Autonomous Vision Group
Prof. Dr.-Ing. Andreas Geiger

Eberhard Karls Universität Tübingen

# Computer Vision

## Exercise 4 – Learning in GMs and 3D Reconstruction

## 4.1 Pen and Paper

### Gradient of Negative Conditional Log-Likelihood

**a)** For a Deep Structured Model

$$p(\mathcal{Y}|\mathcal{X}, \mathbf{w}) = \frac{1}{Z(\mathcal{X}, \mathbf{w})} \exp\{\psi(\mathcal{X}, \mathcal{Y}, \mathbf{w})\}$$

the negative conditional log likelihood is given as:

$$\mathcal{L}(\mathbf{w}) = -\sum_{n=1}^{N} \left[ \psi(\mathcal{X}^n, \mathcal{Y}^n, \mathbf{w}) - \log \sum_{\mathcal{Y}} \exp\{\psi(\mathcal{X}^n, \mathcal{Y}, \mathbf{w})\} \right]$$

Derive this equation so that the parameters $\mathbf{w}$ of the model can be optimized via gradient descent!

$$\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = ?$$

### Inference Unrolling

**a)** For the following python code, draw the unrolled compute graph. This means that starting with the hidden state and the input elements, all the way to the final prediction (pred), draw the compute graph (without loops). Additions, concatenations etc. should be represented by nodes, but the activation function you can just summarize as $\sigma(x)$. You can ignore the loss, that is just there for clarity about how the prediction might be used.

```python
import torch
from torch import nn
from some_package import dataset

class SomeNetwork(nn.Module):
  # don't expect this model to work or do something useful
  # it is just a fairly random example for educational purposes
  def __init__(self, input_size, output_size, hidden_size):
    # Linear Layer: y = xW + b
    self.input_to_hidden = nn.Linear(input_size + hidden_size, hidden_size)
    self.input_to_output = nn.Linear(input_size + hidden_size, output_size)


  def forward(self, input_data, hidden_state):
    concat_vals = torch.cat([input_data, hidden_state], dim=1)
    hidden = self.input_to_hidden(concat_vals)
    hidden_activated = torch.sigmoid(hidden)
    output = self.input_to_output(concat_vals)
    return output, hidden_activated


if __name__ == "__main__":
  input_size = 10
  output_size = 10
  hidden_dim = 20
  num_layers = 3

```

```python
        loss_fn = torch.nn.L1Loss()
        my_net = SomeNetwork(input_size,output_size,hidden_dim)
        for ds_element, target in dataset:
          hidden_state = torch.zeros(1,hidden_dim)
          for layer_idx in range(num_layers):
            pred, hidden_state = my_net(ds_element[layer_idx], hidden_state)
          loss = loss_fn(target, pred)


```

### Surface Integration

**a)** Through some previous method you have obtained surface gradients (see Table 1). Reconstruct the surface from the following surface gradients (hint: you do not need to use the variational approach mentioned in the lecture, there is a much simpler solution): For simplicity, we only look at a 1D example (i.e. you are asked to reconstruct a line.) Please draw a graph of the resulting reconstruction. Note that $\frac{dz}{dx}(x_i) = z(x_{i+1}) - z(x_i)$.

Table 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $x_i$ | 0.0 | 1.0 | 2.0 | 3.0 | 4.0 | 5.0 | 6.0 |
| $\frac{dz}{dx}(x_i)$ | 1.5 | 2.0 | -1.0 | 0.0 | 1.0 | 1.0 | 0.0 |
| $z(x_i)$ | | | | 10.0 | | | |

### Volumetric Fusion Formulation

**a)** Please show that weighted averaging

$$D(\mathbf{x}) = \frac{\sum_i w_i(\mathbf{x})\, d_i(\mathbf{x})}{\sum_i w_i(\mathbf{x})}$$

is the solution to the following weighted least squares problem

$$D^* = \underset{D}{\operatorname{argmin}} \sum_i w_i \, (d_i - D)^2$$

**b)** SDF (Signed Distance Function) Fusion calculates the weighted average per voxel:

$$D(\mathbf{x}) = \frac{\sum_i w_i(\mathbf{x})\, d_i(\mathbf{x})}{\sum_i w_i(\mathbf{x})}$$
$$W(\mathbf{x}) = \sum_i w_i(\mathbf{x})$$

