Deeplearing '18/'19: Assignment 1

Aron Hammond University of Amsterdam
Amsterdam
hammond756@live.nl
10437215

1 MLP backprop and NumPy implementation

1.1 Analytical derivation of gradients

Question 1.1 a)

$$\left(\frac{\partial L}{\partial x^{(N)}}\right)_i = -\frac{t_i}{x_i^{(N)}}$$

$$\left(\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}}\right)_{ij} = \frac{\exp(\tilde{x}_i^{(N)}) \cdot \sum_{k=1}^{d_N} \exp(\tilde{x}_k^{(N)}) - \exp(\tilde{x}_i^{(N)}) \cdot \exp(\tilde{x}_j^{(N)})}{\left(\sum_{k=1}^{d_N} \exp(\tilde{x}_k^{(N)})\right)^2}$$

$$= \text{softmax}(\tilde{x}_i^{(N)}) \cdot \left(\delta_{ij} - \text{softmax}(\tilde{x}_j^{(N)})\right) \qquad \text{where } \delta_{ij} = \begin{cases} 0, \text{if } i = j \\ 1, \text{if } i \neq j \end{cases}$$

$$\left(\frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}}\right)_i = \begin{cases} 0, & \text{if } \tilde{x}_i^{(l<N)} < 0 \\ \tilde{x}_i^{(l<N)}, & \text{if } \tilde{x}_i^{(l<N)} > 0 \end{cases}$$

$$\left(\frac{\partial \tilde{x}^{(l)}}{\partial x^{(l-1)}}\right)_{ij} = W_{ji}$$

$$\left(\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}}\right)_i = x_i^{(l-1)}$$

$$\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} = \mathbb{1}$$

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

Question 1.1 b)

$$\frac{\partial L}{\partial \tilde{x}^{(N)}} = \frac{\partial L}{\partial x^{(N)}} \begin{bmatrix} S_i(1-S_j) & \cdots & -S_i S_D \\ \vdots & \ddots & \\ -S_D S_j & & S_D(1-S_D) \end{bmatrix} \qquad \text{where } S_i = \text{softmax}(\tilde{x}^{(N)})_i$$

$$\frac{\partial L}{\partial x^{(l<N)}} = \frac{\partial L}{\partial \tilde{x}^{(l+1)}} \cdot \left(W^{(l+1)}\right)^T$$

$$\frac{\partial L}{\partial \tilde{x}^{(l<N)}} = \frac{\partial L}{\partial x^{(l)}} \cdot \theta(\tilde{x}^{(l)}) \qquad \text{where } \theta(x)_i = \begin{cases} 0, \text{if } x_i < 0 \\ 1, \text{if } x_i > 0 \end{cases}$$

$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial \tilde{x}^{(l)}} \cdot \left(x^{(l-1)}\right)^T$$

$$\frac{\partial L}{\partial b^{(l)}} = \frac{\partial L}{\partial \tilde{x}^{(l)}}$$

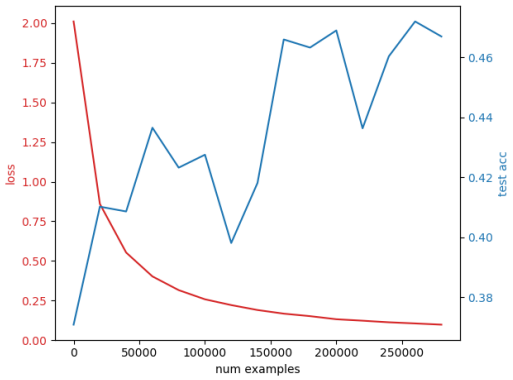
Question 1.1 c)

When $B \neq 1$ the equations will include a batch dimension. Vectors will become matrices and matrices become $3D$ matrices. Also, you have to make sure that the final gradient for the parameters (weight matrix and bias of linear module) have the correct shape. This can be done by either summing or averaging over the batch dimension.

1.2 NumPy implementation

Question 1.2

Figure 1: Training loss (red) and test accuracy (blue) on cifar10 with NumPy MLP and default settings



2 PyTorch MLP

To be structured with the experiments, I implemented a script that performs 5-fold cross validation. The following table shows the results of all 36 experiments. The motivation for the different parameter settings was as follows. For the batch_size, I wanted to test a range from small (64) to large (512). The same goes for the learning rate but then spaced on an exponential scale to find the right order of

magnitude. The architectures range from shallow and wide to deep and narrow. As you can see, the test scores vary a lot depending on the combination of settings. What I find notable is that none of these experiments obtain a test accuracy that is close to the default settings. This could be due to the fact that these models were trained using the `skorch` package that makes it possible to use `sklearn` modules with `pytorch` models.

