AERSP 597: Homework 2

Out: 02/13/2025

I. Warm-up Exercises (20 points)

1. Manually construct a neural network with one hidden layer to emulate the XNOR function. The network has two inputs and one output, all of which are either 0 or 1. The XNOR function maps two bits to one bit like this: $(0,0) \to 1$, $(0,1) \to 0$, $(1,0) \to 0$, $(1,1) \to 1$. Hint: You will need a *nonlinear* activation function for the hidden layer.
2. In class we have shown that the forward and backward modes of automatic differentiation are essentially computing the Jacobian-Vector Product (JVP) and the Vector-Jacobian Product (VJP) of a nonlinear vector function, respectively. Show that computing the JVP of a function is equivalent to computing the VJP of VJP of the same function, and vice versa.

II. Forward- and Back-Propagation (30+10 points)

In this question, you will implement a 4-layer neural network for regression. In the parts labelled $[A/B]$, you can do

1. Either *Track A (10 bonus points)* implement everything by yourself, including the forward and backward propagation. A skeleton code is available in `NN_Skeleton.py` for you to get started.
2. Or *Track B* use a standard package, such as `PyTorch` or `TensorFlow` to bypass some of the implementations, esp. the back-propagation.

The input layer (L1) contains $M+1$ units including a bias term and M is the dimensionality of the data. The first hidden layer (L2) has $K+1$ hidden units including a bias term. The connection weights between L1 and L2 is represented by matrix $\mathbf{A} \in \mathbb{R}^{K\times M+1}$. The element of $\mathbf{A}$ on the ith row and jth column is denoted $[a_{i,j}]$. Layer L2 uses a Swish activation function

$$h(x) = \frac{x}{1+e^{-x}} \tag{1}$$

and the output of the ith node in L2 is

$$y_i = h\left(a_{i,M+1} + \sum_{j=1}^{M} a_{i,j}x_j\right) \tag{2}$$

Next, the second hidden layer (L3) contains $D+1$ hidden units including a bias term. The connection weights is denoted as matrix $\mathbf{B} \in \mathbb{R}^{D\times K+1}$. Layer L3 also uses the Swish activation function again and the output of the ith node in L3 is

$$z_i = h\left(b_{i,K+1} + \sum_{j=1}^{K} b_{i,j}y_j\right) \tag{3}$$

The last layer (L4) contains N units for a N-dimensional output. The connection weights between L3 and L4 is denoted by the matrix $\mathbf{C} \in \mathbb{R}^{N\times D+1}$. Layer L4 uses the ReLU activation function,

$$g(x) = \max(0, x) \tag{4}$$

and the output is,

$$\hat{t}_i = g\left(c_{i,K+1} + \sum_{j=1}^{D} c_{i,j}z_j\right) \tag{5}$$

For a dataset $\mathcal{D} = \{\mathbf{x}_i, \mathbf{t}_i\}_{i=1}^{n}$, where $\mathbf{x}_i \in \mathbb{R}^M$ and $\mathbf{t}_i \in \mathbb{R}^N$, the loss function is defined as

$$E(\mathbf{A},\mathbf{B},\mathbf{C}) = \frac{1}{2}\sum_{i=1}^{n} ||\hat{\mathbf{t}}_i - \mathbf{t}_i||^2 \tag{6}$$

1. Find the gradients of the loss function w.r.t. the weights
$$\frac{\partial E}{\partial \mathbf{A}}, \quad \frac{\partial E}{\partial \mathbf{B}}, \quad \frac{\partial E}{\partial \mathbf{C}} \tag{7}$$
First, use the chain rule to expand these gradients, e.g. $\frac{\partial E}{\partial \mathbf{A}} = \frac{\partial E}{\partial \mathbf{t}} \frac{\partial \mathbf{t}}{\partial \mathbf{A}}$. Then, find the expressions for each term (e.g. $\frac{\partial E}{\partial \mathbf{t}}$ and $\frac{\partial \mathbf{t}}{\partial \mathbf{A}}$) in the chain rule expansion. Note, You do NOT need to substitute these expressions back into the chain rule expansion.
2. $[A/B]$Implement the forward-propagation of the neural network.
3. $[A/B]$Implement back-propagation algorithm.
4. Perform gradient checking to verify the gradient implementation. The procedure for gradient checking is the following:
   (a) Choose dimensionality: Set $M = 100$, $K = 50$, $D = 30$, $N = 10$
   (b) Generate fake data: Uniformly sample $\mathbf{x}$ from $[-0.05, 0.05]^M$ and $\mathbf{t}$ from $[-0.1, 0.1]^N$
   (c) Generate fake weights: Uniformly sample $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ from $[-0.05, 0.05]^{K\times(M+1)}$, $[-0.05, 0.05]^{D\times(K+1)}$ and $[-0.05, 0.05]^{N\times(D+1)}$, respectively.
   (d) Gradient checking: Fix $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{x}$ and $\mathbf{t}$. Sample row index i and column index j 1000 times (e.g. for 50x101 matrix $\mathbf{A}$, sample i from $\{0, 1, ..., 49\}$ and j from $\{0, 1, ..., 100\}$). For each sample of (i, j), compute the gradient of E w.r.t. the (i, j)th entry numerically, and compare the value to the gradient found by the backpropagation algorithm.
   (e) Reporting: Compute the mean absolute error (MAE) of the gradients associated with the 1000 samples of (i, j).

