1 Implementation: MLP with NumPy

In this part of the exercise, you should add your report for the NumPy implementation (**Question 1.2**) of the MLP network using purely NumPy routines. You should provide the accuracy and loss curves here as well as some comments on your findings.

20.0p **a** The grade on this part is related to your implementation. You do not have to enter anything in the cell below.

Click to edit

5.0p **b** Add your accuracy and loss curves here. Also, add your comments/findings related to the implementation here.

2 Implementation: MLP with PyTorch

In this part of the exercise, you should add your report for the Implementation of the MLP network using PyTorch (filling the code for the mlp_torch.py) for **Question 2.1**. You should provide the accuracy and loss curves here, as well as some comments on your findings.

15.0p **a** The grade on this part is related to your implementation. You do not have to enter anything in the cell below.

5.0p **b** Add your accuracy and loss curves here. Also, add your comments/findings related to the implementation here.

3 Theory: Backpropagation

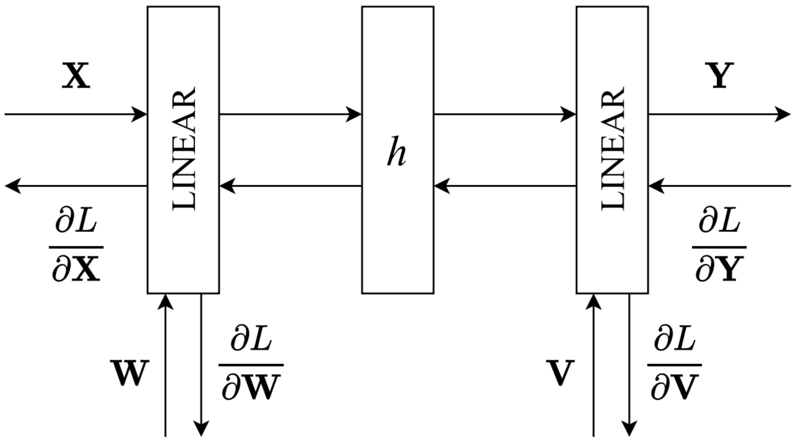
$\mathbf{X}$
LINEAR
h
LINEAR
$\mathbf{Y}$
$\frac{\partial L}{\partial \mathbf{X}}$
$\frac{\partial L}{\partial \mathbf{Y}}$
$\mathbf{W}$
$\frac{\partial L}{\partial \mathbf{W}}$
$\mathbf{V}$
$\frac{\partial L}{\partial \mathbf{V}}$

Figure 1: A schematic of the linear-activation-linear block.

Let there be a linear-activation-linear block with input $\mathbf{X} \in \mathbb{R}^{S\times M}$ and output $\mathbf{Y} \in \mathbb{R}^{S\times N}$ as shown in Fig. 1. The number of samples in each batch is S. The weight matrices are $\mathbf{W} \in \mathbb{R}^{F\times M}$ and $\mathbf{V} \in \mathbb{R}^{N\times F}$ respectively, and the activation function h is applied element-wise. There are M input features, F hidden features, and N output features in this module. (You may ignore biases.)

The gradients necessary for backpropagation in the whole block are of the form:

$$\frac{\partial L}{\partial \mathbf{X}} = (\mathbf{A} \circ \mathbf{B})\mathbf{W}$$

$$\frac{\partial L}{\partial \mathbf{W}} = (\mathbf{A} \circ \mathbf{B})^\top \mathbf{X}$$

$$\frac{\partial L}{\partial \mathbf{V}} = \mathbf{C}$$

Note: All gradients have the shape of the object with respect to which is being differentiated.

5.0p **a** Find closed-form expressions for $\mathbf{A}$ and $\mathbf{B}$ in terms of the backpropagated gradients $\frac{\partial L}{\partial \mathbf{Y}}$ and the other objects (tensors, activation function etc.) defined above.

2.0p **b** Find a closed-form expression for $\mathbf{C}$ in terms of the backpropagated gradients and other objects defined above.

2.0p **c** Briefly mention <u>one</u> drawback of performing backpropagation for a block of modules as a whole, as was done above.

2.0p **d** Biases were ignored in the description above. Explain how you would add biases to the linear layers in the above implementation <u>without</u> explicitly introducing a new object (i.e. without initializing a new list/vector $\mathbf{b}$ which requires new gradients to be calculated from scratch).

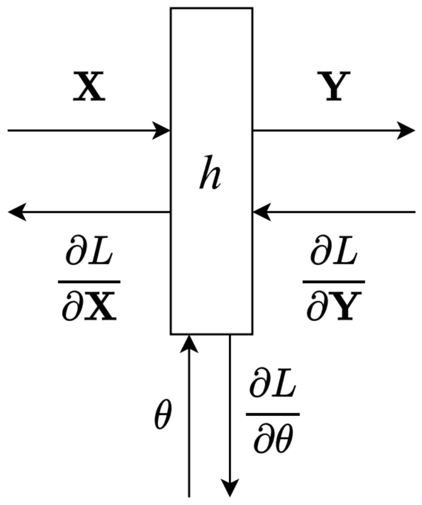
$\mathbf{X}$
$\mathbf{Y}$
h
$\frac{\partial L}{\partial \mathbf{X}}$
$\frac{\partial L}{\partial \mathbf{Y}}$
θ
$\frac{\partial L}{\partial \theta}$

Figure 2: The PReLU activation layer module.

