CME 216/ME 343
Machine Learning for Computational Engineering
D. Patel
Winter 2024

Stanford University

CME 216/ME 343: Homework 5 [Part A]

Total number of points: 95.

Review the slides and demo computer codes provided on Canvas and discussed in class to prepare for this homework. Download `HW5_PartA_starter_code.ipynb` from Canvas to complete this Part A of this assignment. Write your text-based answers directly in the notebook by creating text cells and code parts in corresponding code cells. When submitting, convert the notebook to a PDF file and upload it to Gradescope just like previous homeworks. Ensure that all figures, code, and text answers are visible in this PDF.

# Part A: DNN-based surrogate modeling for transient diffusion equation

## Background

Diffusion of heat in a material can be modeled using following transient heat conduction equation

$$\begin{aligned} \frac{\partial u(\boldsymbol{x},t)}{\partial t} &= \nabla \cdot (\kappa \nabla u(\boldsymbol{x},t)) + f(\boldsymbol{x},t) && \boldsymbol{x} \in \Omega, t \in (0,T) \\ u(\boldsymbol{x},t) &= 0 && \boldsymbol{x} \in \partial\Omega, t \in (0,T) \\ u(\boldsymbol{x},0) &= u_0(\boldsymbol{x}) && \boldsymbol{x} \in \Omega \end{aligned}$$

Here, $f$ represents the heat source, $\kappa$ denotes the thermal diffusivity (which measures the ease with which heat can diffuse through a material), and $u(\boldsymbol{x},t)$ represents the temperature at location $\boldsymbol{x}$ at time $t$. The symbols $\Omega$ and $\partial\Omega$ represent the domain and the boundary of the material under consideration, respectively.

Given the appropriate value of the initial condition $u_0(\boldsymbol{x})$ and boundary condition, one can solve the first equation above using an appropriate numerical method (such as finite element or finite difference) to obtain the temperature solution at some later time $T$, denoted as $u(\boldsymbol{x},T)$. However, solving this partial differential equation (PDE) using numerical methods is computationally expensive and slow. In this Part A of HW5, you will build a DNN-based surrogate in PyTorch that maps the initial condition field to the final temperature field. Once the model is trained, it will serve as a very fast and computationally inexpensive surrogate for predicting the final temperature field from any appropriate initial conditions.

## Dataset

For this task, we will utilize the MNIST dataset to model different initial condition fields. The corresponding final temperature fields are obtained using numerical methods. The complete dataset is provided on Canvas. A sample pair is depicted in Figure 1. The primary objective of this assignment is to develop your skills in creating a complete machine learning pipeline from data splitting to model development to training using PyTorch while learning about important concepts such as hyper-parameter tuning, effect of different optimizers, surrogate modeling, etc.

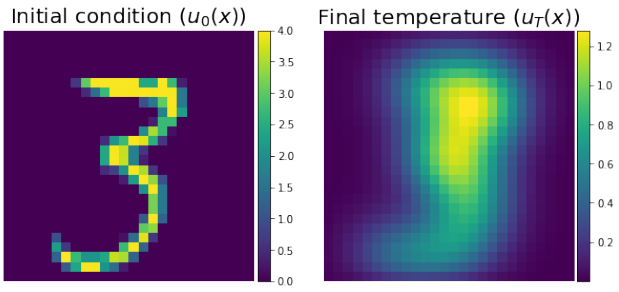


Figure 1: Sample initial condition (left) and final temperature field (right) from the dataset.

## Logistics

- Although not mandatory, we highly recommend using a GPU to complete this assignment. If you do not have access to GPU locally, you may use Google Colab. In Google Colab, you can request access to a GPU by selecting "Runtime" from the menu, then "Change runtime type". This will bring up a dialog box where you can select "T4 GPU" under the hardware accelerator option for your runtime. Colab will then restart the runtime with a GPU assigned to it (it is free).
- You will have to download the assignment data and the starter code from Canvas (from Files/Assignments/HW5_PartA), which comes in the form of two NumPy arrays—one for the initial temperature field (`u0_data.npy`) and the other for the final temperature field (`uT_data.npy`)—each of size (10000, 26, 26).
- If you are using Google Colab, you will need to first upload this data to Google Drive and then mount that Drive to your Colab notebook to load and use this data in your Colab notebook. You can mount the Google Drive in Colab using following command:

  ```
  from google.colab import drive
  drive.mount('/content/drive', force_remount=False)
  ```

  Make sure to use the same Google account for both Google Drive and Google Colab.
- You can find instructions to export your Colab notebook to a PDF at `https://edstem.org/us/courses/51655/discussion/4236442`, or simply add a cell with the following code at the end of your notebook on colab and replacing the path in the last line with the path to your notebook on Colab (you must have mounted the drive first):

  ```
  !pip install nbconvert
  !apt-get install texlive texlive-xetex texlive-latex-extra pandoc
  !jupyter nbconvert --to pdf /content/drive/MyDrive/CME216/hw5_notebook.ipynb
  ```

  The PDF should appear at the same location as the notebook in your drive (its path is indicated in the output of the cell). Note that this operation might take a few minutes to run.

