# Assignment 1: MLPs, CNNs and Backpropagation

**Narendiran Chembu**
12195855
University of Amsterdam
1012 WX Amsterdam
narendiran.chembu@student.uva.nl

## Abstract

In this assignment, we perform image classification task (on CIFAR-10 dataset) implemented using hand-calculated MLP gradients by numpy, MLP auto-grad by Pytorch and CNN Krizhevsky et al. [2012] by Pytorch. Additionally batch-normalization is implemented through Pytorch auto-grad and manually using autograd.Function modules of Pytorch

## 1 MLP backprop and NumPy implementation

### 1.1 Analytical derivation of gradients

Given Module definitions for an MLP:

$$\text{Linear:}\tilde{x}^{(l)} = W^{(l)}x^{(l-1)} + b^{(l)}$$
$$\text{ReLU:}x^{(l)} = max(\tilde{x}^{(l)})$$
$$\text{Softmax:}x^{(N)} = s_i(x^{(N)}) = \frac{\exp\left(\tilde{x}_i^{(N)}\right)}{\sum_j \exp\left(\tilde{x}_j^{(N)}\right)}$$
$$\text{Cross-Entropy Loss:}L = -\sum_i t_i \log x_i^{(N)}$$

(a) Computing the gradients of intermediate modules wrt to immediate outputs: (Note: $\{\mathbb{1}_{(condition)}\}$ refers to Kronecker Delta Function)

$$\frac{\partial L}{\partial x^{(N)}} = \left[..0..\ -\frac{\mathbb{1}_{t_i=1}}{x_i^{(N)}}\ ..0..\right] \quad \textit{dim:}(1 \times d_N)$$
$$\frac{\partial x^{(N)}}{\partial \tilde{x}^{(N)}} : \frac{\partial x_i^{(N)}}{\partial \tilde{x}_j^{(N)}} = s_i(\mathbb{1}_{t_i=j} - s_j) \quad \textit{dim:}(d_N \times d_N)$$

where $s_i$ is the soft-max function corresponding to the $i^{th}$ component. It's a Jacobian Matrix

$$\frac{\partial x^{(l<N)}}{\partial \tilde{x}^{(l<N)}} = \left[\mathbb{I}_{(x_i^{(l<N)}>0)}\right] \quad \textit{dim:}(d_l \times d_l) \text{ - Conditional Identity matrix}$$
$$\frac{\partial \tilde{x}^{(l)}}{\partial x^{(l-1)}} = W^l \quad \textit{dim:}(d_l \times d_{l-1})$$
$$\frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}} : \frac{\partial \tilde{x}_i^{(l)}}{\partial W_{jk}^{(l)}} = \mathbb{1}_{t_{(i=j)}} x_k^{(l-1)} \quad \textit{dim:}\left(d_l \times (d_l \times d_{(l-1)})\right)$$

Submitted to the Spring 2019 Amsterdam Deep Learning course

In matrix form it looks like this:

$$\begin{bmatrix} \begin{bmatrix} .. & x_k^{(l-1)} & .. \\ 0.. & 0 & ..0 \\ 0.. & 0 & ..0 \end{bmatrix} \\ \\ \\ \\ \\ \begin{bmatrix} 0.. & 0 & ..0 \\ 0.. & 0 & ..0 \\ .. & x_k^{(l-1)} & .. \end{bmatrix} \end{bmatrix} \quad \begin{matrix} \\ \\ \\ . \\ . \\ \\ \\ \end{matrix}$$

$$\frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} = [\mathbb{I}] \quad \textit{dim:}(d_l \times d_l) \text{ - Identity matrix}$$

