ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Prof. Ryan Cotterell

Course Assignment Episode 1

08/12/2020 - 17:56h

Question 1: Backpropagation (15 pts)

(a) In the following question, we consider the function $f(v, x, y, z) = z \cdot \log((x - y)^2 + \exp(v)) - \exp(v) \cdot x^{-z}$ where log is the natural logarithm.

(i) Draw the computation graph for f.

(ii) Evaluate your computation graph from (i) at the point (1,2,3,4), filling in the value of all intermediate nodes in your drawing.

(iii) Now apply backpropagation to your computation graph from (i) at the point (1,2,3,4), making use of the values you found in (ii) after applying forward propagation. Again, fill in all intermediate values in your drawing. (To make the graders' lives easier, it may make sense to give two drawings of the graphs to keep it from getting too crowded.) Show that your computation is correct by computing $[\frac{\partial f}{\partial v}, \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}]$ with symbolic differentiation[1] and comparing it to the output of backpropagation.

(b) Backpropagation on a computation graph can be used to efficiently obtain the derivatives of a function with respect to the input. In this problem, we consider a differentiable map $f : \mathbb{R}^n \to \mathbb{R}$. This question makes heavy use of the operator ∇ (pronounced nabla). We write $\nabla f(\mathbf{x})$ to denote the **Jacobian** of f evaluated at point $\mathbf{x}$. Importantly, in the special case that the codomain of f is simply $\mathbb{R}$, we refer to the Jacobian of f as the **gradient** of f. When ∇f gives us a gradient, we will take it to be a row vector. Furthermore, we write $\nabla\nabla f(\mathbf{x})$ or $\nabla^2 f(\mathbf{x})$ to indicate the **Hessian** of f. Recall from Lecture 2 that $f(\mathbf{x})$'s Hessian is the matrix of second derivatives at point $\mathbf{x}$. This notation is intuitive because the Hessian of f is simply the Jacobian of the gradient of f.

You will now derive an algorithm to compute the Hessian $\nabla^2 f(\mathbf{x})$ using repeated calls to backpropagation.

(i) First, show the following identity:

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \nabla(\mathbf{e}_1 \nabla f(\mathbf{x})^\top) \\ \vdots \\ \nabla(\mathbf{e}_n \nabla f(\mathbf{x})^\top) \end{bmatrix} \tag{1}$$

where $\{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$ are the standard basis vectors.

[1] By symbolic differentiation, we mean use partial differentiation as you learned it in calculus class.

(ii) Use the identity in (i), to give an efficient algorithm for computing $\nabla^2 f(\mathbf{x})$. Analyze the runtime of your algorithm. Take m to be the number of edges in the function's computation graph. Recall from class that this implies that the computation of f runs in $\mathcal{O}(m)$ time.

(iii) Now, suppose we wish to compute the tensor of k^{th}-order derivatives. Give an algorithm to produce this tensor and analyze its runtime in terms of m. To make this question not dependent on tedious tensor notation, rigorously give an algorithm to produce the 3-tensor of third-order derivatives and analyze its runtime. Then, give a non-rigorous generalization to the k^{th}-order derivatives.

(iv) **Bonus:** Give an $\mathcal{O}(m)$ algorithm for computing the *diagonal* of $\nabla^2 f(\mathbf{x})$.

(c) Now, we will analyze the finite-difference procedure (also known as numerical differentiation) to calculate the derivative of the loss with respect to each parameter in a model. Recall from Lecture 2 that the finite-difference approximation can give us an estimate of the derivative of a function with respect to a single variable:

$$\frac{\partial f(\mathbf{x})}{\partial x_i} \approx \frac{f(\mathbf{x} + h \cdot \mathbf{e}_i) - f(\mathbf{x})}{h} \tag{2}$$

where $h > 0$ is taken to be small and, again, we take $\{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$ to be the standard basis vectors. The approximation becomes exact as $h \to 0$; eq. 2 is simply the definition of the derivative after all! We now consider a multi-layer perceptron with 3 layers (using sigmoid activations; the number of layers includes the input and output layer) for a c-class classification problem where our input $\mathbf{x} \in \mathbb{R}^n$ and the intermediate layer has q hidden units.

