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

Eberhard Karls
Universität
Tübingen

# Computer Vision

## Exercise 5 – Neural Implicit Models and Recognition

## 1 Pen and Paper

### 1.1 2D Occupancy Network

**a)** Derive the decision boundary for a linear layer $L_{\mathbf{W},b}$ with weight $\mathbf{W} = (w_1, w_2)$ and bias $b$, i.e. where $L_{\mathbf{W},b}(\mathbf{x}) = 0$

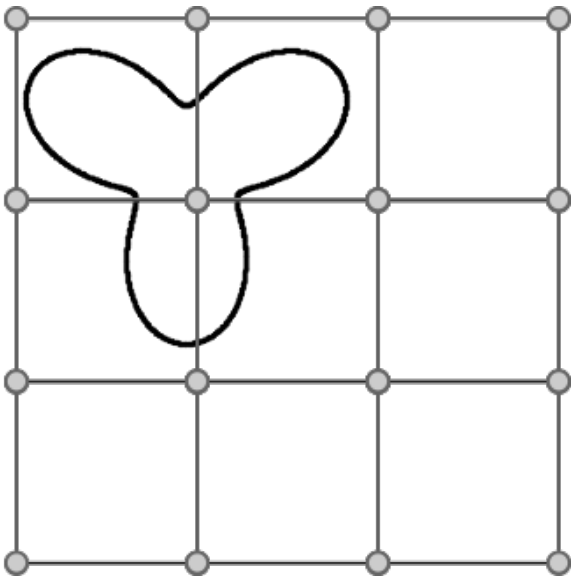
**b)** Draw the decision boundary for

**i)** $\mathbf{W} = (-2, 5)$, $b = -1$

**ii)** $\mathbf{W} = (5, 10)$, $b = -3$

**c)** Now consider a 2-layer fully-connected ReLU-network with weights $\mathbf{W}_1 = \begin{pmatrix} 3 & -2 \\ -1 & 4 \end{pmatrix}$, $\mathbf{b}_1 = \begin{pmatrix} 2 \\ -3 \end{pmatrix}$, and $\mathbf{W}_2 = (1, -2)$, $b_2 = -1$. Let $\sigma(\mathbf{x}) = \begin{pmatrix} \max(0, x_1) \\ \max(0, x_2) \end{pmatrix}$ denote the ReLU activation function. Derive and draw the decision boundary for this network. You can use an online tool, e.g., `https://www.desmos.com/calculator` to plot the decision boundary.

**d)** So far, we have learned how to extract the decision boundary from a trained network. Next, we want to extract a mesh from this implicitly defined boundary. The figure below shows the decision boundary for some classifier. Apply the MISE algorithm manually by coloring occupied points in red and free points in blue and refining the grid where needed. See Lecture 9, Slide 13 for an example. Refine the grid for 3 steps. Lastly, manually execute the marching squares algorithm, the 2D variant of the marching cubes algorithm, and draw the extracted mesh into the last refined grid. Assume that the classifier returns either 0 (free space) or 1 (occupied).



## 1.2 Differentiable Volumetric Rendering

Let the occupancy network $f_{\theta_f}$ in DVR be parametrized by a linear layer $L_{\mathbf{W}_f, b_f}$ followed by a sigmoid function $\sigma$, i.e. $f_{\theta_f}(\mathbf{p}) = \sigma(L_{\mathbf{W}_f, b_f})$. This models the probability that a point $\mathbf{p}$ is inside the object. The surface is implicitly defined as the set of points where $f_{\theta_f} = \tau$, where $\tau = 0.5$. Further, let the texture field $t_{\theta_t}$ be defined similarly, i.e. $t_{\theta_t}(\mathbf{p}) = \sigma(L_{\mathbf{W}_t, b_t})$. We choose the weights: $\mathbf{W}_f = (-3, 2)$, $b_f = 2$, and $\mathbf{W}_t = (1, 4)$, $b_t = -1$. Note that in this case the texture field only predicts a single scalar that corresponds to the intensity of the pixel to simplify computations.

