CSCI 4961/6961: Homework 6

Assigned Friday March 17 2023. Due by 11:59pm Friday March 31 2023.

Create a Jupyter notebook for this assignment, and use Python 3. Write documented, readable and clear code (e.g. use reasonable variable names). Submit this notebook making sure the answers to each question are legible and clearly labeled. You will be graded primarily based on the solutions and answers already present in the notebook, but it must also be runnable to reproduce your results.

In this homework, you will implement a one hidden-layer autoencoder by hand. You will follow a realistic workflow: you will implement a specific architecture, then explore the hyperparameter space to find choices that make this architecture work well, document the performance, and reflect on the results. **NB, this will take time: both to debug your backprop implementation, and to do your fitting and hyperparameter selection.**

1. Write a class `autoencoder` that implements a one hidden-layer autoencoder with loss function given by average binary cross-entropy error. The autoencoder will be trained, roughly, as follows:

```
ae = autoencoder(d, n1)
for e in range(numepochs):
    print("Training loss [%d/%d]: %.2f" % (e+1, numepochs, ae.loss(Xtrain)))
    minibatches = np.split(np.random.permutation(ntrain), numsplits)
    for indices in minibatches:
        Xbatch = Xtrain[:, indices]
        ae.forward(Xbatch)
        ae.backward()
        ae.adam_step()
```

To enable this training workflow, the class should implement the following functions:

(Constructor) A constructor `__init__(self, d, n1)`, where `d` is the dimensionality of the input vectors and `n1` is the dimensionality of the hidden layer.

The constructor should initialize the weights and biases for the two layers, `W1`, `W2`, `b1`, and `b2`. Initialize the biases to zero, and sample the initial weights using the Glorot initialization scheme[1]

$$w_{ij}^{\ell} \sim \text{Unif}\left[-\sqrt{\frac{6}{n_{\ell-1}+n_{\ell}}}, \sqrt{\frac{6}{n_{\ell-1}+n_{\ell}}}\right].$$

Also initialize here the quantities necessary to implement the Adam algorithm described below: the timestep counter, and the storage for the gradient and curvature information for the parameters `W1`, `W2`, `b1`, and `b2`.

(Activations) Activation functions `h1(self, a)` and `h2(self, a)` for the hidden and output layers, respectively. They should accept matrix arguments and respectively return the pointwise evaluation of the ReLu and logistic sigmoid activation functions.

(Activation derivatives) The derivatives of the activation functions, `dh1(self, a)` and `dh2(self, a)`. They should accept matrices and return the pointwise evaluation of the derivatives of the corresponding activation functions.

(Loss) The loss function `loss(self,Xbatch)` that computes the average binary cross-entropy reconstruction loss when the autoencoding of `Xbatch` is used to approximate `Xbatch`, given by

$$\ell(\mathbf{Y}, \mathbf{X}) = -\frac{1}{dm}\sum_{i=1}^{d}\sum_{j=1}^{m}\left[x_{ij}\ln(y_{ij}) + (1-x_{ij})\ln(1-y_{ij})\right],$$

[1] See `http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf` for more details.

where $\mathbf{Y}$ is the output of the autoencoder given the input $\mathbf{X}$. This function should not change any of the attributes of the instance.

(Loss derivative) The derivative of the loss function given above with respect to the output of the autoencoder, `dloss(self, Ybatch)`. It should assume that `Ybatch` was obtained by passing a minibatch through the autoencoder, and use the internal state of the autoencoder to compute the derivative of the loss above with respect to the output `Ybatch`.

(Forward) An inference/forward-pass function `forward(self, Xbatch)`, where `Xbatch` is a $d \times m$ matrix whose columns comprise a minibatch. The inference function should compute the output of passing this minibatch through the autoencoder. In particular, the returned matrix should have the same size as `Xbatch`.

To facilitate their use in backpropagation, this function should store `Xbatch` and the activations `a1` and `a2` and outputs `o1` and `o2` as instance attributes.

(Backward) A backprop/backward-pass function `backward(self)`. This function should compute and store attributes `dW1`, `dW2`, `db1`, `db2` containing the gradients of the training objective with respect to the parameters of the autoencoder, using the values stored from an earlier call to `forward`.

(Adam Optimizer) A function `adam_step(self, alpha=0.001, rho1=0.9, rho2=0.999, delta=1e-8)` that implements a single step of the Adam optimization algorithm to update the autoencoder's parameters. This function assumes that `forward` then `backward` were called immediately before it to generate the quantities needed in the optimization process.

As discussed in class, the Adam algorithm stores exponentially weighted estimates of the gradients and curvature information for the variables being optimized, rescales them to debias them, and applies an update similar to AdaGrad. At the t-th timestep, Adam updates a given parameter $\mathbf{P}$ (a bias vector or weight matrix) according to the scheme

$$\begin{aligned} \texttt{nuP} &\leftarrow \rho_1 \texttt{nuP} + (1-\rho_1)\texttt{dP} \\ \texttt{hP} &\leftarrow \rho_2 \texttt{hP} + (1-\rho_2)\texttt{dP} \odot \texttt{dP} \\ \texttt{nuPhat} &\leftarrow (1-\rho_1^t)^{-1}\texttt{nuP} \\ \texttt{hPhat} &\leftarrow (1-\rho_2^t)^{-1}\texttt{hP} \\ \texttt{P} &\leftarrow \texttt{P} - \alpha \cdot \texttt{nuPhat}/(\sqrt{\texttt{hPhat}} + \delta). \end{aligned}$$

The quantities `nuP` and `hP` respectively contain gradient and curvature information that must be stored as instance attributes. The timestep counter used in the optimization algorithm should be incremented inside this function.

2. Load the Fashion MNIST dataset from Keras and scale and reshape it appropriately for input into your autoencoder class. Train the autoencoder with `n1`=100 until you are satisfied with its reconstruction results: you choose the minibatch sizes, number of epochs, and Adam hyperparameters (if you choose to modify these). For this portion, you may find it helpful to collaborate with other students to discuss and choose good hyperparameter choices.

3. Choose ten images randomly from the test set, one from each class, and display them in a two by ten grid, where the images in the first row are the original images and the images in the second row are the corresponding reconstructed images using the autoencoder. Your submission pdf should contain this image and the following: the final loss on your test set, and your choices for the hyperparameters. If you derive any insights from visual inspection of the reconstructed images or over the course of your hyperparameter selection process, include these.