(b) Computing the gradients of individual modules wrt to the final Loss:

$$\frac{\partial L}{\partial \tilde{x}^{(N)}} : \frac{\partial L}{\partial x_i^{(N)}} \frac{\partial x_i^{(N)}}{\partial \tilde{x}_j^{(N)}} = \left[ ..0.. \ -\frac{\mathbb{1}_{t_i=1}}{x_i^{(N)}} s_i (\mathbb{1}_{t_i=j} - s_j) \ ..0.. \right] \quad \textit{dim:}(1 \times d_N)$$

$$\frac{\partial L}{\partial \tilde{x}^{(l<N)}} : \frac{\partial L}{\partial x_i^{(l)}} \frac{\partial x_i^{(l)}}{\partial \tilde{x}_j^{(l)}} = \frac{\partial L}{\partial x_i^{(l)}} \left[ \mathbb{I}_{(x_i^{(l<N)} > 0)} \right] = \left( \sum_j \left[ \frac{\partial L}{\partial \tilde{x}^{(l+1)}} \right]_j \left[ \frac{\partial \tilde{x}^{(l+1)}}{\partial x^{(l)}} \right]_{ji} \right) \left[ \mathbb{I}_{(x_i^{(l<N)} > 0)} \right] \quad \textit{dim:}(1 \times d_l)$$

Here $\frac{\partial L}{\partial \tilde{x}^{(l+1)}}$ refers to the previous module gradients which are computed already

$$\frac{\partial L}{\partial x^{(l)}} : \frac{\partial L}{\partial x_i^{(l)}} = \sum_j \left[ \frac{\partial L}{\partial \tilde{x}^{(l+1)}} \right]_j \left[ \frac{\partial \tilde{x}^{(l+1)}}{\partial x^{(l)}} \right]_{ji} = \sum_j \left[ \frac{\partial L}{\partial \tilde{x}^{(l+1)}} \right]_j [W_{ji}]_{ji}$$

$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial \tilde{x}^{(l)}} \frac{\partial \tilde{x}^{(l)}}{\partial W^{(l)}} : \frac{\partial L}{\partial \tilde{x}_i^{(l)}} \frac{\partial \tilde{x}_i^{(l)}}{\partial W_{jk}^{(l)}} = \left[ \frac{\partial L}{\partial \tilde{x}^{(l)}} \right]_i \left[ \mathbb{1}_{t_{(i=j)}} x_k^{(l-1)} \right]_{ijk} \quad \textit{dim:}(d_l \times d_{l-1})$$

Here a row vector is multiplied by a 3D tensor yielding a matrix of the same dimensions of the weight matrix

$$\frac{\partial L}{\partial b^{(l)}} = \frac{\partial L}{\partial \tilde{x}^{(l)}} \frac{\partial \tilde{x}^{(l)}}{\partial b^{(l)}} : \frac{\partial L}{\partial \tilde{x}_i^{(l)}} \frac{\partial \tilde{x}_i^{(l)}}{\partial b_j^{(l)}} = \left[ \frac{\partial L}{\partial \tilde{x}^{(l)}} \right]_i \left[ \mathbb{1} \right]_{ij} \quad \textit{dim:}(d_l \times 1)$$

The bias gradient vector is computer similarly as above when the previous module gradient vector is computed by the bias gradient matrix

(c) When Batch size $B \neq 1$ is used, the input vector is assumed to be of dimensions $B \times d_0$ and hence the corresponding intermediate activations also obtain an additional "batch" dimension. The final loss is now an average over all the samples in the mini-batch. This causes the vector gradients of activations and inputs to now become matrices and matrix gradients of activations now become tensors.

But the weight and bias (learnable parameters) gradients would retain the same dimensions due to internal multiplication along the batch dimension. This allows them to have a lesser stochastic/noisy jump during the update.

### 1.2 NumPy Implementation

The loss and accuracy curves for the default parameters are given below in Figures 1 and 2:

