Assignment 1. MLPs, CNNs and Backpropagation

Max Bos
10669027
University of Amsterdam
maxn.bos@gmail.com

1 MLP backprop and NumPy implementation

1.1 Analytical derivation of gradients

Question 1.1 a)

$\frac{\partial L}{\partial x^{(N)}}$ yields a $1 \times d_N$ Jacobian matrix. We can compute each i-th cell of this matrix using $\left(\frac{\partial L}{\partial x^{(N)}}\right)_i = \left(\frac{\partial L}{\partial x_i^{(N)}}\right)$

$$\left(\frac{\partial L}{\partial x_i^{(N)}}\right) = \left(\frac{\partial - \log x^{(N)}_{\arg\max(t)}}{\partial x_i^{(N)}}\right) = \begin{cases} \frac{-1}{x_i^{(N)}}, & \text{if } i = \arg\max(t) \\ 0, & \text{otherwise} \end{cases}$$

$\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}}$ yields a $d_N \times d_N$ Jacobian matrix. We can compute each i-th row as $\frac{\partial x_i^{(N)}}{\tilde{x}^{(N)}}$, and each j-th cell of an i-th row as $\left(\frac{\partial x^{(N)}}{\partial \tilde{x}(N)}\right)_{ij} = \left(\frac{\partial x_i^{(N)}}{\partial \tilde{x}_j^{(N)}}\right)$.

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

$$= \begin{cases} \frac{\exp\left(\tilde{x}_i^{(N)}\right)\left(\sum_{k=1}^{d_N}\exp\left(\tilde{x}^{k\,(N)}\right)\right) - \exp\left(\tilde{x}_i^{(N)}\right)\exp\left(\tilde{x}_j^{(N)}\right)}{\left(\sum_{k=1}^{d_N}\exp\left(\tilde{x}^{k\,(N)}\right)\right)^2}, & \text{if } i = j \\ \frac{-\exp\left(\tilde{x}_i^{(N)}\right)\exp\left(\tilde{x}_j^{(N)}\right)}{\left(\sum_{k=1}^{d_N}\exp\left(\tilde{x}^{k\,(N)}\right)\right)^2}, & \text{otherwise} \end{cases}$$

We can now rewrite the above result in Kronecker delta notation:

$$\left(\frac{\partial x^{(N)}}{\partial \tilde{x}(N)}\right)_{ij} = \begin{cases} \frac{\exp\left(\tilde{x}_i^{(N)}\right)}{\sum_{k=1}^{d_N}\exp\left(\tilde{x}^{k\,(N)}\right)} \frac{\left(\sum_{k=1}^{d_N}\exp\left(\tilde{x}^{k\,(N)}\right)\right) - \exp\left(\tilde{x}_j^{(N)}\right)}{\sum_{k=1}^{d_N}\exp\left(\tilde{x}^{k\,(N)}\right)}, & \text{if } i = j \\ -\frac{\exp\left(\tilde{x}_i^{(N)}\right)}{\sum_{k=1}^{d_N}\exp\left(\tilde{x}^{k\,(N)}\right)} \frac{\exp\left(\tilde{x}_j^{(N)}\right)}{\sum_{k=1}^{d_N}\exp\left(\tilde{x}^{k\,(N)}\right)}, & \text{otherwise} \end{cases}$$

$$= \begin{cases} \frac{\exp\left(\tilde{x}_i^{(N)}\right)}{\sum_{k=1}^{d_N}\exp\left(\tilde{x}^{k\,(N)}\right)} \left(1 - \frac{\exp\left(\tilde{x}_j^{(N)}\right)}{\sum_{k=1}^{d_N}\exp\left(\tilde{x}^{k\,(N)}\right)}\right), & \text{if } i = j \\ -\frac{\exp\left(\tilde{x}_i^{(N)}\right)}{\sum_{k=1}^{d_N}\exp\left(\tilde{x}^{k\,(N)}\right)} \frac{\exp\left(\tilde{x}_j^{(N)}\right)}{\sum_{k=1}^{d_N}\exp\left(\tilde{x}^{k\,(N)}\right)}, & \text{otherwise} \end{cases}$$

$$= \frac{\exp\left(\tilde{x}_i^{(N)}\right)}{\sum_{k=1}^{d_N}\exp\left(\tilde{x}^{k\,(N)}\right)} \left(\delta_{ij} - \frac{\exp\left(\tilde{x}_j^{(N)}\right)}{\sum_{k=1}^{d_N}\exp\left(\tilde{x}^{k\,(N)}\right)}\right)$$

