EE-411, HomeWork 3 : Neural networks

Your solutions must be given in a Jupyter Notebook containing the implemented code and the theoretical answers. The code cells will be executed for evaluation. This homework requires more computational resources and, thus, a valid option is `Google Colab` for its GPU resources. The total points are 70.

1 Backpropagation with logistic loss [40 points]

Let $f_{\boldsymbol{w}} : \mathbb{R}^D \to \mathbb{R}$ be a single-hidden-layer, fully connected, neural network with no biases and parameters $\boldsymbol{w} = \{\boldsymbol{W}^{(1)}, \boldsymbol{w}^{(2)}\}$, such that its forward pass computes

$$\boldsymbol{x}^{(1)} = \sigma\left(\boldsymbol{z}^{(1)}\right) = \sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{x}^{(0)}\right) \tag{1}$$

$$\hat{y} = \sigma\left(z^{(2)}\right) = \sigma\left(\left(\boldsymbol{w}^{(2)}\right)^{\top}\boldsymbol{x}^{(1)}\right). \tag{2}$$

Here, $\sigma(\cdot)$ denotes a sigmoid activation. In what follows, we will assume $D = 7$ and $K = 5$.

1. **[5 points]** Implement a function `predict(X,W)` that given a batch of B samples $\{\boldsymbol{x}_i\}_{i=1}^B$ (an array `X` of size $B \times D$) and a dictionary of weights (`W={w_1: D x K, w_2: K x 1}`) computes the set of outputs $\{(\boldsymbol{z}_i^{(1)}, z_i^{(2)}, \hat{y}_i)\}_{i=1}^B$. Do not use `for` loops.
2. **[5 points]** Implement a function `logistic_loss(y, y_hat)` that given a vector of labels $\boldsymbol{y} \in \{0,1\}^B$ and a vector of predictions $\hat{\boldsymbol{y}} \in \mathbb{R}^B$ computes the average logistic loss of a batch. Compute the average loss for the case $\hat{y} = \mathbf{0}$ and $\boldsymbol{y} = \mathbf{0}$. Explain your result. *Reminder :* Given a true label $y \in \{0,1\}$ and a soft prediction $\hat{y} \in [0,1]$, the logistic loss is defined as

$$\mathcal{L}(y, \hat{y}) = -y \log(\hat{y}) - (1-y)\log(1-\hat{y}). \tag{3}$$

3. **[6 points]** Implement another function `stable_logistic_loss(y, z_2)` that given a vector of labels $\boldsymbol{y} \in \{0,1\}^B$ and a vector of logits $\boldsymbol{z}^{(2)} \in \mathbb{R}^B$ computes the average logistic loss of a batch. Make sure that your function is stable. Compute the average loss for the case when $\boldsymbol{z}_2 = -10^{10} \cdot \mathbf{1}$ and $\boldsymbol{y} = \mathbf{0}$. Explain your result. *Hint :* You can use `np.logaddexp`.
4. **[12 points]** Derive analytically, using the backpropagation algorithm, the expressions for the partial derivatives of the loss with respect to the weights, i.e.,

$$\frac{\partial \mathcal{L}(\boldsymbol{x}, y; \boldsymbol{w})}{\partial w_{ij}^{(1)}} \quad \text{and} \quad \frac{\partial \mathcal{L}(\boldsymbol{x}, y; \boldsymbol{w})}{\partial w_i^{(2)}}, \tag{4}$$

   when the loss is computed using a single pair $(\boldsymbol{x}, y)$. Recall that we are using the logistic loss, and that the final implementation should be stable.
5. **[12 points]** Implement a function `gradient(X,y,W)` which computes the gradient (i.e., vector of partial derivatives) of the average loss with respect to all the weights for a batch $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^B$. Do not use `for` loops.

2 Classifying KMNIST using neural networks [30 points]

In this exercise you will play with the dataset called KMNIST, you can download from https ://github.com/rois-codh/kmnist. You can use code snippets from the labs. Note that the quantitative results are not as important as the qualitative ones.

1. **[4 points]** Load the dataset and construct the dataloaders for train, validation and test, and visualize the data. Use an 50000 and 10000 images for train and validation, respectively. Which transform is necessary for the samples to be compatible with the models we will create ?
2. **[6 points] MultiLayer Perceptron (MLP)** : Construct a two-hidden-layer MLP with 100 neurons (per layer), ReLU activation functions and a linear output layer to classify KMNIST. Train this model for 20 epochs using the cross-entropy loss and the following optimizers : SGD (`lr=0.01`), SGD with momentum (`lr=0.01, momentum=0.9, nesterov=True`), Adam (`lr=0.01`) and Adam (`lr=1`). Plot the training and validation learning curves (loss against steps and epochs) on a single plot. Comment on your results.
3. **[6 points] Convolutional Neural Network (CNN)** : Construct a CNN with three convolutional layers (`kernel_size=3`) and 16, 32, and 64 channels, respectively ; a non-linearity and one max-pooling layer (`kernel_size=2`) after every convolution ; and a final fully connected layer. Train this model with the same four configurations as before, and plot the training and validation learning curves on a single plot. Comment on your results.
4. **[6 points]** Create a function that computes the number of parameters of a given model. Show the number of parameters for the two models you have used. Does more parameters translate to better performance ? Explain.
5. **[8 points]** `PermutedKMNIST`[1] : In this version of the dataset, the pixels are randomly permuted. Visualize the new dataset. Train one MLP and one CNN using SGD with momentum `lr=0.01, momentum=0.9, nesterov=True`). What do you observe ? Is one model more affected than the other ? Explain.
6. (*Bonus* **[5 points]**) Using any of the tips and tricks seen in class or during the exercise sessions, optimize the validation performance on the original KMNIST and report both validation and test performance. Also feel free to explore things beyond the ones seen in class. In any case, explain your decisions. Did you manage to improve the performance ? Why ?

```
class RandomPermutation(object):
    def __init__(self, num_features):
        self.num_features = num_features
        self.reindex = torch.randperm(num_features)

    def __call__(self, img):
        assert self.num_features == img.numel()

        orig_shape = img.shape
        img = img.view(-1)[self.reindex].view(orig_shape)

        return img
```

1. Add the tranformation `RandomPermutation` to the list of transforms. This transform should be the last one.