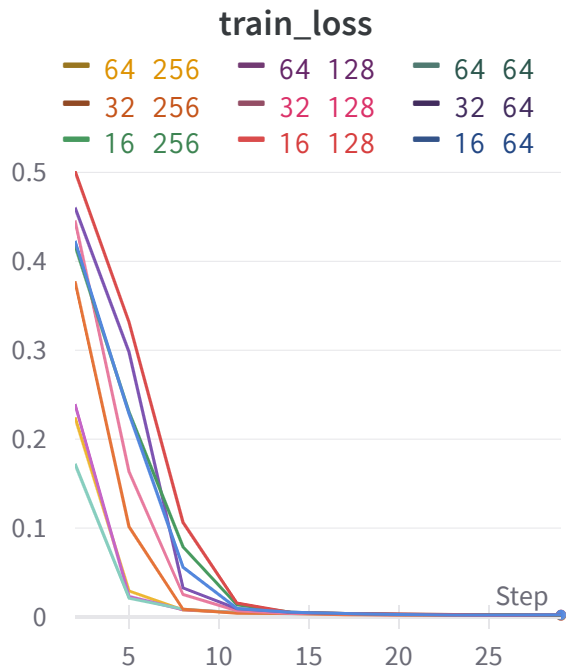
CME216 - Homework 6

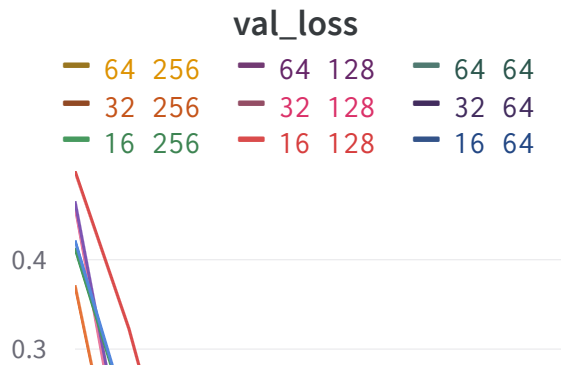
Report for the 6th homework of CME216: Machine Learning for Computational Engineering

Alban Broze

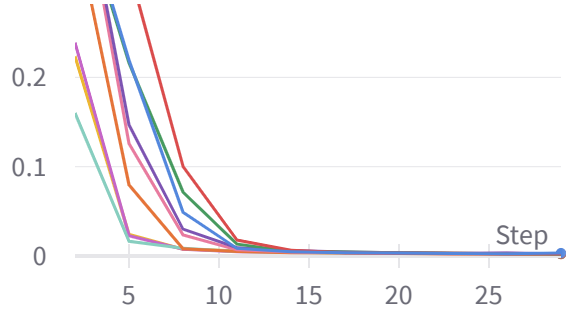
Plots



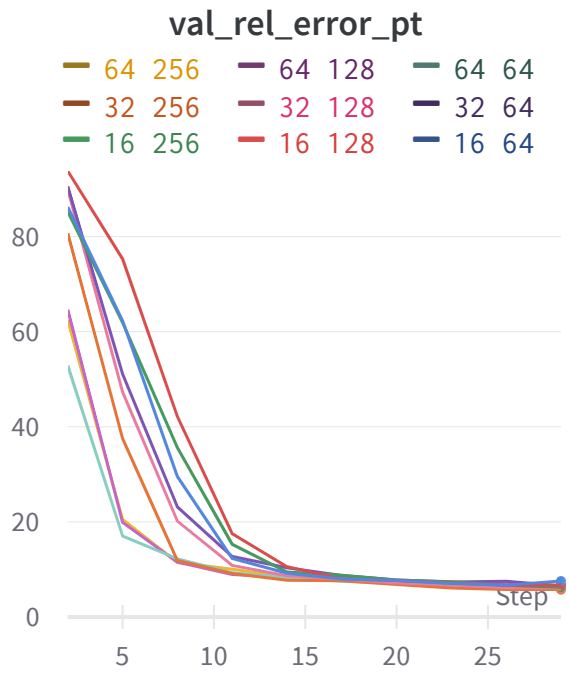
This is the plot for the training loss over the duration of the training time. Each curve represents a "dim"-"h_dim" pair of the grid search. It can be seen from the plot that all the curves converge to 0 over the training period. This indicates that the model has learned to make