(i) What would the runtime of this approach be for a single update of all parameters in the model in terms of n, c, q? Your analysis should take into account the time it takes to compute f.

(ii) Consider an even simpler model: a binary softmax classifier (see lecture 3), again with $\mathbf{x} \in \mathbb{R}^n$ and with weight vector $\mathbf{w} \in \mathbb{R}^{\mathbf{n}}$. Assume the machine we are using gives us precision (relative error) on the order of $\mathcal{O}(10^{-16})$ for additive, multiplicative and division operations, $\mathcal{O}(10^{-12})$ for the sigmoid operation, and $\mathcal{O}(10^{-10})$ for subtraction operations. Give a bound for precision in the calculation of eq. 2 with respect to the weight vector $\mathbf{w}$ (that is, for the partial derivative estimate of a single weight w_i)? **Hints**: you may assume summing a set of numbers is performed under the binary tree paradigm to avoid dependence on sum order; you may adversarially choose the direction of error that leads to the highest possible relative error; you may use the relation $f((1 \pm \epsilon)x) \approx f(x) \pm \epsilon x f'(x)$.

(iii) If you solved part (ii) as above and not as written in the first version of the assignment: Give an informal description of how we might see error compound in our MLP specified earlier.

Question 2: Parameter Estimation for Log-Linear Models (15 pts)

In this problem, we consider multi-*label* classification, which is an extension of binary classification that is distinct from multi-class classification. The problem set-up is as follows: We have an input pre-encoded as $\mathbf{x} \in \mathbb{R}^n$ and a series of K classes. Define $\mathbb{B} = \{0, 1\}$. An element $\mathbf{y} \in \mathbb{B}^K$ is a binary vector of length K, i.e. $\mathbf{y} = [y_1, \ldots, y_K]$ where each

$y_k \in \mathbb{B} = \{0, 1\}$. Our goal is to build a probabilistic model $p(\mathbf{y} \mid \mathbf{x})$ where $\mathbf{y} \in \mathbb{B}^K$. We consider a simple model for multi-label classification that consists of K independent binary classification models. Specifically, let each of the classifiers for label y_k be a logistic model:

$$p(Y_k = 1 \mid \mathbf{x}; \boldsymbol{\theta}_k) = \sigma(\boldsymbol{\theta}_k^\top \mathbf{x}) = \frac{1}{1 + \mathrm{e}^{-\boldsymbol{\theta}_k^\top \mathbf{x}}} \tag{3}$$

where $\Theta \in \mathbb{R}^{n \times K}$ is a matrix of parameters and the column vector $\boldsymbol{\theta}_k \in \mathbb{R}^n$ are the parameters corresponding to the k^{th} class.

(a) Let $\mathcal{D} = \{(\mathbf{y}^{(j)}, \mathbf{x}^{(j)}\}_{j=1}^J$ be a training set of J items. Write down the log-likelihood of the training data $\mathcal{D}$. Discuss how you could compute the maximum-likelihood estimate Θ_{MLE}. (**Hint**: Use calculus to compute the gradient then explain how you could find the minimizer algorithmically.)

(b) Argue that the log-likelihood from (a) is continuous, concave, and differentiable at all points in $\mathbb{R}^{K \times n}$.

(c) Now suppose we would like to regularize our parameters Θ to prevent overfitting. We apply both the L_1 and L_2 regularization, i.e., we subtract $\lambda_1 \sum_{k=1}^K ||\boldsymbol{\theta}_k||_1$ and $\lambda_2 \sum_{k=1}^K ||\boldsymbol{\theta}_k||_2^2$ to the log-likelihood you found in (a) to create an augmented log-likelihood. Derive the gradient of the augmented log-likelihood.

