Probabilistic Graphical Models
Exercise 1: Image Processing

General note: The purpose of this exercise is for you to get hands-on experience with graphical models for image processing. You should work on the exercises in teams of two or at most three. At least one member of each team should be experienced in programming. Do not work alone, please find yourself a partner, and discuss the steps along the way.

The exercise has three segments. First, a small toy problem so you can concentrate on getting the implementation right. Then, an example of binary image denoising that will show how the methods can be used for real problems. And finally, an example of semantic segmentation, similar to what we discussed in the lecture.

1 Toy Example (Ising model)

1.1 Model

We define a graphical model that is a 5×5 grid with 4-connectivity. Figure 1 (left) shows its factor graph. There are 25 binary variables, $(y_i)_{i=1,\dots,25}$, with $y_i \in \{0, 1\}$. As short hand, we write $y = (y_1, \dots, y_{25})$. For each variable there is one unary factor. Its energy term, ϕ_i, is given by $\phi_i(0) = \texttt{pattern}(i)$ and $\phi_i(1) = 1 - \texttt{pattern}(i)$, for the pattern shown in Figure 1 (right). For any two pixels i and j that are horizontally or vertically adjacent there is a pairwise factor, ψ_{ij}, with energy $\psi_{ij}(0, 0) = \psi_{ij}(1, 1) = 0$ and $\psi_{ij}(0, 1) = \psi_{ij}(1, 0) = \eta$, where $\eta \geq 0$ is a free parameter that we will vary as part of the exercises. Grid-shaped models with binary variables and this form of pairwise interactions are know as Ising models.

The corresponding probability distribution is

$$p(y) = \frac{1}{Z} e^{-E(y)} \qquad \text{for} \qquad E(y) = \sum_{i=1}^{25} \phi_i(y_i) + \sum_{(i,j)} \psi_{ij}(y_i, y_j)$$

where (i, j) runs over all neighboring pixel pairs, and Z is the normalizing constant (partition function).

Our first goal is to compute the marginal probabilities, $p(y_i)$, for $i = 1, \dots, 25$.

1.2 Mean Field Inference

The graph underlying the model has cycles, so exact inference is computationally costly (though strictly speaking not impossible in this case, see bonus exercise below). Therefore, we resort to an approximate inference method:

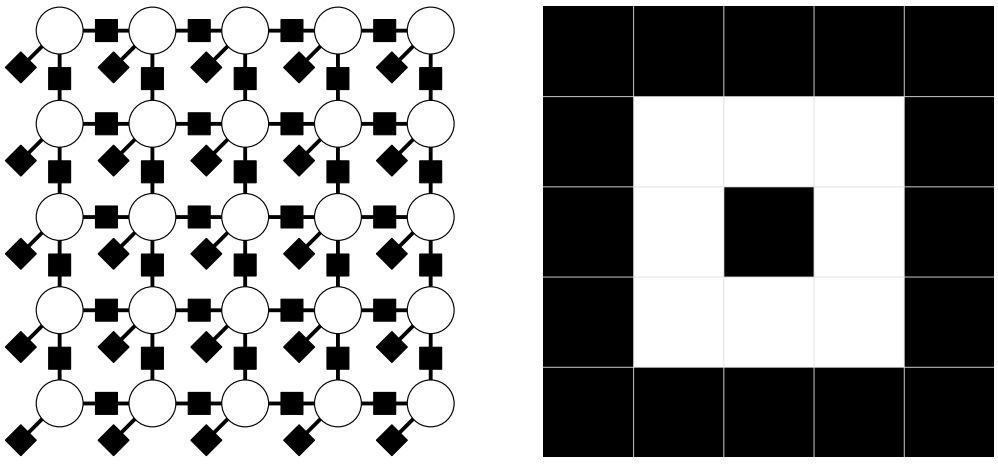

Figure 1: Left: 5×5 grid graphical model used in Section 1.1. Right: pattern of unary terms. Black means a value of 1, white means a value of 0.

mean field. Note that this is not necessarily the best method for this model. We use it here mainly because it is efficient and rather easy to implement.

Mean field inference iteratively computes approximations, q_i, to the true marginal probabilities, $p_i = p(y_i)$. Starting from $q_i(0) = q_i(1) = 0.5$, it uses the following simple update rule[1], which we write in vector notation, *i.e.* $q_i = (q_i(0), q_i(1)) \in \mathbb{R}^2$, *etc.*:

$$\tilde{q}_i \leftarrow \exp\Big(-\phi_i + \eta \sum_{j \in N(i)} q_j \Big), \qquad q_i \leftarrow \frac{\tilde{q}_i}{\tilde{q}_i(0) + \tilde{q}_i(1)}, \tag{1}$$

where $N(i)$ denotes the set of all neighbors of node i in the graph.

After any number of iterations, we can stop the process and read off approximate marginal probabilities $(q_i)_{i=1,\dots,25}$.

1.3 Implementation

Implementing the update rules (2) is straight-forward, except for the fact that we need to identify the neighbors of each variable. When using one-dimensional variables indices $i = 0, \dots, 24$ this can be done by numeric checks: for example, $i \geq 5$ ensures that i has a top neighbor, $i - 5$. Generally, it is more convenient is to use two-dimensional variable indices: (r, c) where $r = 0, \dots, 4$ runs over all rows and $c = 0, \dots, 4$ runs over all columns of the grid.