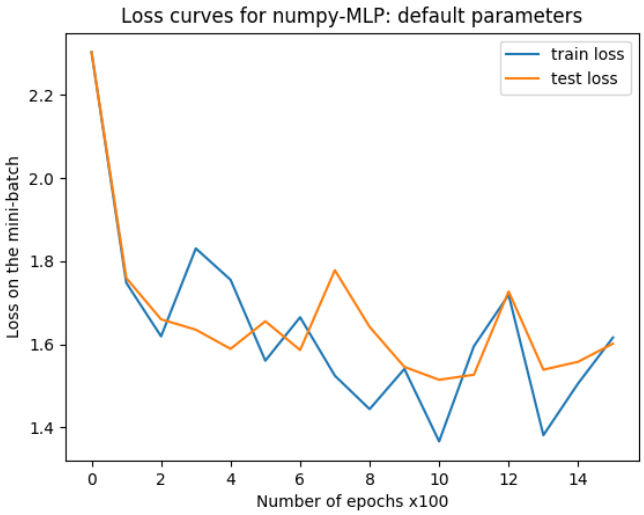


Figure 1: Loss curve for numpy-MLP with default parameters

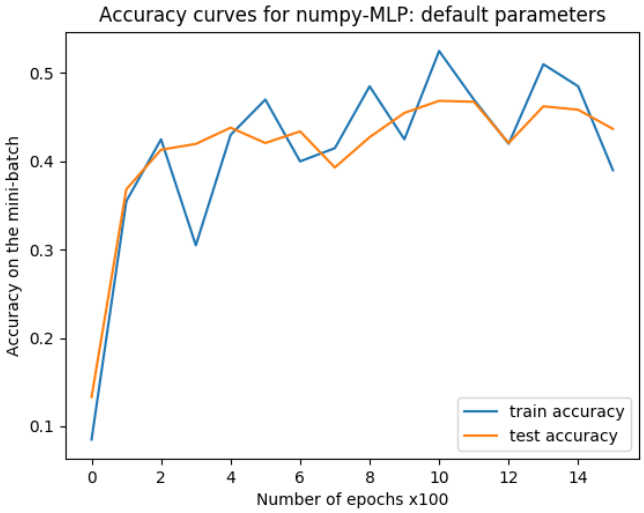


Figure 2: Accuracy curve for numpy-MLP with default parameters

## 2 PyTorch MLP

The loss and accuracy curves for the best parameters are given below in Figures 3 and 4. The parameter list which yields is the best performance ( 53.5%) on a limited grid search is as follows:

Hidden units = '500,200,200,200'
Learning Rate = 8e-4 2e-3
Maximum number of epochs = 3000
Batch size = 300
Optimizer = Adam
Weight Decay (L2 regularization) = 0.01

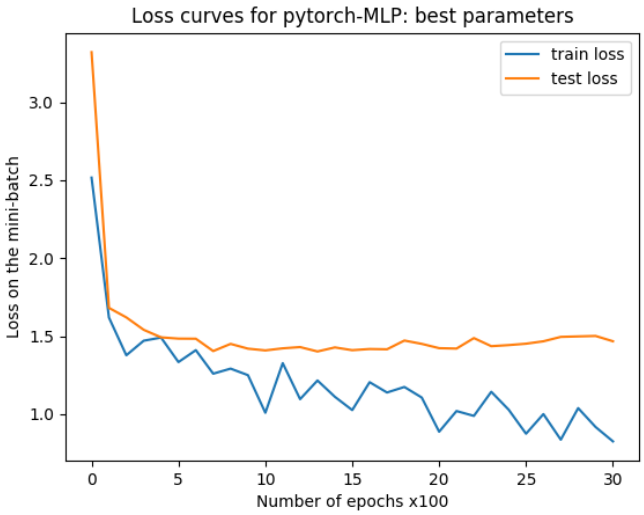


Figure 3: Loss curve for pytorch-MLP with default parameters

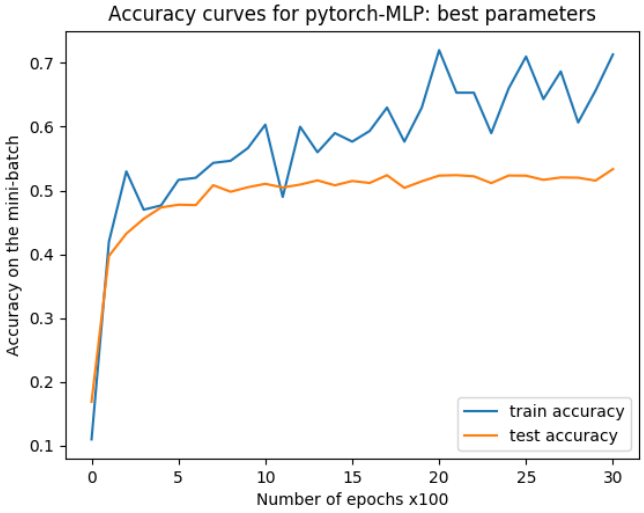


Figure 4: Accuracy curve for pytorch-MLP with default parameters

# 3 Custom Module: Batch Normalization

## 3.1 Automatic differentiation

Batch Normalization operation is implemented as a nn.Module at the designated position in the file custom_batchnorm.py

