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

Eberhard Karls Universität Tübingen

Computer Vision

Exercise 4 – Learning in GMs and 3D Reconstruction

1 Pen and Paper

1.1 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!

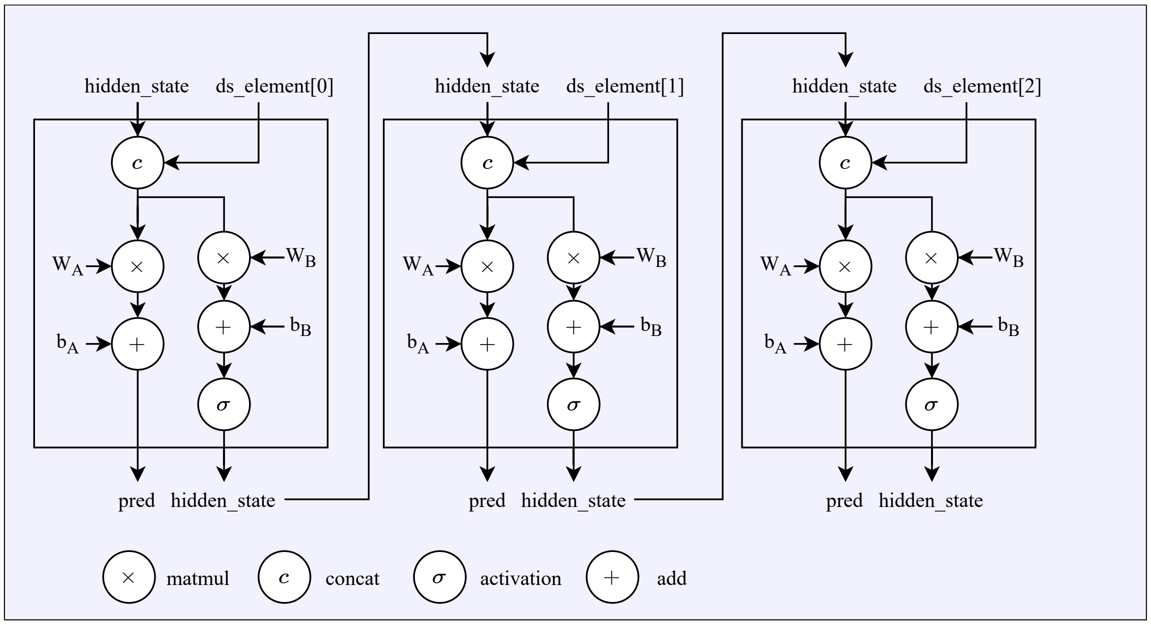
$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = ?$$

$$\begin{aligned}\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) &= -\sum_{n=1}^{N}\left[\nabla_{\mathbf{w}}\psi(\mathcal{X}^n,\mathcal{Y}^n,\mathbf{w}) - \frac{\sum_{\mathcal{Y}}\exp\{\psi(\mathcal{X}^n,\mathcal{Y},\mathbf{w})\}\nabla_{\mathbf{w}}\psi(\mathcal{X}^n,\mathcal{Y},\mathbf{w})}{\sum_{\mathcal{Y}}\exp\{\psi(\mathcal{X}^n,\mathcal{Y},\mathbf{w})\}}\right]\\ &= -\sum_{n=1}^{N}\left[\nabla_{\mathbf{w}}\psi(\mathcal{X}^n,\mathcal{Y}^n,\mathbf{w}) - \sum_{\mathcal{Y}}\frac{\exp\{\psi(\mathcal{X}^n,\mathcal{Y},\mathbf{w})\}}{\sum_{\mathcal{Y}'}\exp\{\psi(\mathcal{X}^n,\mathcal{Y}',\mathbf{w})\}}\nabla_{\mathbf{w}}\psi(\mathcal{X}^n,\mathcal{Y},\mathbf{w})\right]\\ &= -\sum_{n=1}^{N}\left[\nabla_{\mathbf{w}}\psi(\mathcal{X}^n,\mathcal{Y}^n,\mathbf{w}) - \sum_{\mathcal{Y}}p(\mathcal{Y}|\mathcal{X}^n,\mathbf{w})\nabla_{\mathbf{w}}\psi(\mathcal{X}^n,\mathcal{Y},\mathbf{w})\right]\end{aligned}$$