$\frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}}$ yields a $d_{l<N} \times d_{l<N}$ Jacobian matrix. We can compute each cell of this matrix as $\left(\frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}}\right)_{ij} = \left(\frac{\partial x_i^{(l<N)}}{\partial \tilde{x}_j^{(l<N)}}\right)$.

$$\left(\frac{\partial x_i^{(l<N)}}{\partial \tilde{x}_j^{(l<N)}}\right) = \left(\frac{\partial \max\left(0, \tilde{x}_i^{(l<N)}\right)}{\partial \tilde{x}_j^{(l<N)}}\right) = \begin{cases} 1, & \text{if } i = j \text{ and } \tilde{x}_i^{(l<N)} > 0 \\ 0, & \text{otherwise} \end{cases}$$

$\frac{\partial \tilde{x}^{(l)}}{\partial x^{(l-1)}}$ yields a $d_l \times d_{l-1}$ Jacobian matrix.

$$\frac{\partial \tilde{x}^{(l)}}{\partial x^{(l-1)}} = \frac{\partial W^{(l)} x^{(l-1)} + b^{(l)}}{\partial x^{(l-1)}} = W^{(l)}$$

$\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}} = \frac{\partial W^{(l)} x^{(l-1)} + b^{(l)}}{\partial W^{(l)}}$ yields a $d_l \times d_l \times d_{l-1}$ Jacobian matrix. We can compute each cell of this matrix as $\left(\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}}\right)_{ijk} = \left(\frac{\partial \tilde{x}_i^{(l)}}{\partial W_{jk}^{(l)}}\right)$.

$$\left(\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}}\right)_{ijk} = \left(\frac{\partial \tilde{x}_i^{(l)}}{\partial W_{jk}^{(l)}}\right) = \left(\frac{\partial \sum_{m=1}^{d_{l-1}} W_{im}^{(l)} x_m^{(l-1)} + b_i^{(l)}}{\partial W_{jk}^{(l)}}\right) = \begin{cases} x_m^{(l-1)}, & \text{if } i = j \text{ and } m = k \\ 0, & \text{otherwise} \end{cases}$$

$$\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} = \frac{\partial W^{(l)} x^{(l-1)} + b^{(l)}}{\partial b^{(l)}} = \vec{1}$$

Question 1.1 b)

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

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

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

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

$$\frac{\partial L}{\partial b^{(l)}} = \frac{\partial L}{\partial \tilde{x}^{(t)}} \frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} = \frac{\partial L}{\partial \tilde{x}^{(t)}} \vec{1}$$

Question 1.1 c)

The backpropagation equations derived above would not change, since in the case of a batchsize $B \neq 1$, the loss that gets backpropagated through the network is still a single value when you use the mean value of the individual samples' losses.

1.2 NumPy implementation

All necessary code for the NumPy MLP implementation has been implemented into **train_mlp_numpy.py**, **modules.py** and **mlp_numpy.py**. Figures 1 and 2 depict the loss and accuracy curves for the train and test sets, respectively. The desired accuracy of around 0.46 is achieved by the current implementation.