**a)** We shoot a ray from the origin in an angle of 45°, i.e. $\mathbf{r} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + d \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}$. Estimate the surface intersection point $\hat{\mathbf{p}}$ by performing a single update step with the secant method starting in the interval $[d_0 = 1, d_1 = 2]$.

**b)** Compute the predicted texture field, i.e. intensity, at $\hat{\mathbf{p}}$.

**c)** As loss $\mathcal{L}$ we choose the squared L2-distance between the predicted intensity $\hat{I}$ and the ground truth value $I$. Let us assume that $I = 0.7$. The gradient wrt. all parameters $\theta = \theta_f \cup \theta_t$ is

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \theta} &= \frac{\partial \mathcal{L}}{\partial \hat{I}} \frac{\partial \hat{I}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial \hat{I}} \frac{dt_{\theta_t}(\hat{\mathbf{p}})}{d\theta} \\ &= \frac{\partial \mathcal{L}}{\partial \hat{I}} \left[ \frac{\partial t_{\theta_t}(\hat{\mathbf{p}})}{\partial \theta} + \frac{\partial t_{\theta_t}(\hat{\mathbf{p}})}{\partial \hat{\mathbf{p}}} \frac{\partial \hat{\mathbf{p}}}{\partial \theta} \right] \end{aligned}$$

Therefore the gradients wrt. the parameters $f_{\theta_f}$, i.e. $\mathbf{W}_f$ and $b_f$ are given by:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_f} = \frac{\partial \mathcal{L}}{\partial \hat{I}} \left[ \frac{\partial t_{\theta_t}(\hat{\mathbf{p}})}{\partial \mathbf{W}_f} + \frac{\partial t_{\theta_t}(\hat{\mathbf{p}})}{\partial \hat{\mathbf{p}}} \frac{\partial \hat{\mathbf{p}}}{\partial \mathbf{W}_f} \right] \tag{1}$$

$$\frac{\partial \mathcal{L}}{\partial b_f} = \frac{\partial \mathcal{L}}{\partial \hat{I}} \left[ \frac{\partial t_{\theta_t}(\hat{\mathbf{p}})}{\partial b_f} + \frac{\partial t_{\theta_t}(\hat{\mathbf{p}})}{\partial \hat{\mathbf{p}}} \frac{\partial \hat{\mathbf{p}}}{\partial b_f} \right] \tag{2}$$

Start by computing the following components of Eq. (1),(2) $\frac{\partial \mathcal{L}}{\partial \hat{I}}$, $\frac{\partial t_{\theta_t}(\hat{\mathbf{p}})}{\partial \mathbf{W}_f}$, $\frac{\partial t_{\theta_t}(\hat{\mathbf{p}})}{\partial b_f}$, $\frac{\partial t_{\theta_t}(\hat{\mathbf{p}})}{\partial \hat{\mathbf{p}}}$.

**d)** To compute the gradients for $\mathbf{W}_f$ and $b_f$ the only components missing for Eq. (1),(2) are $\frac{\partial \hat{\mathbf{p}}}{\partial \mathbf{W}_f}$ and $\frac{\partial \hat{\mathbf{p}}}{\partial b_f}$, respectively. However, we cannot directly compute them because $\hat{\mathbf{p}}$ is defined only implicitly. Use implicit differentiation on $f_{\theta_f} = \tau$, where $\tau = 0.5$, to obtain $\frac{\partial \hat{\mathbf{p}}}{\partial \mathbf{W}_f}$, $\frac{\partial \hat{\mathbf{p}}}{\partial b_f}$. With these results, compute the gradients for $\mathbf{W}_f$ and $b_f$.