1.2 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

  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)


```



1.3 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 $\frac{dz}{dx}(x_i) = z(x_{i+1}) - z(x_i)$.

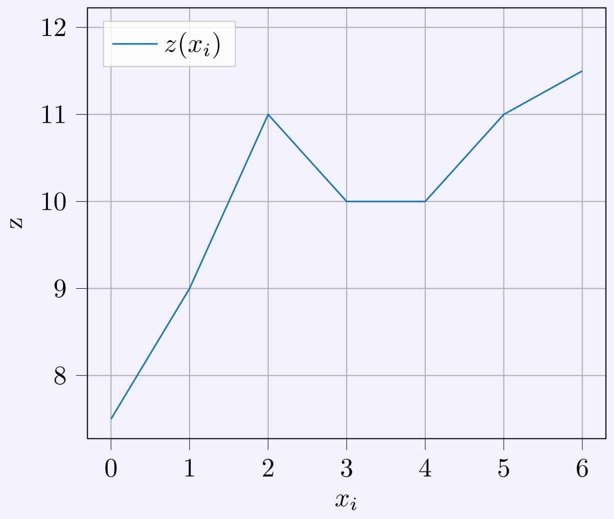
Table 1

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	0.5	0.0
$z(x_i)$				10.0			

Using $z(x_{i+i}) = \frac{dz}{dx}(x_i) + z(x_i)$ when moving to the right and $z(x_i) = z(x_{i+i}) - \frac{dz}{dx}(x_i)$ when moving to the left we obtain:

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	0.5	0.0
$z(x_i)$	7.5	9.0	11.0	10.0	10.0	11.0	11.5

We end up with the following graph:



1.4 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

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

Deriving the weighted squared error WSE w.r.t. D we show that:

$$\text{WSE} = \sum_i w_i \left(d_i - D\right)^2$$

$$\frac{\partial \text{WSE}}{\partial D} = \sum_i 2w_i \left(d_i - D^*\right) = 0$$

$$\sum_i -w_i d_i + \sum_i w_i D = 0$$

$$D = \frac{\sum_i w_i d_i}{\sum_i w_i}$$

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:

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

$$D(\mathbf{x}) = \frac{\sum_i w_i(\mathbf{x})\, d_i(\mathbf{x})}{\sum_i w_i(\mathbf{x})}$$
$$D_i(\mathbf{x}) = \frac{1}{W_i(\mathbf{x})}\left(w_i(\mathbf{x})d_i(\mathbf{x}) + \sum_{n=1}^{i-1} w_n(\mathbf{x})d_n(\mathbf{x})\right)$$