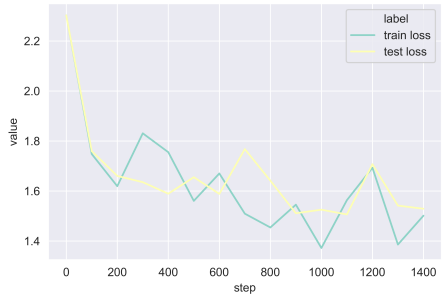

Figure 1: MLP NumPy train and test cross entropy loss over time

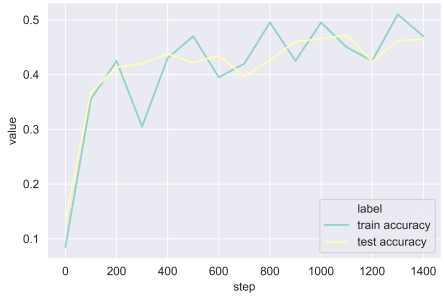

Figure 2: MLP NumPy train and test accuracies over time

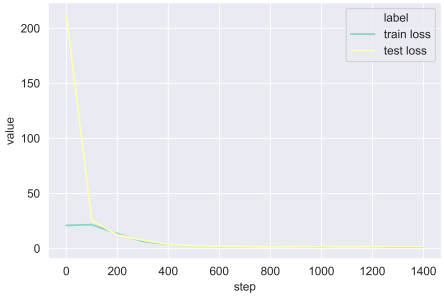

Figure 3: PyTorch MLP loss curve using the default settings

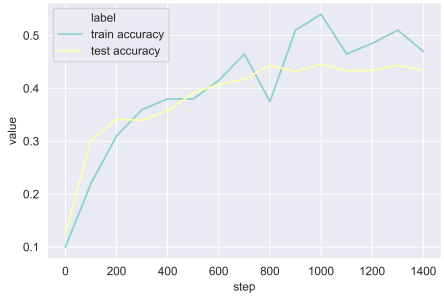

Figure 4: PyTorch MLP accuracy curve using the default settings

2 PyTorch MLP

The PyTorch MLP is implemented into **mlp_pytorch.py** and **train_mlp_pytorch.py**. Figures 3 and 4 depict the loss and accuracy curves using the default parameters for the PyTorch MLP for the train and test set, respectively.

Tests have been performed using one hidden layer of size 200, for which Figures 5 and 6 show the results. Figures 7 and 8 show the results when setting a learning rate of $2e-4$. The best results, an accuracy of around 0.52, were achieved when using three hidden layer of sizes '200,200,100', a learning rate of $2e-4$ and a maximum number of steps of 3000, see Figures **??** and **??**.

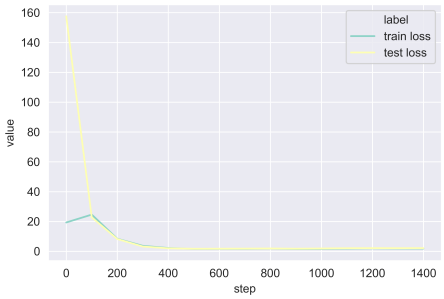

Figure 5: PyTorch MLP loss curve using a hidden layer of size 200

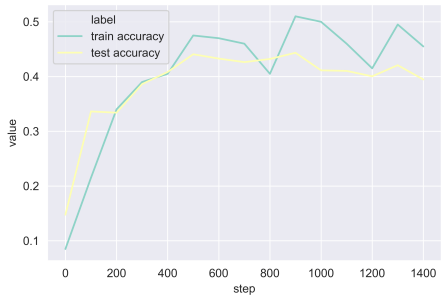

Figure 6: PyTorch MLP accuracy curve using a hidden layer of size 200

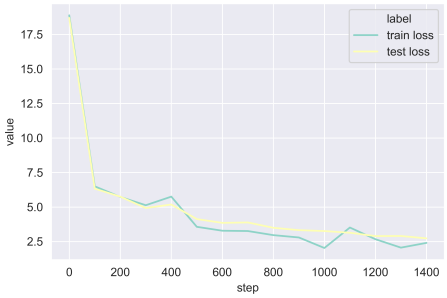

Figure 7: PyTorch MLP loss curve using a learning rate of 2e-4

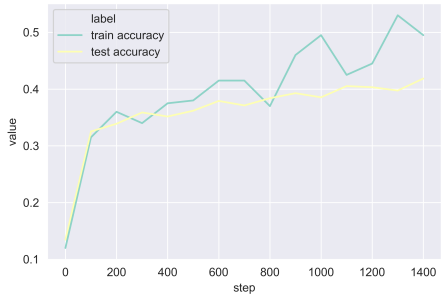

Figure 8: PyTorch MLP accuracy curve using a learning rate of 2e-4

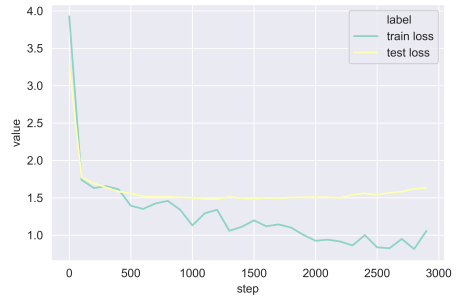

Figure 9: PyTorch MLP loss curve using one hidden layer of size 200, a learning rate of $2e-4$ and a maximum number of steps of 3000

Figure 10: PyTorch MLP accuracy curve using one hidden layer of size 200, a learning rate of 2e-4 and a maximum number of steps of 3000

3 Custom Module: Batch Normalization

Due to time constraints on my part I did not get to this part yet.

4 PyTorch CNN

The PyTorch CNN is implemented but could not be run in time due to time constraints on my part. Running on LISA resulted in a CUDA out of memory runtime error: **RuntimeError: CUDA out of memory. Tried to allocate 1.22 GiB (GPU 0; 10.92 GiB total capacity; 9.41 GiB already allocated; 989.50 MiB free; 13.49 MiB cached)**.