**Hint**: Write $\hat{\mathbf{p}}$ as the ray from the origin $\hat{\mathbf{p}} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \hat{d} \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}$ to link it to the depth prediction $\hat{d}$. Since the depth is predicted by $f_{\theta_f}$, this will allow you to formulate $\frac{\partial \hat{\mathbf{p}}}{\partial \mathbf{W}_f}$, $\frac{\partial \hat{\mathbf{p}}}{\partial b_f}$ in terms of $\frac{\partial f_{\theta_f}(\hat{\mathbf{p}})}{\partial \hat{\mathbf{p}}}$ and $\frac{\partial f_{\theta_f}(\hat{\mathbf{p}})}{\partial \mathbf{W}_f}$ and $\frac{\partial f_{\theta_f}(\hat{\mathbf{p}})}{\partial b_f}$, respectively. For computing the result this means you also need to compute $\frac{\partial f_{\theta_f}(\hat{\mathbf{p}})}{\partial \hat{\mathbf{p}}}$, $\frac{\partial f_{\theta_f}(\hat{\mathbf{p}})}{\partial \mathbf{W}_f}$ and $\frac{\partial f_{\theta_f}(\hat{\mathbf{p}})}{\partial b_f}$.

## 1.3 Neural Network Layers

**a)** Calculate the number of parameters $\#P$ and the receptive field size $S_r$ for the following layers

**i)** Linear layer with a $32 \times 32$ flattened grayscale image as input, output dimension $D_{out} = 256$ and per-output bias.

**ii)** Convolution with filter size $K = 5$, $C_{in} = 16$ input and $C_{out} = 48$ output channels, padding $P = 2$, stride $S = 1$ and bias

**iii)** Two convolutions, both with filter size $K = 3$, $C_{in} = 64$ input and $C_{out} = 64$ output channels, padding $P = 1$, dilation $D = 1$, stride $S = 3$, and bias.

**iv)** Two convolutions:

- First layer: filter size $K_1 = 5$, $C_{in,1} = 24$ input and $C_{out,1} = 64$ output channels, dilation $D_1 = 1$, padding $P_1 = 2$, stride $S_1 = 1$, and bias
- Second layer: filter size $K_2 = 3$, $C_{in,2} = 64$ input and $C_{out,2} = 128$ output channels, dilation $D_2 = 3$, padding $P_2 = 3$, stride $S_2 = 1$, and bias

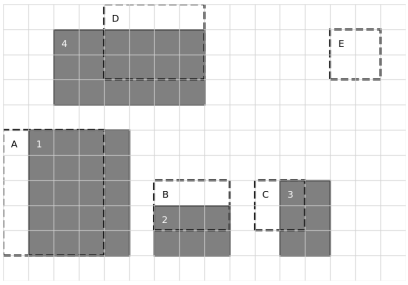
**v)** Max pooling with size 3

**b)** Complete the following table for softmax classification with Cross Entropy Loss. Each row is a different sample of a minibatch and the values of the ground truth class are marked in red. Which samples are classified correctly? Which samples are classified incorrectly? Which samples contribute most strongly to the parameter update of the model?

| Predicted scores $\mathbf{s}$ | softmax($\mathbf{s}$) | CE loss |
|---|---|---|
| $(+3, +1, -2, +0)^T$ | | |
| $(-1, +2, +3, -2)^T$ | | |
| $(+0, +0, +0, +0)^T$ | | |
| $(+4, +1, -2, +3)^T$ | | |

### 1.4 Detection Metrics

Consider the following image where the three gray shaded boxes (1,2,3,4) correspond to the true objects and the five dashed boxes (A,B,C,D,E) correspond to detections of an object detector:




**a)** Calculate the IoU (Intersection-over-Union) metric for the pairs (A, 1), (B, 2), (C, 3), and (D,4). Assuming a detection threshold of $\tau = 0.5$, which of the three objects have been correctly detected?

**b)** Calculate the number of true positives (TP), false positives (FP) and false negatives (FN) assuming the same detection threshold as in the previous question ($\tau = 0.5$).