$$D_i(\mathbf{x}) = \frac{1}{W_i(\mathbf{x})}\left(w_i(\mathbf{x})d_i(\mathbf{x}) + W_{i-1}(\mathbf{x})D_{i-1}(\mathbf{x})\right)$$

shift the index by 1:

$$\begin{aligned} D_{i+1}(\mathbf{x}) &= \frac{1}{W_{i+1}(\mathbf{x})}\left(w_{i+1}(\mathbf{x})d_{i+1}(\mathbf{x}) + W_i(\mathbf{x})D_i(\mathbf{x})\right) \\ &= \frac{w_{i+1}(\mathbf{x})d_{i+1}(\mathbf{x}) + W_i(\mathbf{x})D_i(\mathbf{x})}{W_{i+1}(\mathbf{x})} \\ &= \frac{W_i(\mathbf{x})D_i(\mathbf{x}) + w_{i+1}(\mathbf{x})d_{i+1}(\mathbf{x})}{W_i(\mathbf{x}) + w_{i+1}(\mathbf{x})} \end{aligned}$$

1.5 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 $R = 0.5, 0.8, 0.95$ 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?

1.5.1 (a)

We know from the lecture that

$$\mathbf{n} = \frac{(p, q, 1)^\top}{\sqrt{p^2 + q^2 + 1}}$$

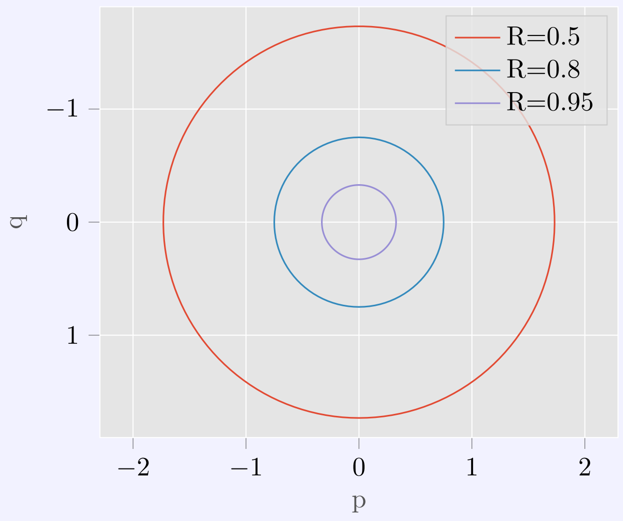
and that

$$R(\mathbf{n}) = \mathbf{n}^\top \mathbf{s} = \frac{p\, s_x + q\, s_y + s_z}{\sqrt{p^2 + q^2 + 1}} = R(p, q)$$

Inserting $\mathbf{s} = [0.0, 0.0, 1.0]^{\mathrm{T}}$ leaves us with

$$\frac{1}{\sqrt{p^2+q^2+1}} = R(p,q)$$

which is represents a circle with radius $\sqrt{\frac{1}{R^2}-1}$ Inserting each value for R one at a time, we can easily draw the corresponding Iso-Brightness contours:



In the plot, $R = 0.5$ corresponds to circle with radius 1.732, $R = 0.8$ corresponds to radius 0.75 and $R = 0.95$ to radius 0.328.

1.5.2 (b)

Using the following graphic we can see that triangles AED and ABC are similar. Further, we know that:

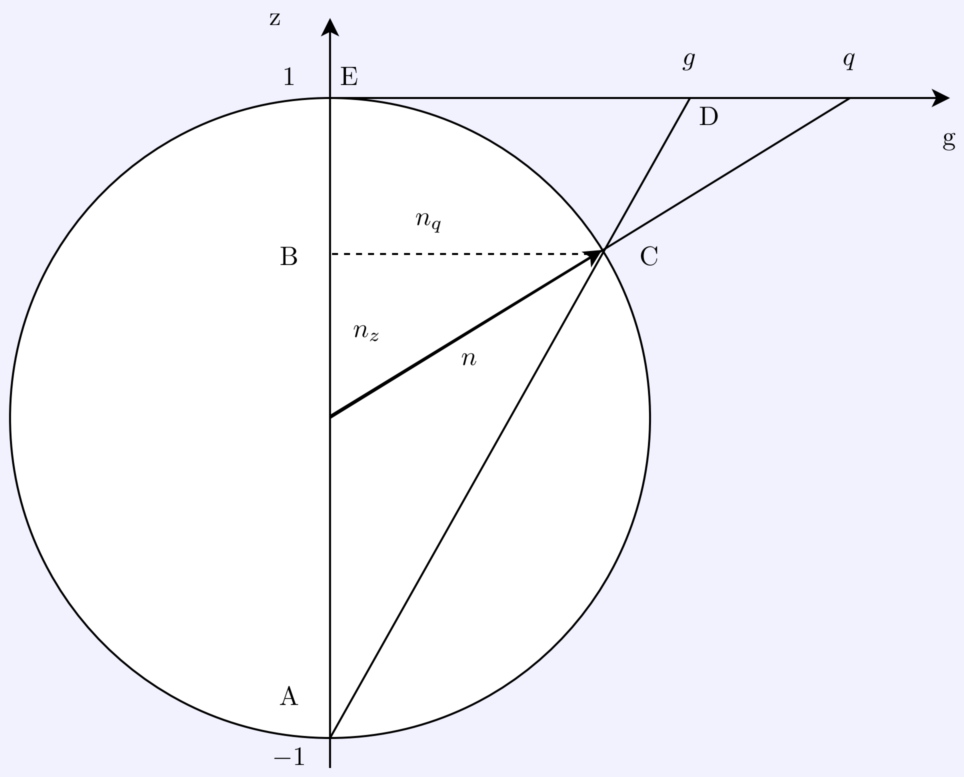
