Deep Learning Course. Assignment 1

Diaconu Nichita 11737980

Abstract

In this assignment I experimented with different deep learning techniques, while at the same time implementing and deriving various low level methods for neural networks and their gradients. We were also encouraged to look into things that we thought were interesting, i.e. weight norm, the less renown counterpart to batch norm.

1 MLP backprop and NumPy implementation

1.1 Analytical derivation of gradients

Question 1.1 a)

Curly brackets represent $\{.\}$ the set of elements over the indices in the subscript $\{.\}_{ij}$.
$\odot$ represents elmentwise multiplication between two tensors.
$ReLU\_back(a)$ represents a function that outputs a tensor with the same shape as the input, which contains on each position, 1 if a is greater than 0 on that position and 0 otherwise.
$diag(a)$ turns a vector, a, into a matrix with a on the diagonal.

$$\frac{\partial L}{\partial x^{(N)}} = \left\{\left(\frac{\partial L}{\partial x^{(N)}}\right)_i\right\}_i = \left\{\left(\frac{\partial L}{\partial x_i^{(N)}}\right)\right\}_i = \left\{\frac{-t_i}{x_i}\right\}_i = -t \odot \frac{1}{x}$$

$$\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}} = \left\{\left(\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}}\right)_{ij}\right\}_{ij} = \left\{\left(\frac{\partial x_i^{(N)}}{\partial \tilde{x}_j^{(N)}}\right)\right\}_{ij} = \{-x_i^{(N)}x_j^{(N)} + \delta_{ij}x_i^{(N)}\}_{ij} = diag(x^{(N)}) - x^{(N)}(x^{(N)})^T$$

$$\frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}} = \left\{\left(\frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}}\right)_{ij}\right\}_{ij} = \left\{\left(\frac{\partial x_i^{(l<N)}}{\partial \tilde{x}_j^{(l<N)}}\right)\right\}_{ij} = \{ReLU\_back(0, \tilde{x}_i^{(l<N)})\delta_{ij}\}_{ij} = ReLU\_back(0, \tilde{x}^{(l<N)})$$

The last relation represents that we are only interested in the diagonal of the tensor where $\delta_{ij} = 1$

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

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

The last relation represents that we are only interested in the diagonal of the tensor where $\delta_{jk} = 1$

$$\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} = \left\{\left(\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}}\right)_{ij}\right\}_{ij} = \left\{\left(\frac{\partial \tilde{x}_i^{(l)}}{\partial b_j^{(l)}}\right)\right\}_{ij} = \{\delta_{ij}\}_{ij} = 1 \in \mathbb{R}^{d_l}$$

The last relation represents that we are only interested in the diagonal of the tensor where $\delta_{ij} = 1$

Question 1.1 b)

$$\frac{\partial L}{\partial \tilde{x}^{(N)}} = \frac{\partial L}{\partial x^{(N)}}\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}} = \left\{\sum_i \left(\frac{\partial L}{\partial x^{(N)}}\right)_i \left(\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}}\right)_{ij}\right\}_j = \left\{\sum_i \left(\frac{\partial L}{\partial x_i^{(N)}}\right)\left(\frac{\partial x_i^{(N)}}{\partial \tilde{x}_j^{(N)}}\right)\right\}_j =$$

$$= \left\{\sum_i \frac{-t_i}{x_i}(-x_i^{(N)}x_j^{(N)} + \delta_{ij}x_i^{(N)})\right\}_j = (-t \odot \frac{1}{x})(diag(x^{(N)}) - x^{(N)}(x^{(N)})^T)$$

$$\frac{\partial L}{\partial \tilde{x}^{(l<N)}} = \frac{\partial L}{\partial x^{(l<N)}}\frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}} = \left\{\sum_i \left(\frac{\partial L}{\partial x^{(l<N)}}\right)_i \left(\frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}}\right)_{ij}\right\}_j = \left\{\sum_i \left(\frac{\partial L}{\partial x_i^{(l<N)}}\right)\left(\frac{\partial x_i^{(l<N)}}{\partial \tilde{x}_j^{(l<N)}}\right)\right\}_j =$$

$$= \left\{\sum_i \frac{\partial L}{\partial x_i^{(l<N)}} ReLU\_back(0, \tilde{x}_i^{(l<N)})\delta_{ij}\right\}_j = \frac{\partial L}{\partial x^{(l<N)}} \odot ReLU\_back(0, \tilde{x}^{(l<N)})$$

$$\frac{\partial L}{\partial x^{(l-1)}} = \frac{\partial L}{\partial \tilde{x}^{(l)}}\frac{\partial \tilde{x}^{(l)}}{\partial x^{(l-1)}} = \left\{\sum_i \left(\frac{\partial L}{\partial \tilde{x}^{(l)}}\right)_i \left(\frac{\partial \tilde{x}^{(l)}}{\partial x^{(l-1)}}\right)_{ij}\right\}_j = \left\{\sum_i \left(\frac{\partial L}{\partial \tilde{x}_i^{(l)}}\right)\left(\frac{\partial \tilde{x}_i^{(l)}}{\partial x_j^{(l-1)}}\right)\right\}_j =$$

$$= \left\{\sum_i \frac{\partial L}{\partial \tilde{x}_i^{(l)}} W_{ij}^{(l)}\right\}_j = \frac{\partial L}{\partial \tilde{x}^{(l)}} W^{(l)}$$

