Optimization for Machine Learning (Spring 2026): Project

- You **must email your submission** to kbala@wsu.edu. The project report should be in **PDF format** and and its name should identify you in this manner. If you are Darryl Weathers, you should name your submission DarrylWeathers_Project.pdf. **Please avoid white spaces in the file name**.
- To submit your code/implementation, you should **include all files in a compressed folder**, e.g., zipped or tar-gzipped, named in the same manner, i.e., **DarrylWeathers_Project.zip**. You could include the PDF file inside this zipped folder, along with the other files.
- WSU Email will **not allow attachments of .py files**, even inside Zipped folders! You can save such files as .txt files, or include corresponding Jupyter notebooks (.ipynb files) instead.
- **Begin the SUBJECT of your email submission with the same `FirstnameLastname`, expression, e.g., "DarrylWeathers Project submission".**
- **This project is due by 10:00 PM on Friday, May 8**.

You will be implementing standard neural networks (NNs) for binary classification from scratch. The default implementation is expected to be done in Python using NumPy. You will compare your own implementation with an equivalent function in PyTorch. You will also compare your NN results with those obtained with SVM that you implemented in Homework 3.

1. (50) Implement a general N-layer feedforward network with the following configuration.

   - Configurable layer widths $(n_0, n_1, \ldots, n_N)$ passed as a list, where n_0 is the input dimension. Note that the width n_ℓ of layer ℓ is the number of nodes in that layer, and hence is also the length of the activation vector $\mathbf{y}^{(\ell)}$ of that layer as well as the number of rows in the weight matrix $W^{(\ell+1)}$.
   - **ReLU** activations in all hidden layers, $\Phi(\mathbf{s}) = \max(\mathbf{s}, \mathbf{0})$, applied elementwise.
   - A **linear** final layer (no activation), producing a raw score $s^{(N)} \in \mathbb{R}$ per example.

   The implementation must follow the decoupled sub-node formulation from the course notes, so that the code structure mirrors the theory. Your variable names should ideally match the notation introduced in the lectures: $\boldsymbol{\delta}^{-(\ell)}$, $\boldsymbol{\delta}^{+(\ell)}$, $\mathbf{s}^{(\ell)}$, $\mathbf{y}^{(\ell)}$.

   Loss function: Use the logistic regression loss function with Tikhonov regularization:

$$J = \sum_{i=1}^{n} \log\left(1 + e^{-y_i s_i^{(N)}}\right) + \frac{\lambda}{2} \sum_{\ell=1}^{N} \|W^{(\ell)}\|_F^2, \tag{1}$$

   where $y_i \in \{-1, +1\}$ and $s_i^{(N)}$ is the network's scalar output for example i, i.e., the i-th component of $\mathbf{s}^{(N)}$. Note that this y_i is the truth value of sample i and is not related to any layer output $\mathbf{y}^{(\ell)}$. This loss function will allow a direct comparison with the results from your previous implementation of logistic regression SVM. Also, we want to use the Frobenius norm of $W^{(\ell)}$ here as a direct generalization of the Euclidean norm of weight vector $\|\mathbf{w}\|_2$ used in the SVM loss function. You can adapt the Adam method implemented in Homework 3 for optimization here.

2. (15) Before starting any training, verify your backpropagation implementation using a centered finite-difference check:

$$\frac{\partial J}{\partial W_{jk}^{(\ell)}} \approx \frac{J(\mathbf{w} + \varepsilon \mathbf{e}_{jk}) - J(\mathbf{w} - \varepsilon \mathbf{e}_{jk})}{2\varepsilon}, \qquad \varepsilon = 10^{-5},$$

where $\mathbf{w}$ is the vector concatenation of all trainable weights $W^{(\ell)}$ across all layers and $\mathbf{e}_{jk}$ is the unit vector of the same length as this $\mathbf{w}$ with its 1 in the entry corresponding to the $W_{jk}^{(\ell)}$ entry. Check a random sample of at least 20 weight entries across all layers. Report the maximum relative error, defined as

$$\text{rel. error} = \frac{|\text{analytical} - \text{numerical}|}{\max(|\text{analytical}|, |\text{numerical}|) + 10^{-8}},$$

and confirm that it is below 10^{-4}. Repeat this check for *each* NN configuration you work with. **Do not proceed to training until this check passes.** This check corresponds to verifying the relation (3) in Theorem 15 in the Lecture Notes.

3. (10) Set all $W^{(\ell)} = [O]$ (zero matrix). Train a two-hidden-layer network for several epochs and observe the training loss. Can you explain why the approach fails?

   After this demonstration, you will use **He initialization** (also known as Kaiming initialization, after Kaiming He, who proposed it in 2015) for all your experiments. Draw each entry independently as

$$W_{jk}^{(\ell)} \sim \mathcal{N}\left(0, \frac{2}{n_{\ell-1}}\right), \text{ where } \mathcal{N} \text{is the normal distribution.}$$

4. **Computational Tests:** Use (some of) the same datasets you used in Homework 3, picking at least one on which logistic regression SVM performed well and one on which it struggled (or required more tuning). Also use the same training/test/validation splits of the data as used previously, so that the comparisons can be as direct as possible.

   (5) **[Sanity check]** Set $N = 1$ (no hidden layers). This NN configuration is identical to the logistic regression SVM model. For each of your datasets, demonstrate that the training loss (as a curve) of this NN matches that of logistic regression SVM (within numerical tolerances) and so does the final accuracy.

   (35) **[Effect of depth]** Fix the hidden layer width n_ℓ to the same value for all ℓ. For at least two datasets, report how the loss curves and accuracy vary as you increase the number of hidden layers $N = 1, 2, 5, 10$. You could use the (best) learning rates you used in Homework 3. Report the best accuracies for each data set for logistic regression SVM (from Homework 3), shallow NN ($N = 1, 2$), and deep NN ($N = 5, 10$).

   (25) **[PyTorch comparison]** Re-implement the same NN using `torch.nn.Sequential`, `torch.nn.Linear`, and `torch.optim.Adam` with identical hyperparameters and He initialization via `torch.nn.init.kaiming_normal_`. On **one** dataset, show that the NumPy and PyTorch training loss curves agree for the first 20 epochs (as a verification of correctness of your implementation). Report wall-clock training time per epoch for both implementations. Comment briefly on the source of the speedup and at what approximate network size the gap becomes practically significant.

5. (20) Submit a report (10–12 pages) detailing all your findings. Also comment on the following topics in your report.

   (a) **When does depth help?** On which dataset did adding hidden layers improve over the logistic-SVM, and by how much? If the improvement is small, is the added complexity justified?

   (b) **What the degenerate case tells you.** Because the zero-hidden-layer NN with logistic loss is mathematically identical to the logistic-SVM, any accuracy gain from adding layers is attributable entirely to the nonlinear representational capacity of the hidden layers. What does this controlled comparison reveal about the geometry of the decision boundary for each dataset?

   (c) **Computational cost and non-convexity.** The SVM loss is convex with a guaranteed global minimum. The NN loss is non-convex. Did you observe any sensitivity to initialization or training instability that was absent in the SVM optimization? What does this suggest about the practical tradeoff between model capacity and ease of optimization?