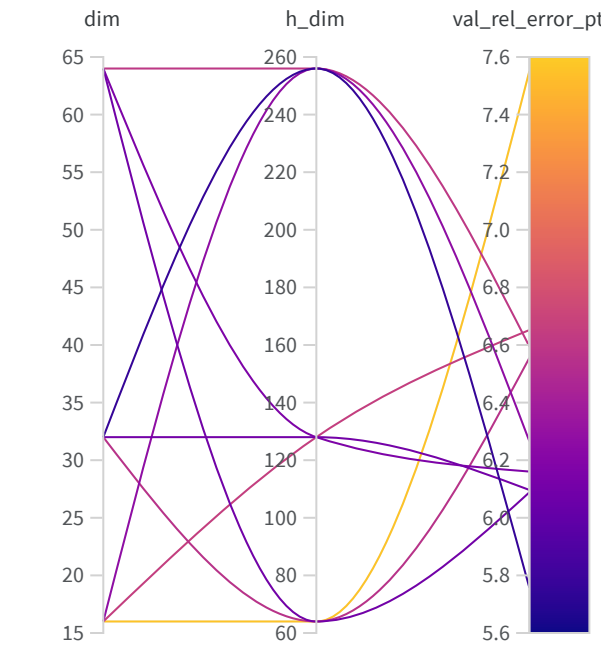
This is the plot for the validation loss over the duration of the training time. Each curve represents a "dim"-"h_dim" pair of the grid search. It can be seen



from the plot that all the curves converge to 0 over the training period. This indicates that the model has learned to make



This is the plot for the mean relative error (in percentage) for the validation set over the duration of the training time. Each curve represents a "dim"-"h_dim" pair of the grid search. It can be seen from the plot that all the curves converge to values in the range of 5-8 over



This plot shows which values of the mean relative error (in percentage) for the validation set for each pair of hyperparameter values "dim"-"h_dim". The pair minimizing this value is dim=32, h_dim=256, as mentioned above. This is because it is the lowest curve on the

