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

Stanford University

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

Total number of points: 70.

Review the slides and wandb demo code shared on Canvas and discussed during the class to prepare for this homework. There is no starter code for this part. Instead use the wandb demo code as a reference code to complete this.

Surrogate Modeling of Darcy's Flow Using Convolutional Autoencoder

To turn in your results for this part B of HW5, you will need to create a report using WandB, which is a tool for visualizing and tracking machine learning experiments. This report should be similar to the one demonstrated in class. Here is a step-by-step explanation:

1. Create a free WandB account: if you don't already have a WandB account, you will need to create one. It is free to create an account.
2. Write Python code: you will write Python code that populate results data in your WandB account dashboard. This results data will be used to create the WandB report.
3. Use WandB to generate the report: once you have the data in your WandB account, you can use WandB to generate the report. You also need to include your code in the report.
4. Share the report: you need to share the WandB report with the teaching staff using a shared URL. To do this, copy the URL of the report and paste it in the textbox of Homework 5 - Part B posted on Gradescope.

Background and motivation

In this assignment, you will design and implement a surrogate model model (using a convolutional autoencoder) of Darcy's flow to map the permeability field to the pressure field. We will consider a stationary incompressible flow for simplicity. Darcy's flow is used extensively in many science and engineering applications to determine the flow through permeable media. Figure 1 displays a representative permeability

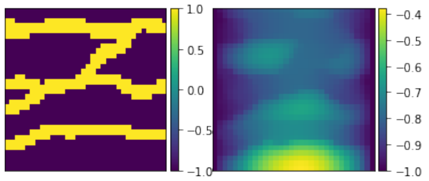

Figure 1: Sample permeability (left) and pressure field (right) from the training set.

and pressure field from the training set. The permeability field represents the structure of the groundwater channel inside the Earth's surface. To obtain the pressure field that corresponds to a given permeability field and boundary conditions, a partial differential equation (PDE) needs to be solved. However, solving the PDE using traditional numerical methods is computationally demanding, especially for online and many-query applications. In this assignment, you will construct a deep learning-based surrogate model of Darcy's flow, which is significantly less expensive computationally. This deep learning model will take the permeability field as input and generate the pressure field as output.

The main goal of this assignment is to develop your skills in creating an end-to-end machine learning pipeline using PyTorch and integrating it with WandB, a tool for monitoring training progress, optimizing hyperparameters, and documentation.

Logistics

- Although not mandatory, we recommend using a GPU to complete this assignment. 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.
- You will have to download the assignment data from Canvas (from Files/Assignments/HW5_PartB), which comes in the form of two NumPy arrays—one for the permeability field and the other for the pressure field—each of size (4608, 32, 32).
- 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 will need to create a WandB account to complete this assignment. You can create an account for free by signing up at WandB.
- You will also need to install the wandb python library. You can do this by executing the command `pip install wandb` in your local machine. If you are using Google Colab, you can install it by entering `!pip install wandb` in the first cell of your notebook.
- After installing the wandb library, you can import it as a Python library using the command `import wandb`. To log your results to the dashboard of your online WandB account, you need to provide your unique API key to your Python code by calling `wandb.login(key = "your_api_key_here")` method. You can obtain your API key by navigating to your account settings on the WandB website.
- The provided WandB demo code on Canvas can be used as a reference.

Questions

1. 15 points. Write a class named `CAE` (Convolutional Autoencoder) that takes as input:
   - `dim`: the number of feature maps/channels in the first convolutional layer.
   - `h_dim`: dimension of the hidden/bottleneck layer

   Turn in your code.

The purpose of the class is to construct an Autoencoder based on the provided input argument `dim` and `h_dim`. The CAE comprises an encoder, decoder, and a hidden/bottleneck layer. Both the encoder and the decoder consist of four convolutional blocks and transpose convolutional blocks, respectively. The convolutional block comprises a 2D convolution layer (`torch.nn.Conv2d`), a leakyrelu activation layer (`torch.nn.LeakyReLU`), and a 2D batchnormalization layer (`torch.nn.BatchNorm2d`). The kernel size for all 2D convolution layers is set to 4, with a stride of 2, and padding of 1. The input channel of the first 2D convolutional layer is 1, as the given image has dimensions (1, 32, 32). The output channel of the first 2D convolutional layer is `dim`, and the subsequent convolutional layers should double the number of output channels each time.

The `models.py` *file in the sample wandb demo code follows a similar encoder architecture.*

The output of the encoder is reshaped into a 2D tensor or matrix of shape [`batch_size`, `p`], where `p` is the size of the output of the encoder, i.e., the number of output channels $\times$ height $\times$ width of the output of the encoder, then connected to a hidden/bottleneck layer of dimension `h_dim` (using `torch.nn.Linear()`), followed by a leakyrelu activation layer (`torch.nn.LeakyReLU`) and a 1D batchnormalization layer (`torch.nn.BatchNorm1d`). The output of this batchnorm layer is then connected to another layer of dimension `p` (using `torch.nn.Linear()`), followed by a leakyrelu activation layer, and a 1D batchnormalization layer. Finally, the output of the last batchnorm layer is reshaped into a 4D tensor of appropriate shape [`batch_size`, `q`, 2, 2] and passed into the decoder.

