Tsinghua-Berkeley Shenzhen Institute
Learning From Data
Fall 2020

Programming Assignment 2

Issued: Friday 23rd October, 2020 **Due:** Friday 6th November, 2020

POLICIES

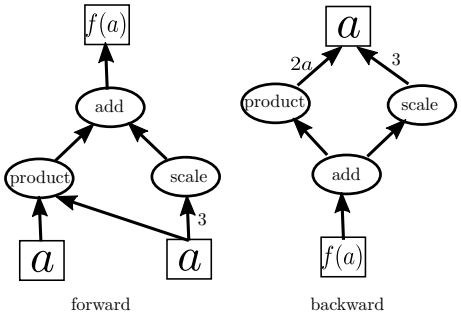
- **Acknowledgments:** We expect you to make an honest effort to solve the problems individually. As we sometimes reuse problem set questions from previous years, covered by papers and web pages, we expect the students **NOT** to copy, refer to, or look at the solutions in preparing their answers (relating to an unauthorized material is considered a violation of the honor principle). Similarly, we expect you not to google directly for answers (though you are free to google for knowledge about the topic). If you do happen to use other material, it must be acknowledged in `README.md`, with a citation on the submitted solution.
- **Required homework submission format:** You should submit the assignment by the invitation link `https://classroom.github.com/a/CLGZIvE-`. This link will create a private GitHub repository from the code template. Push your modification to the master branch and view the auto-grading results afterwards. The teaching assistant will grade your assignment mainly based on the rightness of your programming implementation. Optionally you can write your ideas in `README.md` or in code comments.
- **Collaborators:** If you collaborated with others, in `README.md`, list their names and GitHub ID and for which question(s).

1.1. (5 points) *Automatic differentiation* In class, we have learned to build a neural network, we need to implement the forward and backward propagation of given network model. This underlining mechanism is also called Automatic differentiation, autodiff for short. Following the convention of popular neural network library. We call an object with the ability of autodiff `tensor`. To explain autodiff operation of tensor, we take a single variable function $f(a) = a^2 + 3a$ for an example. Then the tensor object `a,p,s,f_a` are defined as follows:

```
a = tensor()
p = product(a, a)
s = scale(a, 3)
f_a = add(p, s)

```

The forward propagation evaluates the function value f for given a and can be decomposed into the following computational graph. To implement this forward pass, we only need to implement `add`, `product`, `scale` operators and use the following pseudo Python code to evaluate $f(a)$ at $a = 1.2$:




```
f_a.eval(1.2) = s.eval(1.2) + p.eval(1.2)
s.eval(1.2) = 3 * a.eval(1.2)
p.eval(1.2) = a.eval(1.2) * a.eval(1.2)
```

The backward pass of $f(a)$ computes the derivative of f about a, also illustrated in the above computational graph.

```
a.back(f_a) = p.diff(a) * p.back(f_a) + s.diff(a) * s.back(f_a)
p.back(f_a) = f_a.diff(p) * f_a.back(f_a)
s.back(f_a) = f_a.diff(s) * f_a.back(f_a)
```

where `x.back(x)` is defined as identity as a convention since there is no back operation starting from self. The `diff` method of each tensor object computes the derivative about the adjacent variable. For example, `p.diff(a) = 2 * a`, `s.diff(a) = 3` etc.

For multiple variable functions, the idea is the same as above but caution is needed for dimension coherence in matrix multiplication.

Please implement Automatic differentiation for rank 2 tensor (matrix) based on coding template within directory `lfdnn`. To be more specific, you need to implement the atomic tensor object `product`, `scale`, `sigmoid` and some derived tensor object. We call a tensor object atomic if the method `eval` and `diff` should be implemented, while a derived tensor can be treated as composition of atomic tensor object.

Hint: For rank 0 tensor (scalar) or rank 1 tensor (vector), we can treat them as special rank 2 tensor.

1.2. (5 points) *Multilayer perceptron* You have learned Ridge Regression and softmax regression, which can be treated as special kind of multilayer perceptron. There are no hidden layers in these regression models and their representational power is limited. To represent more complex input-output relationship, we need to introduce hidden layers. For multi-class classification problems, suppose there are k hidden layers, then the mathematical form of multilayer perceptron is as follows:

$$\begin{aligned} h_0 &= x \\ h_i &= \sigma(w_i h_{i-1} + b_i), i = 1, \ldots, k \\ \hat{y} &= \text{softmax}(w h_k + b) \end{aligned}$$

We use the averaged cross entropy to define the loss function, which can also be found

in WA1.2.

$$\ell(\mathbf{\Theta}) \stackrel{\text{def}}{=} \sum_{i=1}^{m} \log p(y^{(i)}|\boldsymbol{x}^{(i)}; \mathbf{\Theta}) = \sum_{i=1}^{m} \sum_{l=1}^{k} \mathbf{1}\left\{y^{(i)} = l\right\} \log \frac{e^{\boldsymbol{\theta}_l^{\mathrm{T}} \boldsymbol{x}^{(i)}}}{\sum_{j=1}^{k} e^{\boldsymbol{\theta}_j^{\mathrm{T}} \boldsymbol{x}^{(i)}}}.$$

where σ is the activation function and $\hat{y}$ is the probability vector for each class We can use stochastic gradient descent to minimize $\ell(\mathbf{\Theta})$, which has the following update scheme for $\ell(\mathbf{\Theta})$:

$$\ell(\mathbf{\Theta})_{t+1} \leftarrow \ell(\mathbf{\Theta})_t - \alpha \nabla_{\mathbf{\Theta}} \ell(\mathbf{\Theta})_{\boldsymbol{\theta}_t}$$

where α is called the learning rate.

To compute the gradient of the loss function, we can use the autodiff library of the first question and all forward and backward propagation details are hidden in the autodiff library.

Please implement Multilayer perceptron classifier by completing the code in **model.py**. Besides, use your model to re-implement `logistic regression` in **logistic_regression.py**, which you have already implemented with IRLS in PA1.

1.3. (bonus 2.5 points) *Ridge Regression with autodiff* Please re-implement ridge regression by completing the code in **ridge_regression.py**, which you have already implemented with direct solution in PA1. You are required to use SGD with the autodiff library of question 1.

Notice:

2.1. Use matrix operations other than loops for efficiency. If the running time of Auto-Grading steps exceeds 5 minutes, you will get point deductions.

2.2. You are expected to only use `numpy` packages to implement the algorithms.

2.3. All questions assume that the data are not centered around zero. Therefore, you need to train the extra bias parameter in your code.