Figure 2: Results of hyperparameter search

	mean_test_score	batch_size	learning_rate	n_hidden
0	0.11242	64	0.1	[100, 100]
1	0.11162	64	0.1	[1000]
2	0.11866	64	0.1	[50, 30, 20]
3	0.37482	64	0.01	[100, 100]
4	0.25468	64	0.01	[1000]
5	0.36146	64	0.01	[50, 30, 20]
6	0.37192	64	0.001	[100, 100]
7	0.39170	64	0.001	[1000]
8	0.37270	64	0.001	[50, 30, 20]
9	0.15542	128	0.1	[100, 100]
10	0.14622	128	0.1	[1000]
11	0.15166	128	0.1	[50, 30, 20]
12	0.41258	128	0.01	[100, 100]
13	0.32458	128	0.01	[1000]
14	0.38232	128	0.01	[50, 30, 20]
15	0.37902	128	0.001	[100, 100]
16	0.35014	128	0.001	[1000]
17	0.33658	128	0.001	[50, 30, 20]
18	0.22294	256	0.1	[100, 100]
19	0.19582	256	0.1	[1000]
20	0.32544	256	0.1	[50, 30, 20]
21	0.41644	256	0.01	[100, 100]
22	0.35870	256	0.01	[1000]
23	0.38614	256	0.01	[50, 30, 20]
24	0.32568	256	0.001	[100, 100]
25	0.35258	256	0.001	[1000]
26	0.29142	256	0.001	[50, 30, 20]
27	0.32258	512	0.1	[100, 100]
28	0.23844	512	0.1	[1000]
29	0.38744	512	0.1	[50, 30, 20]
30	0.40430	512	0.01	[100, 100]
31	0.36790	512	0.01	[1000]
32	0.36450	512	0.01	[50, 30, 20]
33	0.31560	512	0.001	[100, 100]
34	0.31206	512	0.001	[1000]
35	0.26624	512	0.001	[50, 30, 20]

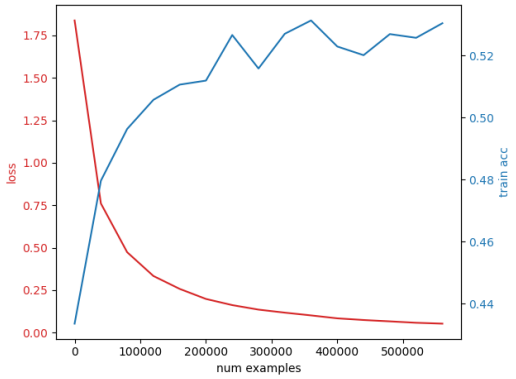
Since the gridsearch was not very helpfull in finding hyperparameters that would score at least 53% on the test set I eventually asked my classmates and managed to obtain 0.53 accuracy by using the Adam optimizer, a batch size of 400 and 4 layers of 500 hidden units.

3 Custom Module: Batch Normalization

3.1 Automatic differentiation

see `custom_batchnorm.py`

Figure 3: Training loss (red) and test accuracy (blue) on cifar10 with the PyTorch MLP (Adam, $lr = 0.0005$, $batch\_size = 400$, $n\_hidden = [500, 500, 500, 500]$



3.2 Manual implementation of backward pass

Question 3.1 a)

$$\left(\frac{\partial L}{\partial \gamma}\right)_j = \sum_s \sum_i \frac{\partial L}{\partial y_i^s} \hat{x}_i^s$$

$$\left(\frac{\partial L}{\partial \beta}\right)_j = \sum_s \sum_i \frac{\partial L}{\partial y_i^s}$$

Question 3.1 b)

see `custom_batchnorm.py`

Since I wasn't able to manually figure out the derivative w.r.t the input, the implementation is based on the following blog post (link)

Question 3.2

see `custom_batchnorm.py`

4 PyTorch CNN

Figure 4: Training loss (red) and test accuracy (blue) on cifar10 with the PyTorch ConvNet (Adam, $lr = 0.001$, $batch\_size = 128$