$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial \tilde{x}^{(l)}}\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}} = \left\{\sum_i \left(\frac{\partial L}{\partial \tilde{x}^{(l)}}\right)_i \left(\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}}\right)_{ijk}\right\}_{jk} = \left\{\sum_i \left(\frac{\partial L}{\partial \tilde{x}_i^{(l)}}\right)\left(\frac{\partial \tilde{x}_i^{(l)}}{\partial W_{jk}^{(l)}}\right)\right\}_{jk} =$$

$$= \left\{\sum_i \frac{\partial L}{\partial \tilde{x}_i^{(l)}} x_i^{(l-1)}\delta_{jk}\right\}_{jk} = \frac{\partial L}{\partial \tilde{x}^{(l)}}^T x^{(l-1)}$$

$$\frac{\partial L}{\partial b^{(l)}} = \frac{\partial L}{\partial \tilde{x}^{(l)}}\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} = \left\{\sum_i \left(\frac{\partial L}{\partial \tilde{x}^{(l)}}\right)_i \left(\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}}\right)_{ij}\right\}_j = \left\{\sum_i \left(\frac{\partial L}{\partial \tilde{x}_i^{(l)}}\right)\left(\frac{\partial \tilde{x}_i^{(l)}}{\partial b_j^{(l)}}\right)\right\}_j =$$

$$= \left\{\sum_i \frac{\partial L}{\partial \tilde{x}_i^{(l)}} \delta_{ij}\right\}_j = \frac{\partial L}{\partial \tilde{x}^{(l)}}$$

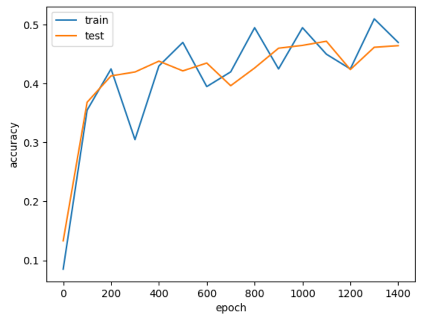
Question 1.1 c)

When batch size is different from 1, we should:

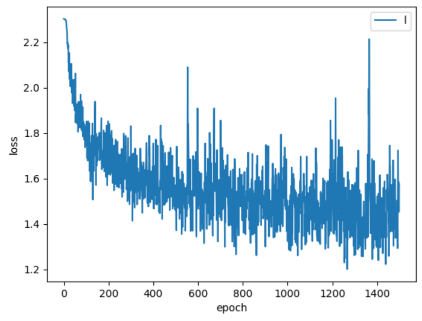
- average gradient over batch
- the local gradient (the one in each module) is multiplied with the one coming from the previous module, which also contains a batch size. Therefore, everything should be done similarly, but for each of the incoming gradients in the batch
- the gradient w.r.t weights and bias should be added when taking derivative of the Linear module

Question 1.2)

Training the numpy version of the multy layer perceptron with the default parameters achieves 0,4645 accuracy on the test set after 1500 batches. The loss curve and accuracy are presented in Figure 1.



(a) train and test accuracy every 100 batches



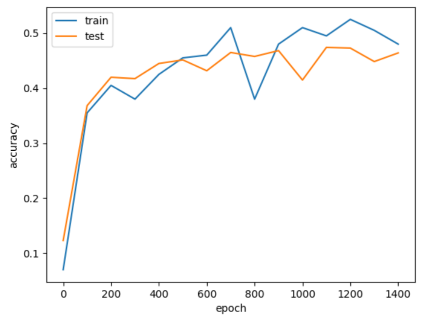
(b) train loss every batch

Figure 1: Learning curves of the numpy MLP with default paramters

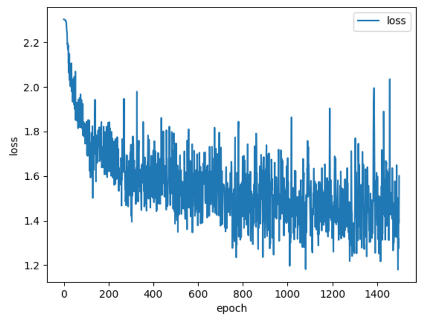
2 PyTorch MLP

Question 2

By utilizing the default values and getting the initialization to be similar to the one in the numpy version of the MLP, the resulting learning curves of the PyTorch MLP are shown in Figure 2.



