightarrow 0} \frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{2\epsilon} \quad (15)$$

Let $\epsilon = 10^{-4}$; if we evaluate $\frac{J(\theta+\epsilon)-J(\theta-\epsilon)}{2\epsilon}$ for each parameter in turn, we get a numerical approximation to the gradient. How close is this numerical gradient to the analytical version you derived in section 3 above? Implement gradient checking in the python notebook for V and U ; for each parameter in V and U , calculate the squared difference between the numerical gradient and the analytical gradient; what is the sum of the squared differences for V and U ? Report two numbers: the sum of squared differences for V and the same for U .

For more information on gradient checking in the context of neural networks, see: http://ufldl.stanford.edu/wiki/index.php/Gradient_checking_and_advanced_optimization.

Deliverables

- Answers to §1, §3.1 and §4 as a PDF.
- Completed python notebook including functions for:
 - forward
 - backward
 - calc_numerical_gradients_V
 - calc_numerical_gradients_U
 - check_gradient
- PDF of the completed notebook with each cell executed.