EE-411, HomeWork 3 : Neural networks

This homework involves some coding, with the language of your choice. Present your results with graphs, plots, and data. Jupyter notebooks are a good option, and we recommend you to send your work as a notebook on Google colab.

1 Backpropagation with logistic loss

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

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

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

Here, $\sigma(\cdot)$ denotes a sigmoid activation. In what follows, we will assume $D = 5$ and $K = 6$.

1. 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 $\{(\boldsymbol{z}_i^{(1)}, z_i^{(2)}, \hat{y}_i)\}_{i=1}^B$. *Hint :* Try to avoid using `for` loops.
2. 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.
3. 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. Derive analytically, using the backpropagation algorithm, the expressions for the partial derivatives of the loss with respect to the weights, i.e.,

$$\frac{\partial \mathcal{L}(\boldsymbol{x}, y; \boldsymbol{w})}{\partial w_{ij}^{(1)}} \quad \text{and} \quad \frac{\partial \mathcal{L}(\boldsymbol{x}, y; \boldsymbol{w})}{\partial w_i^{(2)}}, \tag{3}$$

   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. 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 $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^B$. *Hint :* Try to vectorize your code and expressions.

2 Classifying `FashionMNIST` using neural networks

In this exercise you will play with the dataset called `FashionMNIST`. Some examples are shown in Figure 1. You can use code snippets from the labs.

1. Load the dataset, construct the dataloaders and visualize the data. Which transform is necessary [1] ?
2. **MultiLayer Perceptron (MLP)** : Construct a two hidden-layer MLP with 100 neurons and ReLU activation functions to classify `FashionMNIST`. Train this model using the cross-entropy loss using SGD (`lr=0.01`), SGD with momentum (`lr=0.01, momentum=0.9, nesterov=True`) and Adam (`lr=0.1`), and plot the training and test learning curves for every epoch on a single plot. Comment on your results.

1. for the samples to be compatible with the models we will create

FIGURE 1 – Some examples from the `FashionMNIST` dataset.

3. **Convolutional Neural Network (CNN)** : Construct a CNN with three convolutional layers (`kernel_size=3`) and 16, 32, and 64 channels, respectively ; one max-pooling layer (`kernel_size=2`) after every convolution ; and a final fully connected layer. Train this model with the same configurations as before, and plot the training and validation learning curves on a single plot. Comment on your results.
4. 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. `PermutedFashionMNIST` [2] : In this version of the dataset, the pixels are randomly permuted. Visualize the new dataset. Train one MLP and one CNN with the best parameters from before. What do you observe ? Is one model more affected than the other ? Explain.
6. (*Bonus*) Using any of the tips and tricks seen in class or during the exercise sessions, optimize the test performance on the original `FashionMNIST`. 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
```

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