$$\mathbf{n} = (n_p, n_q, n_z)^\top = \frac{(p,q,1)^\top}{\sqrt{p^2+q^2+1}}$$

Using this in combination with the similarity yields:

$$\frac{g}{n_q} = \frac{2}{1+n_z}$$

$$g = \frac{2n_q}{1+n_z} = \frac{\frac{2q}{\sqrt{p^2+q^2+1}}}{1+\frac{1}{\sqrt{p^2+q^2+1}}} = \frac{2q}{1+\sqrt{p^2+q^2+1}}$$

The derivation of f is completely analogous.



1.5.3 (c)

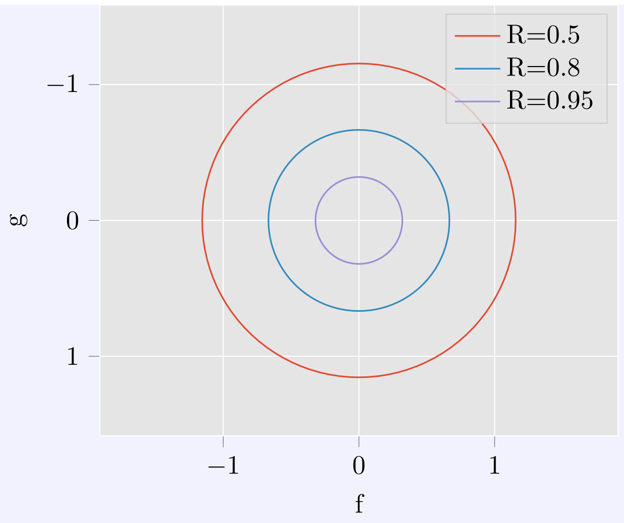
The converted Iso-Brightness Contours are of course also circles. But note how the relative size of the circles with respect to the other circles has changed due to the conversion. Since

$$f = \frac{2p}{1+\sqrt{p^2+q^2+1}}, g = \frac{2q}{1+\sqrt{p^2+q^2+1}},$$

we have

$$f^2+g^2 = \frac{4(\frac{1}{R^2}-1)}{(1+\frac{1}{R})^2} = \frac{4(1-R^2)}{(1+R)^2}$$

$R = 0.5$ corresponds to circle with radius 1.155, $R = 0.8$ corresponds to radius $\frac{2}{3}$ and $R = 0.95$ to radius 0.320.



1.6 Marching Cubes

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

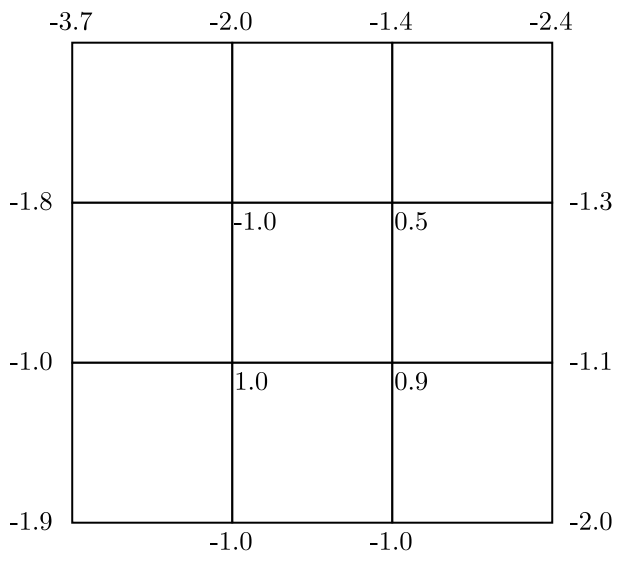


Figure 1: 2D grid with values for SDF. $D_{i,j}$ refers to the value of the SDF to the grid cell center in the i-th row, j-th column of the grid.

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.

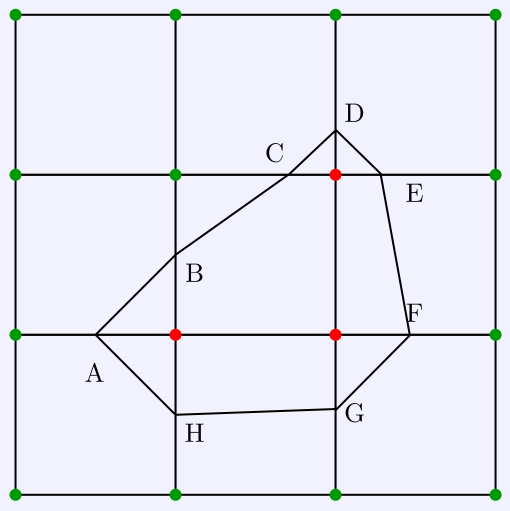
iii) Draw the outer bound (i.e. connect your computed intersection points) of the mesh into Figure 1.