Import panel

Add panel

Parameter importance with respect to

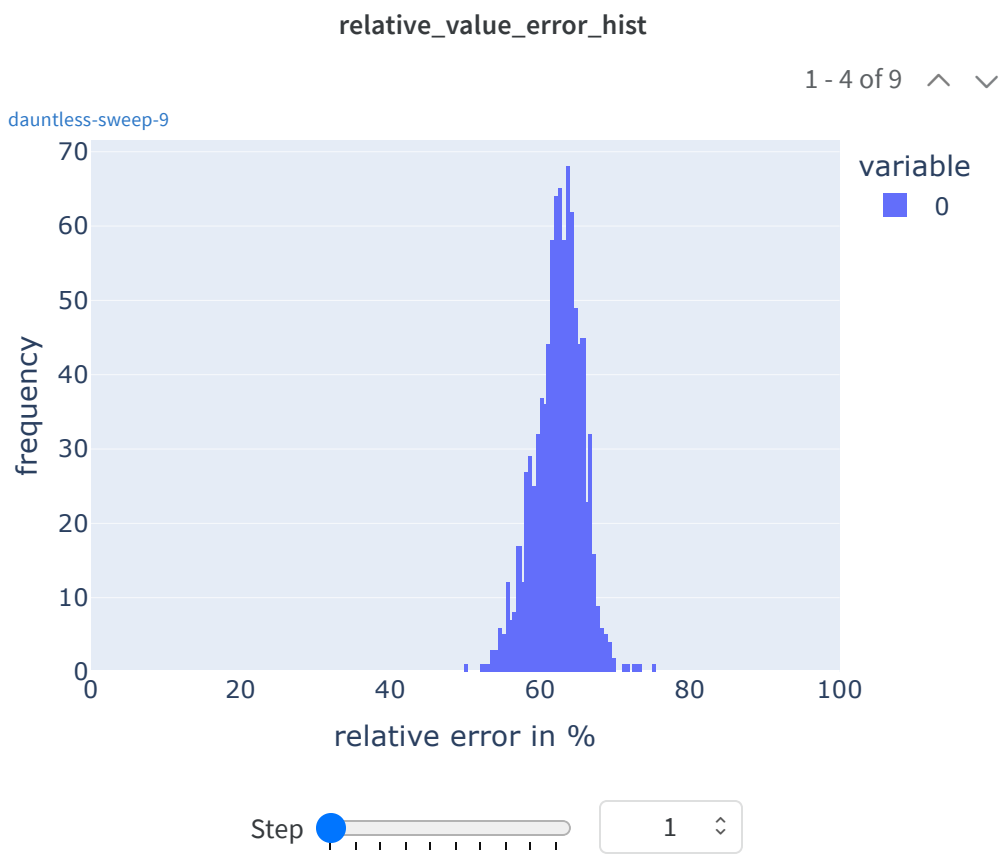
val_rel_error_pt

Search

Parameters

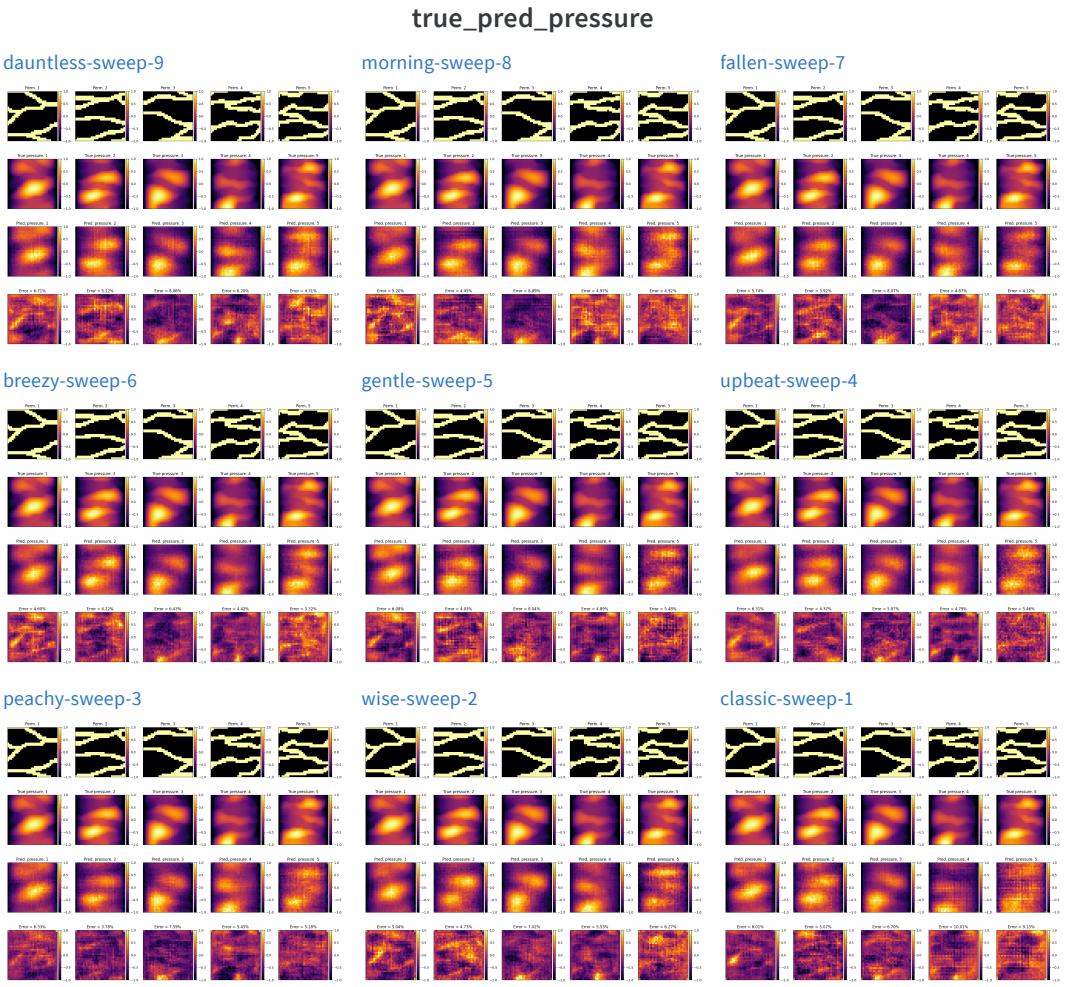
Config para…	Importance.	Correlation
dim		
h_dim		

