Convolutional neural networks for text classification

September 8, 2017; due September 21, 2017 (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 skeleton of a script, along with training data, can be found on bCourses in `Files/HW2/`

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 V = vocab size, k = window width, F = num filters; w = one-hot vector $\in \mathbb{R}^V$ (all zeros except a single 1 identifying its position in the vocabulary). A text of N words is the sequence of such vectors: $[w_0, \ldots, w_N]$. 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 $i + k - 1$; $x_i \in \mathbb{R}^{Vk}$. A convolutional *filter* $u_f \in \mathbb{R}^{Vk}$ is applied in a sliding window to each position i in the sentence, followed by a $\tanh$ nonlinearity, yielding:

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

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

This model is therefore comprised of two sets of parameters:

$$U = [u_1, \ldots, u_F] \tag{2}$$

$$V \in \mathbb{R}^F \tag{3}$$

For simplicity, we omit all bias terms here.

Given those parameters and the definition of this model, write the `forward` function in `hw2_cnn.py` to calculate o (the probability of the positive class) for an input text. Do not change anything in the script outside of this function (including importing additional libraries).

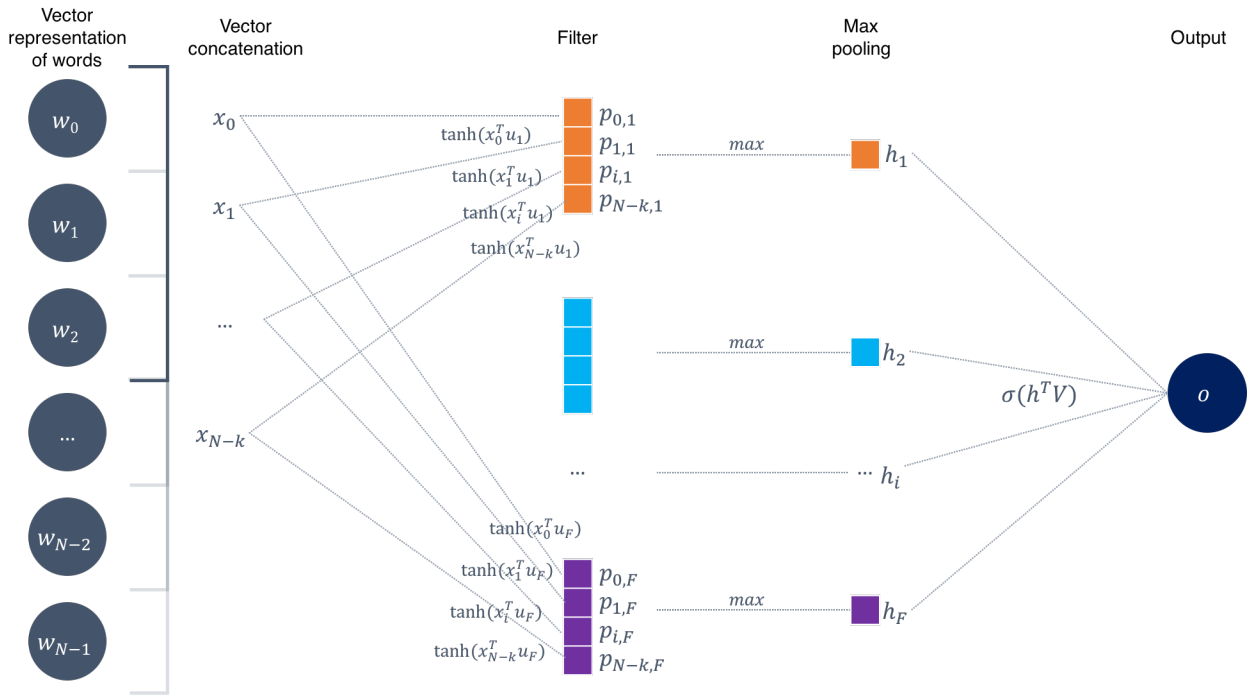

Figure 1: Variables and their relations

3 Backward

Given training data in the form of $< x, y >$ 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 move in the direction of that gradient to minimize that loss. We'll use the negative log likelihood as our loss (alternatively, we seek the maximize the log likelihood). For a single training example, the log likelihood is the following:

$$L(U, V) = \underbrace{y \log o}_{A} + \underbrace{(1 - y) \log(1 - o)}_{B} \tag{4}$$

Your task for this part 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.

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

A

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

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

$$= y\left(1 - \sigma\left(Vh\right)\right) h \tag{8}$$

B

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

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

$$= -(1-y)\left(\sigma\left(Vh\right)\right)h \tag{11}$$

$$\frac{\partial A+B}{V} = \frac{\partial A}{V} + \frac{\partial B}{V} \tag{12}$$

$$= y\left(1-\sigma\left(Vh\right)\right)h - (1-y)\left(\sigma\left(Vh\right)\right)h \tag{13}$$

$$= \left(y-\sigma\left(Vh\right)\right)h \tag{14}$$

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

3.1 Gradient

What is the gradient update for U?

3.2 Implementation

Implement both updates (for V and U) in the `backward` function of `hw2_cnn.py`. Again, do not change anything in the script outside of this function (including importing additional libraries). The learning rate, convolution window size, number of filters, random seed, and training regime are all defined (and should not be modified) there.

4 Gradient checking

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

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

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 `hw2_cnn.py` 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 here: the sum of squared differences for V and the sum of squared differences 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.
- Your modified script of `hw2_cnn.py` including functions for:
  - `forward`
  - `backward`
  - `calc_numerical_gradients_V`
  - `calc_numerical_gradients_U`
  - `check_gradient`