The `models.py` *file in the sample wandb demo code does not contain this part and you have to implement it. You will need to figure out the appropriate dimensions (*`p` *and* `q`*).*

The decoder comprises four transpose convolutional blocks, each consisting of a 2D transpose convolution layer (`torch.nn.ConvTranspose2D`), a leakyrelu layer, and a batchnormalization layer. The last transpose convolutional block comprises only of a 2D transpose convolutional layer and a leakyrelu layer. Like the convolutional blocks, the kernel size for all the transpose convolutional layers is set to 4, with a stride of 2, and padding of 1. The number of output channels for the first three transpose convolutional layers should be one-half of the number of their respective input channel, and the last transpose convolutional layer should have only 1 output channel to match the output dimensions (1, 32, 32).

The `models.py` *file in the sample wandb demo code follows a similar decoder architecture.*

Finally, the hyperparameter `negative_slope` for all leakyrelu layers should be set to 0.25. Note that this parameter is currently set at 0.2 in the demo code.

Using the above class CAE, we will construct an end-to-end training pipeline to train this reduced-order model. Our goal is to optimize the performance of CAE on the validation set by performing a grid search for the hyperparameters: `dim`, `h_dim`, and β_1 (where, β_1 is the decay rate for the first moment estimates in Adam optimizer). We will try the following hyper-parameter values for `dim=[16, 32]`, `h_dim=[64, 128, 256]`, and β_1=`[0.1, 0.9]`.

2. Given some `dim` and `h_dim`, implement Python code as described in the questions below. Turn in your code.

   (a) 10 points. Normalize the data so that the range is between -1 and 1. Split the data into training and validation sets. Use 80% data for training and the remaining 20% for validation. Construct a training dataloader using PyTorch's `torch.utils.data.TensorDataset` utility.

   Implement a training pipeline with the mean squared error (squared L2 norm) as the optimization criterion and Adam optimizer as the optimizer. Use a batch size of 1024. Iterate through the *train loader* constructed above for 150 epochs.

For each batch of training data, perform the forward pass using your model with the permeability field as input and the pressure field as output. Calculate the loss between the predicted pressure field and ground truth pressure field using the mean squared error objective, backpropagate through the network to compute the gradients, and perform parameter updates using the Adam optimizer with learning rate of 10^{-3}.

The `wandb_demo.py` file in the sample wandb demo contains a similar training pipeline.

(b) 20 points.

i. Print (on screen) and track/plot (on wandb) the training and validation loss at regular intervals of 15 epochs.

ii. Additionally, compute the mean relative error (in percentage) for the validation set and print (on screen) and track it on wandb at the same regular intervals.
The mean relative error (in percentage) is defined as

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

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

iii. Create a histogram of the relative error of individual samples in the validation set, defined as

$$\frac{\|\boldsymbol{u}_{\text{true}}^{(i)} - \boldsymbol{u}_{\text{pred}}^{(i)}\|_2}{\|\boldsymbol{u}_{\text{true}}^{(i)}\|_2} \times 100,$$

and track it at the same regular interval.

iv. Pick any 5 random samples from the validation set at the beginning of training and track how your model performs on these samples over the course of training. For this, you may plot a 5×4 figure, where each row corresponds to an individual random sample. In the first column, plot the permeability field. In the second column, plot the true pressure field. In the third column, plot the predicted pressure field. In the final column, plot the difference between the predicted and true pressure fields.

The `wandb_demo.py` and `utils.py` files in the sample wandb demo code demonstrates how to perform steps i, ii, and iv as outlined above. It does not cover step iii.

3. 10 points. Run the Python code that you have written for the previous question and try the following hyper-parameter values for `dim=[16, 32]`, `h_dim=[64, 128, 256]`, and β_1 `= [0.1, 0.9]`. You will be running the simulation pipeline 12 times using a WandB grid search. Hyperparameter tuning in WandB is called a Sweep. Report (using a WandB report) the best mean relative error (in percentage) and the corresponding hyper-parameter configuration {`dim`, `h_dim`, β_1}. If your implementation is correct, the percent error on the validation set for the better performing hyper-parameter combinations should be less than or around 8%.

4. 15 points. Create a WandB report by including all the plots you generated (in step 2b above) (training loss, validation loss, mean relative error plot, histogram, sample prediction). Include these plots in your report and add suitable text descriptions/captions for each plots. Also include the parallel coordinate plot and hyper-parameter importance plots for your hyper-parameter sweep. Include your Python code in the WandB report. To include your code you may create a code-block directly in the WandB report by typing `/code` and simply copy-paste your code there.

Comment on your observations. Which hyper-parameters are most important? What is the general trend with different hyper-parameter (does increasing a specific hyper-parameter value leads to improvement or degradation of performance on validation set and why?). Do you observe any peculiar behavior? If so what are they and why? Include all your comments in the appropriate place in the WandB report. Create a shareable link of this report and paste it on Gradescope for grading.