Consider a Parametric ReLU (PReLU) activation function module as in Fig. 2. This activation function is a variant of leaky ReLU and has an optimizable parameter $\theta > 0$. It is defined as follows:

$$h_\theta(z) = \begin{cases} z, & z \geq 0 \\ \theta z, & z < 0 \end{cases}.$$

2.0p **e** The gradient $\frac{\partial L}{\partial \theta}$ can be calculated as the sum of the elements in a matrix $\mathbf{D}$. Formally: $\frac{\partial L}{\partial \theta} = \sum_{i,j} D_{ij}$. Find an explicit, closed-form expression for the matrix $\mathbf{D}$ in terms of the backpropagated gradients $\frac{\partial L}{\partial \mathbf{Y}}$ and the input features $\mathbf{X}$.

Hint: You might find the element-wise maximum/minimum functions to be useful. For example: $[\max(0, \mathbf{X})]_{ij} = \max(0, X_{ij})$.

2.0p **f** The gradient required for backpropagation can be found to have the following form: $\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \circ \mathbf{E}$. What is the matrix $\mathbf{E}$? (Must not be in closed form, i.e. describe its elements.)

4 Implementation: Batch Normalization with PyTorch

In this exercise, you should report your findings for Question 2.2 related to the hyperparameter configurations found in the Table of this question.

3.0p **a** What do you experience when you are running with and without batch normalization?

3.0p **b** Compare the models by plotting their validation and training accuracies over epochs.

4.0p **c** Explain the different behaviors of the models and trends over increasing the network size. What does it tell us about the effect of batch normalization in such networks?

5 Theory: Normalization

2.0p **a** A gradient shows the direction for optimizing each parameter individually, i.e. this direction is only valid when all other parameters remain fixed. However, in training neural networks, the parameters change simultaneously. Explain how batch normalization alleviates this issue.

2.0p **b** Experimental analysis showed that a high percentage of neurons are dead in networks with ReLU activation functions (you can refer to https://uvadlc-notebooks.readt... for more information). Explain the concept of a dead neuron and how it harms training.

3.0p **c** How does batch normalization prevent neurons from dying?

6 Theory: Optimization

Consider point p where $\nabla_{\mathbf{x}} f(\mathbf{x}_p) = \mathbf{0}$; we call this point a *critical* or *stationary* point. If a critical point is not a local maximum or minimum, it will be classified as a *saddle* point.
To determine if a critical point in a higher dimension is a local minimum or maximum, we can use the Hessian matrix check. For continuously differentiable function f and real non-singular (invertible) Hessian matrix H at point p, if H is positive definite we have a strictly local minimum, and if it is negative definite we have a strictly local maximum.

3.0p **a** Show that the eigenvalues for the Hessian matrix in a strictly local minimum are all positive.

3.0p **b** If some of the eigenvalues of the Hessian matrix at point p are positive and some are negative, this point would be a saddle point; intuitively explain why the number of saddle points is exponentially larger than the number of local minima for higher dimensions? *Hint: Think of the eigenvalue sign as flipping a coin with probability* $\frac{1}{2}$ *for a head coming up (positive sign).*

2.0p **c** By using the update formula of gradient descent around saddle point p, show why saddle points can be harmful to training.

7 Theory: Regularization

3.0p **a** Explain why regularization is always applied to the network weights and not its biases?

3.0p **b** What are the differences between using a Norm 2 regularizer and its max-norm counterpart?

3.0p **c** We define the norm ℓ_p for vector $\mathbf{x}$ as $||\mathbf{x}||_p = \left(\sum_{i=1}^{n} |x_i^p|\right)^{\frac{1}{p}}$, What is simple form (not in the form of any powers) for ℓ_∞ and ℓ_0? explain by derivation.

8 Theory: Initialization

6.0p The common initialisation technique for linear layers in a MLP with ReLU activation is the Kaiming initialisation: $W \sim \mathcal{N}\left(0, \frac{2}{d_x}\right)$. As discussed in the lectures, factor 2 is needed because the ReLU sets (in expectation) half of the input to zero.
Suppose we want to replace the ReLU with a LeakyReLU with a negative slope α: $\mathrm{LeakyReLU}(x) = \max(\alpha \cdot x, x)$. How do we have to adjust our initialisation for this activation function? Show a short derivation and explanation of the result. You can assume that the input to the activation function, x_i, is zero-centred: $\mathbb{E}[x_i] = 0$.

Hint: Follow the same steps we have taken for deriving the Kaiming initialization in the lecture/Tutorial 4 (link), which involve the calculation of $\mathbb{E}[\mathrm{LeakyReLU}(x_i)^2]$.

Make use of the symmetry of the input to the activation function around zero. No complicated integrals or similar is necessary for the calculation.
You can verify your result by setting $\alpha = 0$ *which should give you the standard Kaiming initialization.*