Deep Learning Teaching Kit

Lab 1

---

1 Backpropagation

1. Warmup: Logistic regression is a pretty popular technique in machine learning to classify data into two categories. This technique builds over linear regression by using the same linear model but this is followed by the sigmoid function which converts the output of the linear model to a value between 0 and 1. This value can then be interpreted as a probability. This is usually represented as:

   $$P(y = 1|x_{\text{in}}) = x_{\text{out}} = \sigma(x_{\text{in}}) = \frac{1}{1 + e^{-x_{\text{in}}}} \tag{1}$$

   where x_{in} as the name would suggest is the input scalar (which is also the output of linear model) and x_{out} is the output scalar.

   If the error backpropagated to x_{out} is $\frac{\partial E}{\partial x_{\text{out}}}$, write the expression for $\frac{\partial E}{\partial x_{\text{in}}}$ in terms of $\frac{\partial E}{\partial x_{\text{out}}}$.

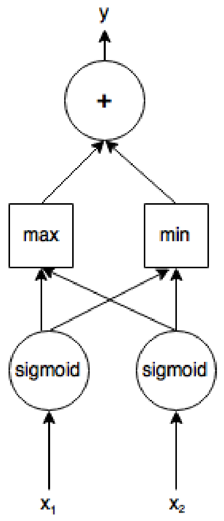
2. Multinomial logistic regression is a generalization of logistic regression into multiple classes. The softmax expression is at the crux of this technique. After receiving n unconstrained values, the softmax expression normalizes these values to n values that all sum to 1. This can then be perceived as probabilities attributed to the various classes by a classifier. Your task here is to backpropagate error through this module. The softmax expression which indicates the probability of the i-th class is as follows:

   $$P(y = i|X_{\text{in}}) = (X_{\text{out}})_i = \frac{e^{-\beta(X_{\text{in}})_i}}{\sum_k e^{-\beta(X_{\text{in}})_k}} \tag{2}$$

   What is the expression for $\frac{\partial(X_{\text{out}})_i}{\partial(X_{\text{in}})_j}$? (Hint: Answer differs when $i = j$ and $i \neq j$).

   The variables X_{in} and X_{out} aren't scalars but vectors. While X_{in} represents the n values input to the system, X_{out} represents the n probabilities output from the system. Therefore, the expression $(X_{\text{out}})_i$ represents the i-th element of X_{out}.

Figure 1: a DAG of modules



3. Figure 1 describes a simple network with the inputs being x_1 and x_2 (both scalars). Both inputs are passed through the sigmoid layer described above. Next, the outputs from the sigmoid layers are passed through max and min layers whose outputs can be represented as $o_{\min} = \min(i_1, i_2)$ and $o_{\max} = \max(i_1, i_2)$. The values output by the min and max layers are then summed in the final layer.

   Suppose the error passed down (backpropagated) to y is $\frac{\partial E}{\partial y}$. Write the values for $\frac{\partial E}{\partial x_1}$ and $\frac{\partial E}{\partial x_2}$ in terms of $\frac{\partial E}{\partial y}$.

2 MNIST Handwritten Digit Recognition (Torch)

For this problem, you will need to run and understand the `Lab1 sample_code` located at `https://bitbucket.org/junbo_jake_zhao/deeplearningkit/src/master/Lab1/sample_code/`. The sample code is based on Clement Farabet's tutorial on supervised learning: `https://github.com/clementfarabet/ipam-tutorials/tree/master/th_tutorials`. The difference is that we are using MNIST instead of SVHN. It is recommended that you walk through the tutorial in an itorch notebook.

Note that this lab is recommended to be run on a NVIDIA GPUs because CPUs would take a notoriously long time. In Torch, the conversion between CPU and GPU is easy:

```
-- Shift a model from host to device
model:cuda()
-- Shift the criterion from host to device
criterion:cuda()
-- Shift tensor from host to device
data = data:cuda()
-- Note, you need to assign the resulted tensor to a new tensor.
-- However this wouldn't be needed when shifting model or criterion.
```

You need to save the predictions and submit them to the Kaggle competition. Once you have understood the starter code, try to improve the model and beat the competition. After you finalize your model, create a script `result.lua`, that generates a file named `predictions.csv`. We will check that `predictions.csv` matches your final Kaggle submission. You will need to save your trained model as well to be able to generate the predictions via `result.lua`. The function `torch.save()` in the starter code does exactly this.

DO NOT submit your trained model file (the binary generated by torch during training) via email. These files are generally huge.

DO NOT train on the test data. Training on the test data is considered plagiarism. Note that the sample code does not use a validation set which we recommend you implement by yourselves.

Evaluation

Your grade for this section will be based on

- 30% - Kaggle performance. Full score as long as you beat the benchmark.
- 40% - Writeup on your model structure, training process, experiments, results, etc. We expect a rather formal report written with LaTeX.
- 30% - Simple, readable, commented code of final submitted script **`result.lua`** that is able to execute on the test data and generate a prediction file consistent with your final Kaggle submission.

Submission

Send your submission (writeup and `result.lua`) to your corresponding TA by the deadline. Merge your solutions to section 1 with the writeup from section 2. Include a link to the trained model file in the email. Please use the following title for your email.

[*CourseName* `YOUR_TEAM_NAME] Submission Lab1`

We will be sending out Kaggle invitations as soon as you form a team. Please compete as teams on Kaggle with exactly the same team names.