From this figure, we can understand which hyperparamer has the most importance (i.e. influences the most) on the mean relative validation error. As can be seen, the parameter "dim", representing the number of feature channels in the first



The above histograms represent the occurrences of the relative validation error (in %) for each data point. The slider allows visualizing the evolution of the relative validation error over the training period. This can be done for all of the 9 runs, i.e. for each

pair of hyperparameters. These interactive figures allow us to clearly follow the evolution over the training period of the relative validation error. For all of the above depicted scenarios, it can be seen that the relative validation error strongly decreases as the model is learning more. In other words, we see a shift of the



The above plots help us visualize how our model is able to predict the pressure field. The 9 locations represent the 9 possible pairs of hyperparameters "dim" and "h_dim". For each pair of hyperparameters, 20 plots are generated with 4 rows and 5 columns. The first row shows the permeability field (model input), the second row shows the pressure field (model expected output), the third row shows the prediction of the pressure field (model prediction), and the last row shows the relative error (in %) of the actual pressure field and the model prediction. This slider allows seeing how these plots evolve over time. It can be seen that, after training (slider to the right), the model is able to

To conclude, the hyperparameter pair that minimizes the relative validation error is "dim"=32 and "h_dim"=256 as mentioned above. The observations earlier in the report can justify this. The associated best mean relative error is 5.757%.