(a) train and test accuracy every 100 batches



(b) train loss every batch

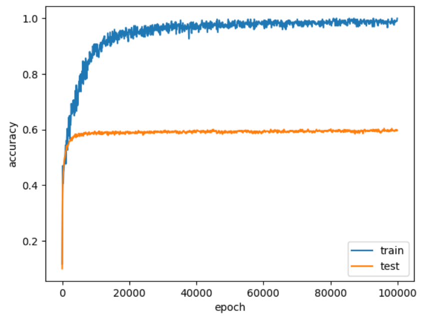
Figure 2: Learning curves of the PyTorch MLP with default paramters

Also, we tried to see how big of an accuracy we can achieve with an MLP (train_mlp_pytorch_best.py). The resulting MLP reaches 0,604 accuracy on the test set. The updates we made to the model are as follows:

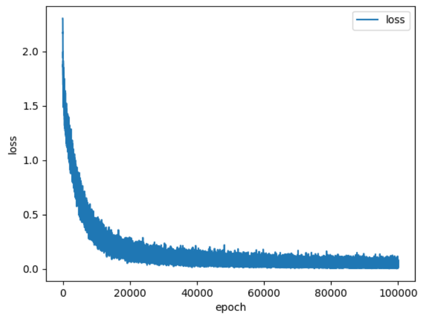
- We used the default initialization in PyTorch, which is He initialization. This initialization tries to make the variance of the inputs of a layer to be equal to the variance of the output of a layer. This initialization method makes the assumption that the input distribution comes from a normal distribution and that the activation is ReLU.
- The optimizer that we used is ADAM due to the well known success it has on various problems.
- We increased the architecture size to: '2000,1000,200,200,200,200,200,200,200'
- We increased the number of steps to 100000
- We added dropout to the inputs of the first 5 layers. This was used as a regularization technique. We did not want this to be too strong, so we chose 0,2 probability of dropping a input.
- Finally, we added weight norm [Salimans2016] in order to solve the vanishing gradient problems, exploding gradient problems, and the possible overfitting problems. This tech-

nique has a similar purpose as batch normalization, but is less biased, because it does not depend on a sample from the dataset or a subset of the dataset.

The results are presented in Figure 3. We can see that for the most part, weight normalization allows for fast convergence and stable training.



(a) train and test accuracy every 100 batches



(b) train loss every batch

Figure 3: Learning curves of the best PyTorch MLP

3 Custom Module: Batch Normalization

3.2 a)

$$\frac{\partial L}{\partial \gamma} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial \gamma} = \left\{\sum_s \sum_i \left(\frac{\partial L}{\partial y}\right)_{si}\left(\frac{\partial y}{\partial \gamma}\right)_{sij}\right\}_j = \left\{\sum_s \sum_i \left(\frac{\partial L}{\partial y_i^s}\right)\left(\frac{\partial y_i^s}{\partial \gamma_j}\right)\right\}_j =$$

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

$$\frac{\partial L}{\partial \beta} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial \beta} = \left\{\sum_s \sum_i \left(\frac{\partial L}{\partial y}\right)_{si}\left(\frac{\partial y}{\partial \beta}\right)_{sij}\right\}_j = \left\{\sum_s \sum_i \left(\frac{\partial L}{\partial y_i^s}\right)\left(\frac{\partial y_i^s}{\partial \beta_j}\right)\right\}_j =$$

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

$$\frac{\partial L}{\partial x} = \left\{\sum_s \sum_i \left(\frac{\partial L}{\partial y}\right)_{si}\left(\frac{\partial y}{\partial x}\right)_{sirj}\right\}_{rj} = \left\{\sum_s \sum_i \left(\frac{\partial L}{\partial y_i^s}\right)\left(\frac{\partial y_i^s}{\partial x_j^r}\right)\right\}_{rj} =$$

(this turns to a summation between

-the derivative of L w.r.t normalized x * derivative of normalized x w.r.t x

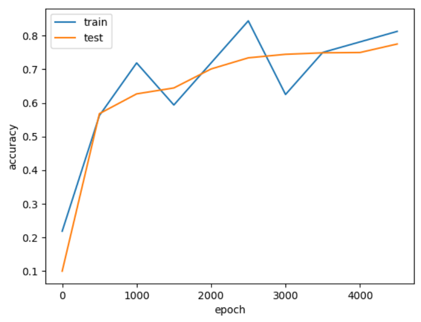
- derivative of L w.r.t. mean * derivative of mean w.r.t. x

- derivative of L w.r.t. variance * derivative of variance w.r.t. x.

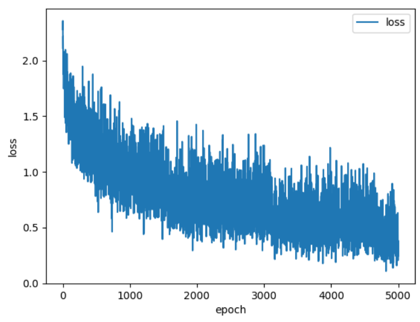
Due to lack of time this is not written explicitly from this point on.

4 PyTorch CNN

We also trained a CNN, which is architecturally a mini version of the VGG-net. They follow the default parameters in the Assignment document. The resulting learning curves are presented in Figure 4. It achieves 0.78 accuracy on the test set.



(a) train and test accuracy every 500 batches



(b) train loss every batch

Figure 4: Learning curves of the PyTorch CNN

References

[Salimans2016] Tim Salimans, Diederik P. Kingma Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks