# Assignment 4 (12.5 pts)

**Due March 31, 2023, 23:59 EST**

*Starter code for this assignment is provided in* neural_network.py *on Quercus.*

**Q1) 3pts** Consider the hidden unit output given by $z_i = \sigma(\sum_{j=1}^{D} w_{ij} x_j)$ where $z_i$, $w_{ij}$, and $x_j$ $\in \mathbb{R}$ and $\sigma : \mathbb{R} \to \mathbb{R}$ is an activation function. Assuming that each input is a normally distributed random variable $x_j \sim \mathcal{N}(0, \eta^2)$ along with a linear (identity) activation, Xavier initialization randomly initializes each weight, $w_{ij}$, from a normal distribution $\mathcal{N}(0, \varepsilon^2)$ where the standard derivation, $\varepsilon$, is chosen so that,

$$\mathrm{Var}(z_i) = \eta^2.$$

Derive the standard deviation, $\varepsilon$, which will satisfy this condition. Using this derivation, complete the init_xavier function provided in the starter code.

**Q2) 2pts** Consider the function neural_net_predict provided in the starter code. This function takes a set of parameters and inputs and returns the predicted log-probabilities for each class. Explain why we have used logsumexp to compute the log-probabilities instead of a more naive approach such as:

```
outputs = outputs - np.log(np.exp(outputs).sum(-1, keepdims=True))
```

**Q3) 2pts** Write out the log-likelihood for multiclass classification assuming a model, $\hat{\mathbf{f}}(\mathbf{x}, \mathbf{w})$, outputs normalized class probabilities. Complete the function mean_log_like provided in the starter code. This function takes a set of parameters, inputs, and targets and returns the log-likelihood divided by the number of inputs in the batch. Note that neural_net_predict returns normalized class log-probabilities.

**Q4) 3.5pts** Tune the parameters of a neural network to perform classification on the MNIST dataset. The starter code provided shows how to train a neural network with a single hidden layer with 200 units. Feel free to tune the number of layers, the number of hidden units, the number of epochs, and the learning rate. With a bit of tuning, you should be able to achieve an accuracy of approximately 95% on the validation set. For your final model, plot the training loss and validation loss versus the epoch count and comment on their convergence. Report your final model's training, validation, and test accuracy. Explain your procedure for the tuning the hyperparameters of the network.

**Q5) 2pts** Using the model trained in Q4, plot the confusion matrix for the validation set. You are free to use sklearn.metrics.confusion_matrx and sklearn.metrics.ConfusionMatrixDisplay to create and display the confusion matrix. Using insights from the confusion matrix, suggest two strategies to improve your model.

**Submission guidelines:** Submit an **electronic copy** of your report (**maximum 10 pages** in at least 10pt font) in **pdf** format and **documented** Python scripts. You should include a file named "README" outlining how the scripts should be run. Upload both your report in pdf format and a single tar or zip file containing your code and README to Quercus. You are expected to verify the integrity of your tar/zip file before uploading. Do not include (or modify) the supplied *.npz data files or the data_utils.py module in your submission. The report must contain

- Objectives of the assignment
- A brief description of the structure of your code, and strategies employed
- Relevant figures, tables, and discussion

Do not use scikit-learn for this assignment, the intention is that you implement the simple algorithms required from scratch. Also, for reproducibility, always set a seed for any random number generator used in your code. For example, you can set the seed in numpy using `numpy.random.seed`.