## 3.2 Manual implementation of backwards pass

Gradient wrt gamma:

$$\frac{\partial L}{\partial \gamma} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial \gamma}$$

Computing wrt individual components:

$$\begin{aligned}\Big(\frac{\partial L}{\partial \gamma}\Big)_j &= \sum_s \sum_i \frac{\partial L}{\partial y_i^s}\frac{\partial y_i^s}{\partial \gamma_j} \\ &= \sum_s \sum_i \frac{\partial L}{\partial y_i^s}\mathbb{1}_{(i=j)}\hat{x}_i^s \\ &= \sum_s \frac{\partial L}{\partial y_j^s}\hat{x}_j^s\end{aligned}$$

Gradient wrt beta:

$$\frac{\partial L}{\partial \beta} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial \beta}$$

Computing wrt individual components:

$$\begin{aligned}\Big(\frac{\partial L}{\partial \beta}\Big)_j &= \sum_s \sum_i \frac{\partial L}{\partial y_i^s}\frac{\partial y_i^s}{\partial \beta_j} \\ &= \sum_s \sum_i \frac{\partial L}{\partial y_i^s}\mathbb{1}_{(i=j)} \\ &= \sum_s \frac{\partial L}{\partial y_j^s}\end{aligned}$$

Gradient wrt x:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x}$$

Computing wrt individual components and batch:

$$\begin{aligned}\Big(\frac{\partial L}{\partial x}\Big)_j^r &= \sum_s \sum_i \frac{\partial L}{\partial y_i^s}\frac{\partial y_i^s}{\partial x_j^r} \\ &= \sum_s \sum_i \frac{\partial L}{\partial y_i^s}\gamma_i\frac{\partial \hat{x}_i^s}{\partial x_j^r} \\ &= \sum_s \sum_i \frac{\partial L}{\partial y_j^s}\gamma_i\frac{\partial\Big(\frac{x_i^s-\mu_i}{\sqrt{\sigma_i^2+\epsilon}}\Big)}{\partial x_j^r}\end{aligned}$$

Let's consider the partial derivative at the end of this equation:

$$\begin{aligned}\frac{\partial\Big(\frac{x_i^s-\mu_i}{\sqrt{\sigma_i^2+\epsilon}}\Big)}{\partial x_j^r} &= \frac{\sqrt{\sigma_i^2+\epsilon}\;\frac{\partial(x_i^s-\mu_i)}{\partial x_j^r} - (x_i^s-\mu_i)\;\frac{\partial(\sqrt{\sigma_i^2+\epsilon})}{\partial x_j^r}}{\sigma_i^2+\epsilon} \\ &= \frac{\sqrt{\sigma_i^2+\epsilon}\;\;\mathbb{1}_{(i=j)}\big[\mathbb{1}_{(s=r)}-1/B\big] - (x_i^s-\mu_i)\;\frac{1}{2(\sqrt{\sigma_i^2+\epsilon})}\frac{\partial\sigma_i^2}{\partial x_j^r}}{\sigma_i^2+\epsilon} \\ &= \frac{\sqrt{\sigma_i^2+\epsilon}\;\;\mathbb{1}_{(i=j)}\big[\mathbb{1}_{(s=r)}-1/B\big] - (x_i^s-\mu_i)\;\mathbb{1}_{(i=j)}\frac{1}{2(\sqrt{\sigma_i^2+\epsilon})}2/B\sum_b(x_i^b-\mu_i)\big[\mathbb{1}_{(b=r)}-1/B\big]}{\sigma_i^2+\epsilon} \\ &= \frac{\mathbb{1}_{(i=j)}}{\sqrt{\sigma_i^2+\epsilon}}\left[\big[\mathbb{1}_{(s=r)}-1/B\big] - (1/B)\;\frac{(x_i^s-\mu_i)(x_i^r-\mu_i)}{\sigma_i^2+\epsilon}\right] \\ &= \frac{\mathbb{1}_{(i=j)}}{\sqrt{\sigma_i^2+\epsilon}}\left[\mathbb{1}_{(s=r)} - (1/B) - (1/B)\;\hat{x}_i^s\hat{x}_i^r\right] \\ &= \frac{\mathbb{1}_{(i=j)}}{B\sqrt{\sigma_i^2+\epsilon}}\left[B\;\mathbb{1}_{(s=r)} - 1 - \;\hat{x}_i^s\hat{x}_i^r\right]\end{aligned}$$