Import panel Add panel

Import panel Add panel

Import panel Add panel

Import panel Add panel

```
# Import libraries
import os
import pprint
import numpy as np
import wandb
import torch
import torch.nn as nn
import time
from torchsummary import summary
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
from utils import *
from models import FCN
# Ensure reproducibility
torch.backends.cudnn.deterministic = True
seed_no = 108
np.random.seed(hash("improves reproducibility") % seed_no)
torch.manual_seed(hash("by removing stochasticity") % seed_no)
torch.cuda.manual_seed_all(hash("so runs are repeatable") % seed_no)
# Device configuration
device = torch.device("cuda" if torch.cuda.is_available() else "cpu"

class CAE(nn.Module):
    def __init__(self, dim, h_dim):
        super(CAE, self).__init__()
        self.dim = dim
        self.h_dim = h_dim

        # Encoder
        self.encoder = nn.Sequential(
```

```
            nn.Conv2d(1, self.dim, 4, 2, 1),
            nn.LeakyReLU(0.25),
            nn.BatchNorm2d(self.dim),

            nn.Conv2d(self.dim, 2 * self.dim, 4, 2, 1),
            nn.LeakyReLU(0.25),
            nn.BatchNorm2d(2 * self.dim),

            nn.Conv2d(2 * self.dim, 4 * self.dim, 4, 2, 1),
            nn.LeakyReLU(0.25),
            nn.BatchNorm2d(4 * self.dim),

            nn.Conv2d(4 * self.dim, 8 * self.dim, 4, 2, 1),
            nn.LeakyReLU(0.25),
            nn.BatchNorm2d(8 * self.dim)
        )

        # Bottleneck
        self.bottleneck = nn.Sequential(
            nn.Linear(self.dim * 32, self.h_dim),
            nn.LeakyReLU(0.25),
            nn.BatchNorm1d(self.h_dim),

            nn.Linear(self.h_dim, self.dim * 32),
            nn.LeakyReLU(0.25),
            nn.BatchNorm1d(self.dim * 32)

        )


        # Decoder
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(8 * self.dim, 4 * self.dim, 4, 2,
            nn.LeakyReLU(0.25),
            nn.BatchNorm2d(4 * self.dim),

            nn.ConvTranspose2d(4 * self.dim, 2 * self.dim, 4, 2,
            nn.LeakyReLU(0.25),
            nn.BatchNorm2d(2 * self.dim),

            nn.ConvTranspose2d(2 * self.dim, self.dim, 4, 2, 1),
```

```
                nn.LeakyReLU(0.25),
                nn.BatchNorm2d(self.dim),

                nn.ConvTranspose2d(self.dim, 1, 4, 2, 1),
                nn.LeakyReLU(0.25)
            )

    def forward(self, x):
        x = self.encoder(x)
        x = torch.reshape(x, (-1, self.dim * 32))
        x = self.bottleneck(x)
        x = torch.reshape(x, (-1, self.dim * 8, 2, 2))
        output = self.decoder(x)
        return output


# Loading the data
permeability = np.load("hw6_data/permeability.npy")
pressure = np.load("hw6_data/pressure.npy")

# Normalize the data between -1 and 1
permeability = ((permeability - permeability.min())/(permeability.ma
pressure = ((pressure - pressure.min())/(pressure.max()-pressure.mi

# Hyperparameters
num_epochs = 150
log_freq = int(0.1 * num_epochs)
n_train = int(0.8 * permeability.shape[0])

# Data loader
train_permeability = torch.from_numpy(permeability[:n_train, :, :])
train_pressure = torch.from_numpy(pressure[:n_train, :, :]).float()

val_permeability = torch.from_numpy(permeability[n_train:, :, :]).fl
val_pressure = torch.from_numpy(pressure[n_train:, :, :]).float().to

train_dataset = torch.utils.data.TensorDataset(train_permeability, t
test_dataset = torch.utils.data.TensorDataset(val_permeability, val_


#wandb hyperparameter dictionary
```

```
sweep_configuration = {
    "method": "grid",
    "name": "grid_search",
    "metric": {"goal": "minimize", "name": "val_loss"},
    "parameters":
    {
        "dim": {"values": [16.0, 32.0, 64.0]},
        "h_dim": {"values": [64.0, 128.0, 256.0]},
        "lr": {"values": [0.001]},
        "reg_param": {"values": [0.0]},
        "batch_size": {"values": [1024]}
     }
}
pprint.pprint(sweep_configuration)
project_name = "cme216_hw6"
group_name = "grid_search_hw6"
sweep_id = wandb.sweep(sweep_configuration, project=project_name)


t1 = time.time()
# ================================================================
# Training
# ================================================================
# Train the model
def train(config=None):
    # Initialize the new wandb run
    wandb.init(config=config, project=project_name, group=group_name
    config = wandb.config
    train_loader = torch.utils.data.DataLoader(dataset=train_dataset
                                               batch_size=config.batch
                                                  shuffle=True)
    total_step = len(train_loader)
    loss_list = []


    # Model, Loss, and Optimizer
    model = CAE(config.dim, config.h_dim).to(device)
    criterion = nn.MSELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=config.lr, w
    for epoch in range(num_epochs):
        for i, (train_x, train_y) in enumerate(train_loader):
```

```
            # Run the forward pass
            model.train()
            output = model(train_x.unsqueeze(1))
            loss = criterion(output, train_y.unsqueeze(1))
            loss_list.append(loss.item())
            # Backprop and perform Adam optimisation
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        if (epoch+1) % log_freq == 0:
            # Calculate the validation loss
            model.eval()
            with torch.no_grad():
                val_pressure_pred = model(val_permeability.unsqueeze
                val_loss = criterion(val_pressure_pred, val_pressure

            plot_perm_and_temp(val_permeability.detach().cpu().numpy
                               val_pressure.detach().cpu().numpy()
                               val_pressure_pred.detach().cpu().nu
            plot_rel_val_err(val_pressure.detach().cpu().numpy().squ
                             val_pressure_pred.detach().cpu().numpy
            diff_ = (val_pressure_pred - val_pressure.unsqueeze(1))
            diff_vec = np.reshape(diff_, (diff_.shape[0], -1))
            val_l2_pt_error = np.mean(np.linalg.norm(diff_vec, axis=
            val_indiv_rel_error = np.linalg.norm(diff_vec, axis=1) /
            #table = wandb.Table(val_indiv_rel_error, "val_indiv_re

            wandb.log({"val_loss": val_loss.item(), "train_loss": lo
            print (f"Epoch [{epoch+1}/{num_epochs}], Step [{i+1}/{to
                    Training Loss: {loss.item():.4f}, Validation Los
                    Val. error (in %) = {val_l2_pt_error:.2f}%")

    # Save the model checkpoint (optional)
    save_path = os.path.join(wandb.run.dir, "model.ckpt")
    torch.save(model.state_dict(), save_path)

wandb.agent(sweep_id, train)
t2 = time.time()
print(f"Total time taken: {t2-t1}")
wandb.finish()
```