You may choose either of the following two methods to compute the numerical gradient,

1. Central difference (use $\epsilon = 10^{-8}$):
$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} \approx \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon} \tag{8}$$
2. Complex step (use $\epsilon = 10^{-12}$):
$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} \approx \frac{\mathrm{imag}(f(x+\mathrm{i}\epsilon))}{\epsilon} \tag{9}$$

For each of the gradients $\frac{\partial E}{\partial \mathbf{A}}$, $\frac{\partial E}{\partial \mathbf{B}}$, and $\frac{\partial E}{\partial \mathbf{C}}$, report the MAE values. Reasonable MAE values would be below 10^{-3} (central difference) or 0 (complex step).

III. Training by Gradient Descent (30 points)

In this problem, you will implement three typical stochastic gradient descent (SGD) methods to train the neural network developed in the previous problem using the training and test datasets `aerothermal_train.csv` and `aerothermal_test.csv`.

Even if you chose Track B in the previous problem, from its Part 4 you should be able to obtain the gradients that are necessary for this problem.

The three SGD methods are listed in the following. Note that the weights are updated in an *element-wise* manner, i.e. the gradient descent is carried out for each *individual* weight.

1. SGD with fixed learning rate
$$w_{k+1} = w_k - \eta \frac{\mathrm{d}E}{\mathrm{d}w_k} \tag{10}$$
2. SGD with adaptive moment estimation (Adam)
$$\begin{aligned} v_k &= \beta_1 v_{k-1} + (1-\beta_1)\tfrac{\mathrm{d}E}{\mathrm{d}w_k} \\ g_k &= \beta_2 g_{k-1} + (1-\beta_2)\left(\tfrac{\mathrm{d}E}{\mathrm{d}w_k}\right)^2 \\ \hat{v}_k &= \tfrac{v_k}{1-\beta_1^k} \\ \hat{g}_k &= \tfrac{g_k}{1-\beta_2^k} \\ w_{k+1} &= w_k - \tfrac{\eta}{\sqrt{\hat{g}_k+\epsilon}}\hat{v}_k \end{aligned} \tag{11}$$

   where $\beta_1 = 0.9$, $\beta_2 = 0.999$.

3. Nesterov-accelerated Adam (Nadam), which is Adam with the momemtum correction $\hat{v}_k$ replaced by

$$\hat{v}_k = \frac{\beta_1 v_k + (1-\beta_1)\frac{\mathrm{d}E}{\mathrm{d}w_k}}{1-\beta_1^k} \tag{12}$$

For this problem, M and N are determined by the sample dataset, K and D are specified below.

1. Implement the three SGD methods.
2. Let $K = D = 5$, and train the neural network using the three SGD methods. Plot the training and test errors versus iteration for each method. Caveats:
   (a) This time, it would be better to normalize the data by `min` and `max`, instead of by `mean` and `std`.
   (b) For the initial guess of the weight matrices, try using Xavier and/or He initialization.
   (c) The training error would decrease with more iterations, but the test error would decrease initially and start to increase after a certain number of iterations. Therefore, during the training, it would be necessary to record the best weights associated with the minimum test error so far. These weights will be the final optimal weights for the neural network.
3. Comparing the convergence history of the three SGD methods, which method is the fastest? Try to explain why is so.

IV. Learning Dynamics (20+10 points)

The students with the top 3 most accurate model gets the 10 bonus points; if the model were developed by a team of N students, each student would get 10/N bonus points.

Same procedure as HW1, you are given a dataset of trajectories from a dynamical system, and asked to identify the dynamics model from the data.

This is an open-ended question, in the sense that you could try as many as possible methods from the neural network modules to produce a model that is as accurate as possible.

You may use existing packages to experiment different methods/models quickly, but in the submitted work, please implement your own model.

Dataset The dataset consists of `train_dyn_hw2.npy` and `test_dyn_hw2.npy`. The dynamical system is 3-dimension. The sample trajectories are recorded with a step size of 0.04 s and 51 steps. The training and test datasets contain 40 and 10 trajectories, respectively.

Assessment First, plot and compare the predicted and true trajectories in the test dataset.

Second, like in HW1, the accuracy of a model should be quantified by two errors, one-step error ϵ_o and roll-out error ϵ_r, using the test dataset. Report the mean and maximum of ϵ_o and ϵ_r of your model.