Inserting that back into the original equation:

$$\left(\frac{\partial L}{\partial x}\right)_j^r = \sum_s \sum_i \frac{\partial L}{\partial y_i^s} \gamma_i \frac{\mathbb{1}_{(i=j)}}{B\sqrt{\sigma_i^2+\epsilon}} \left[ B\,\mathbb{1}_{(s=r)} - 1 - \hat{x}_i^s \hat{x}_i^r \right]$$

$$= \sum_s \frac{\partial L}{\partial y_j^s} \frac{\gamma_j}{B\sqrt{\sigma_j^2+\epsilon}} \left[ B\,\mathbb{1}_{(s=r)} - 1 - \hat{x}_i^s \hat{x}_i^r \right]$$

$$= \frac{\gamma_j}{B\sqrt{\sigma_j^2+\epsilon}} \left[ B \frac{\partial L}{\partial y_j^r} - \frac{\partial L}{\partial \beta_j} - \hat{x}_i^r \frac{\partial L}{\partial \gamma_j} \right]$$

$$\frac{\partial L}{\partial x} = \frac{\gamma}{B\sqrt{\sigma^2+\epsilon}} \left[ B \frac{\partial L}{\partial y} - \frac{\partial L}{\partial \beta} - \hat{x} \frac{\partial L}{\partial \gamma} \right]$$

## 4 PyTorch CNN

The loss and accuracy curves for the default parameters are given below in Figures 5 and 6:

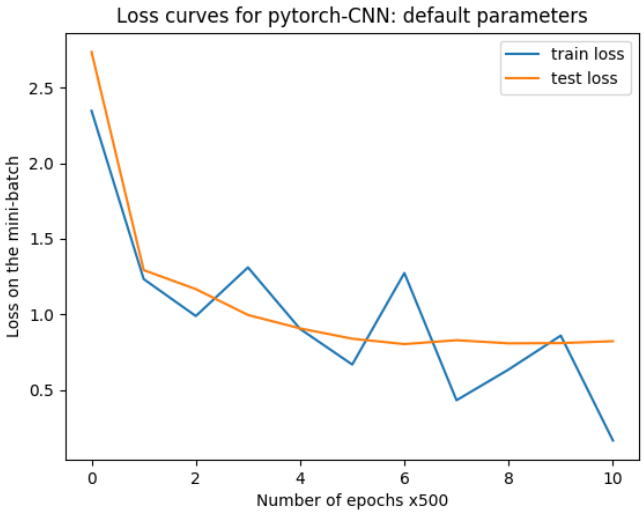


Figure 5: Loss curve for pytorch-CNN with default parameters

## 5 Conclusion

CNN gives the best accuracy because of the kernel structure of neurons and spatial feature learning

## References


Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

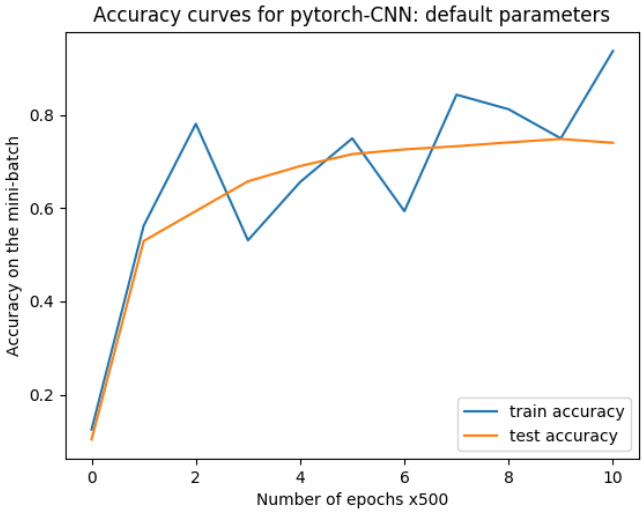


Figure 6: Accuracy curve for pytorch-CNN with default parameters