**c)** Calculate precision and recall at a detection threshold of $\tau = 0.3$.

## 2 Coding Exercises

The following two exercises will be coding exercises. They are contained in the jupyter notebooks `code/implicit.ipynb` and `code/recognition.ipynb`. The notebooks are self-contained but you can also use this document as guidance. If you are stuck, you can find Hints in the notebook itself which are written upside-down.

### 2.1 Occupancy Network

In this exercise, you will train an occupancy network to represent a single 3D object. More specifically, you will start from the point cloud of a chair together with its occupancy values, see Fig. 1. Then, you will train an occupancy network to encode the shape and reconstruct the mesh from the trained network. The code is provided in `code/implicit.ipynb`. Make sure to copy the data for this task `code/data/points.npz` into your working directory to a folder called `data` if you use Google Colab.

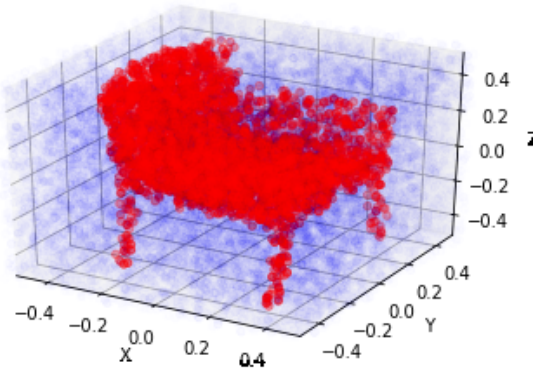



Figure 1: **3D Point Cloud of a Chair.** Red points belong to the object while blue points represent free space.

**a)** Prepare your data by implementing the function `get_train_val_split`.

**b)** Complete the class `OccNet` such that you obtain a network with 4 hidden layers of hidden dimension 64. The network takes a batch of 3D locations as input and predicts a single occupancy value for each point.

**c)** Complete the training loop, i.e. the function `train_model`. Do the curves for training and validation loss look good? Or is there under / overfitting?

**d)** Load the trained model and predict the occupancy values for all points on a 3D grid. You can use the `test_loader` to load the grid points in batches.

## 2.2 Person Detector

In this exercise, you will develop your own person detector. You will extract your training data using images of pedestrians with bounding box labels, see Fig. 2. Then, you train two simple classifiers on the feature representation of the training data. Lastly, you use the trained classifiers to detect pedestrians in new images using a sliding window approach. The code is provided in `code/recognition.ipynb`. Extract the dataset in `code/data/PennFudanPed.zip` using `unzip code/data/PennFudanPed.zip -d code/data/`.

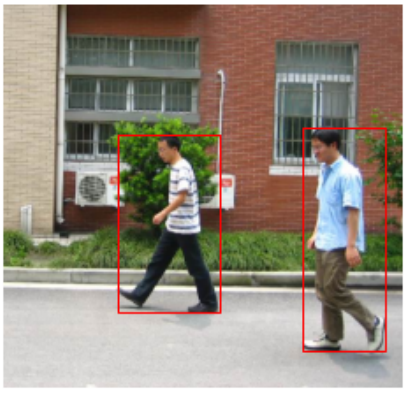


Figure 2: **Person Detector.** Example image of pedestrians with bounding box labels.

**a)** To mine negative training samples, implement the function `get_random_box`. The function should return the (left, upper, right, lower) pixel coordinates of a box of a given size at a random location in the image.

**b)** To further improve the accuracy our classifier we can add extra difficult negative samples to our training data. We construct hard negative examples by adding a small offset to the ground truth boxes. Thereby, the negative examples are close to positive detections. Implement the function `add_offset_to_box`. This function takes a box and an image size as input and returns a box that is slightly translated in the image.

**c)** Implement the function `img_to_hog_patches` that extracts image patches of a given size and converts these patches to HOG features. Of course, you should use the provided helper functions, e.g. to get the HOG features.