(d) Assume $\lambda_1, \lambda_2 \neq 0$. Is the augmented log-likelihood from (c) still continuous, concave and differentiable for all $\Theta \in \mathbb{R}^{K \times n}$?

(e) Show that maximizing the augmented log-likelihood is a form of **maximum-a-posteriori** estimation. Specifically, express the problem as maximization of the log of the likelihood multiplied by a prior.

(f) Prove the prior found in (e) is a member of the exponential family.

Question 3: Sentiment Classification* (15 pts)

(a) Download the Pang and Lee movie review data.[2] Hold out a randomly-selected 400 reviews as a test set. Download a sentiment lexicon, such as the one currently available from Bing Liu.[3]

(i) Tokenize the data using a library of your choosing[4] and classify each document as positive if it has more positive sentiment words than negative sentiment words. Compute the accuracy and F-measure[5] on detecting positive reviews on the test set, using this lexicon-based classifier.

(ii) Then train a discriminative classifier (averaged perceptron or logistic regression) on the training set, and compute its accuracy and F-measure on the test set.

(iii) Qualitatively compare the difference in accuracy between the two methods. Does the discriminative classifier do overwhelmingly better than guessing the mode? Describe a method for quantifying the significance of the difference between the two methods.

[2] `http://www.cs.cornell.edu/people/pabo/movie-review-data/`

[3] `https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html`

[4] e.g., NLTK, Stanford CoreNLP, mosestokenizer

[5] We take F-measure (F_1) to be the harmonic mean of precision p and recall r, i.e., $F_1 = \frac{2}{\frac{1}{p} + \frac{1}{r}}$

(b) The next problem will require you to build a classifier and test its properties. Pick a multi-class text classification dataset that is not already tokenized. One example is a dataset of New York Times headlines and topics (Boydstun, 2013).[6] Divide your data into training (60%), development (20%), and test sets (20%), if no such division already exists. If your dataset is very large, you may want to focus on a few thousand instances at first.

(i) Compare various vocabulary sizes of $10^2, 10^3, 10^4, 10^5$, using the most frequent words in each case. Train log-linear models, using feature extraction methods discussed in Lecture 3, for each vocabulary size. Plot the accuracy and macro F-measure on the test set with the increasing vocabulary size.

(ii) Train the same set of log-linear models for each vocabulary size but this time, with an L_2 penalty on the models' weights. Tune the regularizer's coefficient to maximize accuracy on the development set. Again plot the accuracy and macro F-measure on the test set against the increasing vocabulary size.

(iii) Provide an analysis of the results from above. At what point do we see diminishing returns in evaluation metrics when increasing the vocabulary size? Does regularization appear to help some of the classifiers? If so, is there a clear relationship between vocabulary size and the difference in evaluation metrics for regularized vs. unregularized models?

Question 4: Unigram Language Models* (10 pts)

In this problem, we consider the estimation of unigram language distributions. If V is a vocabulary, a unigram language is a distribution over words in V out of context. A unigram has no context!

(a) Consider a unigram language model over a vocabulary of size $|V|$. Suppose that a word appears m times in a corpus with M tokens in total. Apply add-λ smoothing to the corpus. For what values of m is the smoothed probability greater than the unsmoothed probability?

(b) Consider a simple language in which each token is drawn from the vocabulary V with probability $\frac{1}{|V|}$, independent of all tokens.

(i) Given a corpus of size M, what is the expected value of the fraction of all possible bigrams with zero count? (You may assume BOS is in V so that the corpus has M bigrams.)

(ii) Determine the value of M such that the fraction of bigrams with zero count is at most $\varepsilon \in (0, 1)$.

[6] available as a CSV file at `http://www.amber-boydstun.com/supplementary-information-for-making-the-news.html`; Use the field *majortopic* for this problem.

* Problems adapted from Jacob Eisenstein's Introduction to Natural Language Processing textbook.