Deep Learning in Scientific Computing

Project: Thermal Storage Design

Duy Lai
(13-827-530)

General procedure for training the neural networks. We use feed-forward networks to approximate the maps from inputs to outputs of the concerned model and L-BFGS optimizer to minimize the loss functions. Then for each network, we perform an ensemble training on a broad range of (hyper)-parameters to select the configuration with the best trade-off between relative training and validation errors. Those that we inspect include the number of hidden layers, neurons and regularization parameter. In order to get more satisfying errors, we tune "by hand" these parameters around their selected values to see if this could be achieved. We will use this procedure for all of the tasks.

Task 1: Function Approximation

- We first normalize the training and testing sets by using the standard score method[1] (standardizing). All the columns are now of mean $\mu = 0$ and of standard deviation $\sigma = 1$.
- Next, we create two feed-forward networks to approximate the maps, one for the fluid $T_f^{0,*} \approx T_f^0 : t \longmapsto T_f(0,t)$, the other for the solid $T_s^{0,*} \approx T_s^0 : t \longmapsto T_s(0,t)$.
- We train both networks on the training set with the procedure described above, apply them to the standardized testing set, then scale the obtained values back with the mean and standard deviation of the training set to get final results.

Remark. From now on, without having to repeat at every tasks, if we normalize/standardize the training and/or testing sets in the first place, we will *always* scale the values computed from the testing set, or equivalently the final results, back to the original range.

Task 2: Observable Predictions

- We first standardize all the three training sets, the testing set and also the sequences of Sobol points.
- We pick one of the training sets and assign separately each sequence of Sobol points to their respective observable (thus eight neural networks in total), then create another network mapping the eight variables to the capacity factor (hence its input dimension is 8 and output dimension is 1).
- We train separately all the networks, then apply the first eight on their respective column of the testing set to obtain the observable quantities. With these quantities/variables, we employ the last model to finally get the capacity factor.

Task 3: Time Series Forecasting

- Out of the four phases of a cycle, the idle phase after discharging (the last one) is the most difficult one to predict since its temperatures vary the most after each cycle. However, we speculate that the temperatures "converge" as a function of time and thus could approximate this convergence[2] with $\left(T_f^0, T_s^0\right) \sim \log_5(t)$. Naturally, since a future cycle is most related to its direct antecedents, we amplify their weight by adding 5^{n-1} times the n^{th} cycle into the training set.
- We are now more concerned with predicting the temperatures as a function of cycle than of time, we can thus disregard the time. We create two networks, each one takes an increasing sequence of numbers[3] as input and, respectively, T_f^0 and T_s^0 in *ascending* order as output; together they form our new training set.
- Similarly to Task 1 and 2, we standardize the *modified* training set, testing set and train the networks. We then apply them to the "testing" increasing sequence of numbers to obtain the predicted values for T_f^0 and T_s^0. The last step is to rearrange them back to the old order, then assign them to the time of the testing set, i.e. to our future cycle.

[1] Given the training set X, its standardized version is given as $Z = (X - \mu[X])/\sigma[X]$.

[2] "Convergence" here means that the temperatures vary less and less over time, we may not understand it with the strict mathematical definition.

[3] The sequence has the same number of values as the time.

Task 4: Inference of Fluid Velocity

- We do *not* normalize or manipulate the data in this task. Given the training set, we train a network which takes two inputs (time t and velocity of the fluid u_f) and one output (temperature T_f^L). Its map is defined as

$$\mathcal{L}: (t, u_f) \longmapsto T_f^L = \mathcal{L}(t, u_f). \tag{1}$$

- We are given the *set* of measurements $\left(t,\, T_f^{L,*}\right)$. The goal is to infer the target velocity u_f^* which, together with the time t, generates $T_f^{L,*}$. Let us therefore define the *scalar* cost function

$$G(u_f) := \left\| \mathcal{L}(t, u_f) - T_f^{L,*} \right\|_2, \tag{2}$$

  where $\|.\|_2$ denotes the 2-norm.

- Since $T_f^{L,*} \approx \mathcal{L}(t, u_f^*)$, we can infer u_f^* by minimizing $G(u_f)$ on the continuous domain $[u_f^1, u_f^N]$, where $\{u_f^k\}_{k=1,\dots,N}$ are the N discrete values of u_f in the training set, given in the ascending order $u_f^k < u_f^{k+1}$.

Task 5: Design of Storage Geometry

- In the training set, we normalize the diameter $D \in [2, 20]$ and the volume $v \in [50, 400]$ onto the intervals $[0, 1]$ while keeping the capacity factor CF.

- We train a network taking D and v as inputs and CF as output. Its map is defined as

$$\mathcal{L}: (D, v) \longmapsto CF = \mathcal{L}(D, v). \tag{3}$$

- Similar to Task 4, we are interested in finding the geometries that correspond to the reference value CF_{ref} with the cost function

$$G(D, v) = \left(\mathcal{L}(D, v) - CF_{ref}\right)^2. \tag{4}$$

- The solution to minimizing $G(D, v)$ is two-dimensional, it is thus not an unique point but a curve γ. In practice, we approximate γ with a set of N points (D^*, v^*). This can be done by firstly initializing (D^*, v^*) randomly in the domain $[0, 1] \times [0, 1]$. With M different retrain seeds, we obtain M different initial points (D_i^*, v_i^*). Then for each point, we minimize (4) with the L-BFGS optimizer and obtain (D^*, v^*). If the point after optimizing is outside the domain $[0, 1] \times [0, 1]$, we do not include it in the final N points[4] and continue with a new initial one (D_i^*, v_i^*).

[4]Hence M and N do not necessarily equal each other, but rather $M \geq N$.