From this, derive the incremental solution:

$$D_{i+1}(\mathbf{x}) = \frac{W_i(\mathbf{x})\, D_i(\mathbf{x}) + w_{i+1}(\mathbf{x})\, d_{i+1}(\mathbf{x})}{W_i(\mathbf{x}) + w_{i+1}(\mathbf{x})}$$
$$W_{i+1}(\mathbf{x}) = W_i(\mathbf{x}) + w_{i+1}(\mathbf{x})$$

### Drawing the p/q Reflectance Map for a Fixed R

**a)** Given a fixed and known light direction $\mathbf{s} = [0.0, 0.0, 1.0]^{\mathrm{T}}$ draw the Iso-Brightness Contours for the different Reflectance values $R = 0.5, 0.9, 0.95$ in $p, q$-space.

**b)** Derive the formulas for the Stereographic Mapping ($f, g$-space) as introduced in the lecture.

**c)** Now draw the Iso-Brightness Contours in ($f, g$-space). What shape do you observe after converting your previously drawn Iso-Brightness contours from gradient space to f/g space?

**Marching Cubes**

**a)** You are provided with the 2D grid in Figure 1.

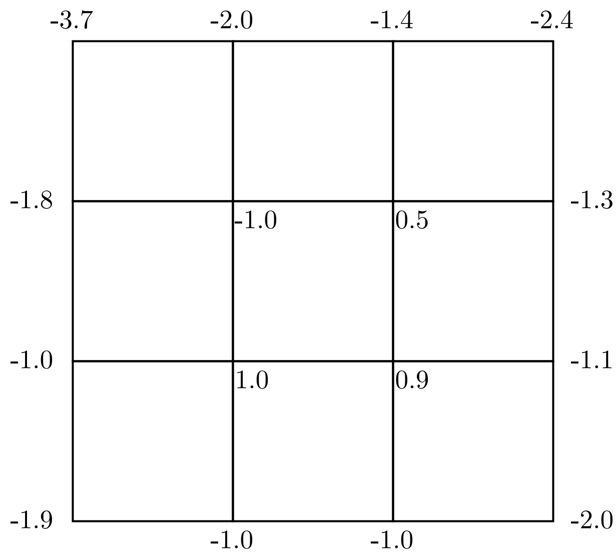


Figure 1: 2D grid with values for SDF. $D_{i,j}$ refers to the value of the SDF to the grid cell center in the i-th row, j-th column of the grid.

Your task is to apply the "Marching Cubes" algorithm as discussed in the lecture in the 2D case.

**i)** Find the points inside / outside the mesh.

**ii)** Compute the intersection points.

**iii)** Draw the outer bound (i.e. connect your computed intersection points) of the mesh into Figure 1.

## 4.2 Coding Exercises

To get started with the coding exercise you need to perform the following steps.

**a)** Go to the folder "data" with the file "get_data.sh". Execute this script to download the necessary files for the exercise.

**b)** Go back up one folder where the ".ipynb" files are. ("cd ..")

**c)** **Create a new conda environment: "conda env create -f environment.yml"** - otherwise the exercise might not work correctly.

**d)** Activate the environment by running "conda activate lecturecv-ex04".

**e)** Start jupyter-notebook by running "jupyter-notebook".

Installation instructions are also provided in the file "install_guide.txt".
If you would like to run the exercises on Google Colab, please use '!pip install k3d' and '!pip install trimesh' before running the actual code in "marching_cubes.ipynb". For the other exercise this is not necessary. After that, you can start working on following the exercises:

**a)** "photometric_stereo.ipynb": You will reconstruct the normals and albedo from images taken with light sources from different locations.

b) "marching_cubes.ipynb": You will learn all about the marching cubes algorithm and reconstruct a mesh using a voxel grid of SDF values.

Happy coding!