First, we determine the points lying inside and outside the final mesh by looking for sign changes. (See the red and green points.) Then we compute the intersection points A to H, using interpolation between the values at the red and green points. From the lecture we know:

$$x = \frac{D_{i,j}}{D_{i,j} + D_{i,j+1}}$$

Applied to the intersection A through H:

$$\begin{aligned}
\text{A:} \quad x_A &= \frac{D_{2,1}}{D_{2,1} - D_{2,0}} = 0.5 \\
\text{B:} \quad x_B &= \frac{D_{2,1}}{D_{2,1} - D_{1,1}} = 0.5 \\
\text{C:} \quad x_C &= \frac{D_{1,2}}{D_{1,2} - D_{1,1}} = 0.333 \\
\text{D:} \quad x_D &= \frac{D_{1,2}}{D_{1,2} - D_{0,2}} = 0.263 \\
\text{E:} \quad x_E &= \frac{D_{1,2}}{D_{1,2} - D_{1,3}} = 0.278 \\
\text{F:} \quad x_F &= \frac{D_{2,2}}{D_{2,2} - D_{2,3}} = 0.45 \\
\text{G:} \quad x_G &= \frac{D_{2,2}}{D_{2,2} - D_{3,2}} = 0.474 \\
\text{H:} \quad x_H &= \frac{D_{2,1}}{D_{2,1} - D_{3,1}} = 0.5
\end{aligned}$$



2 Coding Exercises

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

- Go to the folder “data” with the file “get_data.sh”. Execute this script to download the necessary files for the exercise.
- Go back up one folder where the “.ipynb” files are. (”cd ..”)

- **Create a new conda environment: "conda env create -f environment.yml"** - otherwise the exercise might not work correctly.
- Activate the environment by running "conda activate lecturecv-ex04".
- 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!

Please check "photometric_stereo_solution.ipynb" and "marching_cubes_solution.ipynb" for the solutions to the coding exercises.