## Questions

1. 5 points. Data wrangling: Split the data into training, validation, and test sets. Use 80% data for training, 10% for validation, and remaining 10% for testing. Since, we are going to use fully connected network here (which can only take vector inputs and produces vector outputs), you need to flatten

the data images into vectors of size $26 \times 26 = 676$. Construct a training dataloader using PyTorch's `torch.utils.data.TensorDataset` utility.

2. 15 points. Model construction: Write a class called `FullyConnectedNetwork` that takes as input:

   - `input_dim`: dimension of the input
   - `output_dim`: dimension of the output
   - `n_layers`: number of layers
   - `n_units`: number of nodes/neurons per layer
   - `activation`: type of activation function

   and builds a fully connected network out of it.

3. 10 points. Training routine: Write a function called `train_and_plot` which takes as argument following things:

   - `model` (model – an object instantiation of the class `FullyConnectedNetwork`)
   - `optimizer` (type of optimizer for performing gradient updates)
   - `max_epochs` (maximum number of epochs)
   - `batch_size` (number of samples in each batch)

   The function should perform the entire training routine for `max_epochs` number of epochs using `optimizer` optimizer and squared loss function.

   Additionally, this function should be able to do the following tasks:

   (a) 10 points. Print the following quantities on the screen at every $100^{th}$ epoch:

   - epoch number
   - training loss
   - validation loss
   - The mean relative error (in percentage) for the validation set.
     The mean relative error (in percentage) is defined as

$$\frac{1}{N_{\text{val}}}\Big(\sum_{i=1}^{N_{\text{val}}} \frac{\|\boldsymbol{u}_{\text{true}}^{(i)} - \boldsymbol{u}_{\text{pred}}^{(i)}\|_2}{\|\boldsymbol{u}_{\text{true}}^{(i)}\|_2} \times 100\Big),$$

     where $\boldsymbol{u}_{\text{true}}^{(i)}$ and $\boldsymbol{u}_{\text{pred}}^{(i)}$ refer to the true temperature field at time T and the predicted temperature field at time T from DNN, respectively, for the $i^{th}$ validation sample and $\|\cdot\|_2$ refers to the Frobenius norm.

   (b) 10 points. At the end of training for `max_epochs` number of epochs, the function `train_and_plot` should also plot the training and validation loss curves as shown in Figure 2.

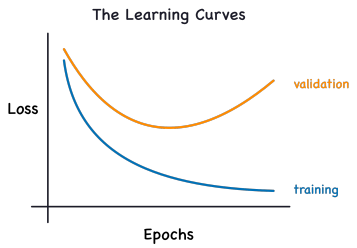


Figure 2: Learning curves to be drawn at the end of training

For plotting such curves, you may store the training and validation loss values at every $100^{th}$ epochs during training in some array/list.

(c) 10 points. At the end of training for `max_epochs` number of epochs, the function `train_and_plot` should also plot the predictions for 10 randomly picked samples from the **testing set** as shown in Figure 3.

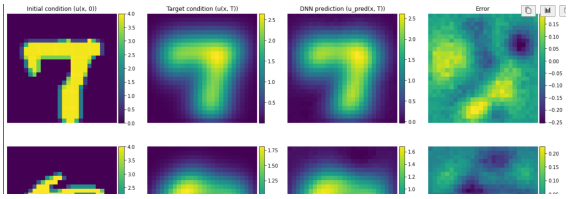

Figure 3: Sample prediction plot figure. The actual figure should have 10 rows and 4 columns like shown here.

You may use the helper function `prediction_plots` provided in the starter code for this.

4. 20 points. Selecting optimizer: As discussed in class, tuning hyperparameters is an important step in deep learning. Typically, this is done in multiple passes, where the first pass involves selecting important hyperparameters, and subsequent passes involve selecting related secondary hyperparameters. In this question, we will select the best optimizer for the given dataset.

   For this task, first create a model using the `FullyConnectedNetwork` class with `n_layers = 8` and `n_units = 700`. Set `input_dim` and `output_dim` to be equal to the data dimension (676). Use the ReLU activation function for all layers except the last layer, where no activation function should be used. Train this model using the `train_and_plot` function ( with the mean squared error (squared L2 norm) as loss function) with four different optimizers one after the other: Full batch gradient descent, Mini-batch gradient descent, Adam, and RMSProp. Use a batch size of 1024 for optimizers that need it. Set `max_epoch = 2000` and learning rate=1e-3 for all optimizers. Submit your code and results. Your results should include training and validation loss, as well as the relative error printed out, along with training and validation curves and prediction plots corresponding to each optimizer.

   Comment on your observations. Which optimizer performs best? Which criteria are you using to judge the relative performance of different optimizers.

5. 15 points. Tuning hyperparameters: Once we have selected the best optimizer, in this second pass of hyperparameter tuning, we will select the best performing optimizer and tune its learning rate.

   For this, select the same model architecture and other hyperparameters as in Question 4 above and pick the best-performing optimizer from the four you tried. Now, train this model with learning rates

= [1e-5, 1e-2, 1]. Turn in your code and results. Your results should include training and validation loss, as well as the relative error printed out, along with training and validation curves and prediction plots corresponding to each learning rate value for this optimal DNN.

Comment on your observations. Which learning rate performs best? Which criteria are you using to judge the relative performance? Why do you think the specific value of learning rate performed best compared to other two.