For the exercise, we recommend to use and think in *array* or *matrix* notation, which has the advantage that it is also far more efficient that using indices and loops.

As there is one entry of q for each variables in the grid, we can think of q as a 5×5 matrix (actually, a $5 \times 5 \times 2$ tensor, as each q_i has two entries, but we suppress the third axis where possible). Analogously, $\phi = (\phi_1, \dots, \phi_{25})$ is a matrix of the same shape. Then, we can update all entries of q simultaneously using the following pseudo-code:[2]

```
tmp = -φ
tmp[ : ,1:5] += eta*q[ : ,0:4]    # all columns except first
tmp[ : ,0:4] += eta*q[ : ,1:5]    # all columns except last
tmp[1:5, : ] += eta*q[0:4, : ]    # all rows except first
tmp[0:4, : ] += eta*q[1:5, : ]    # all rows except last
qtilde = exp(tmp)
q = qtilde / (qtilde[:,:,0]+qtilde[:,:,1])
```

We have used *python notation*, in which indices starting at 0, and $i : j$ stands for $(i, i+1, \dots, j-1)$, *i.e.* the right value is not included. The first index specifies rows and the second index specifies columns. To access specific subsets of rows and/or columns, we made us of *slicing*: for example, `q[0:4,:]` is the 4×5 matrix consisting of the first four rows of q with all five columns. `q[1:5,:]` is the 4×5 array consisting of the second to fifth row of q with all columns. `q[:,0:4]` is the the 5×4 matrix consisting of all rows and the first to fourth column of q, *etc.* `qtilde[:,:,0]` and `qtilde[:,:,1])` are the first and second channel of `qtilde[:,:]`, etc. *Before starting to program anything, please take a moment to check that you really understood why the above pseudocode implements* (2).

1.4 Exercise

Implement the above components and run *mean field* inference for the 5×5 model, once for each values of $\eta \in \{0, 0.5, 1\}$. Visualize the resulting approximate marginal probabilities after one to three steps. The results should look similar to Figure 2.

[1] These are not the general expressions for mean field inference, but they have already been simplified for the specific Ising model we are studying.

[2] Many other ways of implementation are also possible. For example, one could define q with additional rows and columns of *padding* entries at the boundaries. When set to 0, these can act as *pseudo-neighbors* for variables at the boundary.

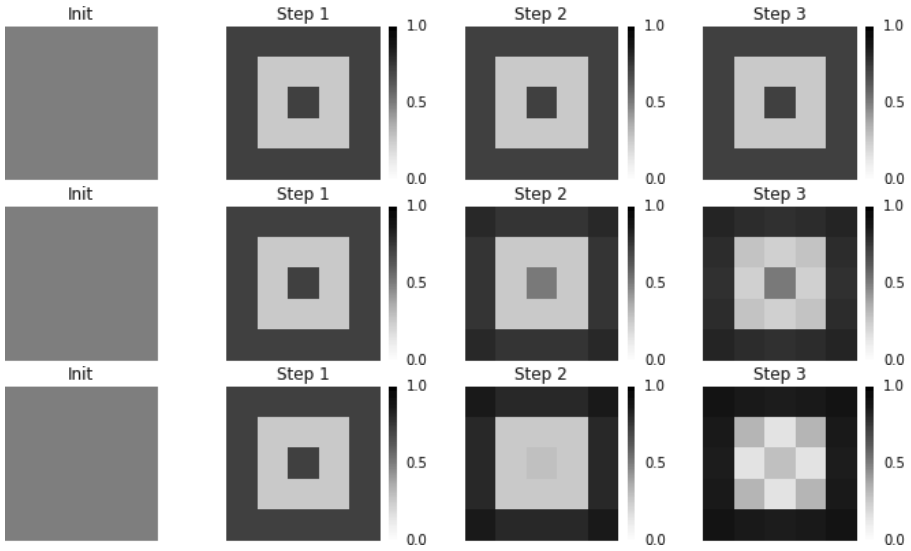

Figure 2: Results of mean field inference in the 5×5 grid graph (showing $q_i(1)$ for $i = 1, \ldots, 25$). Top row: $\eta = 0$; middle row: $\eta = 0.5$; bottom row: $\eta = 1$

2 Binary Image Denoising

We now study a real image processing problem: binary image denoising. We are given a binary image, *e.g.* a scan, that is corrupted by noise (see Figure 3 left). Our goal is to recover the image without the noise, which could look, for example, like Figure 3 (right). To do so, we rely on the above defined Ising model and mean field inference.

2.1 Model

For each pixel of the image we introduce a variable and for a grid graph (here: 10×10). As unary energies we use the intensities of the noisy image (0 for *white*, 1 for *black*). As pairwise interactions we use the same (Ising) terms as in the first part. Mean field inference provides us with approximate marginal probabilities for each pixel. Thresholding the marginal probabilities at 0.5 yields a (hopefully) cleaned up binary image.

2.2 Exercise

Implement the above components by adapting the code from the first part. Then run the indicated steps using the noisy example image `image-noisy.png` (the same as Figure 3 left). Try different values for $\eta \in \{0, 0.1, 0.5, 1\}$. Visualize the approximate marginal probabilities and resulting denoised images after one to five steps.

Now, try $\eta > 1$. Most likely, you will see artifacts and/or oscillations in the results. This is a consequence of the (terribly naive) way we implemented the mean field method. For some ways to overcome these, see the additional exercises below.

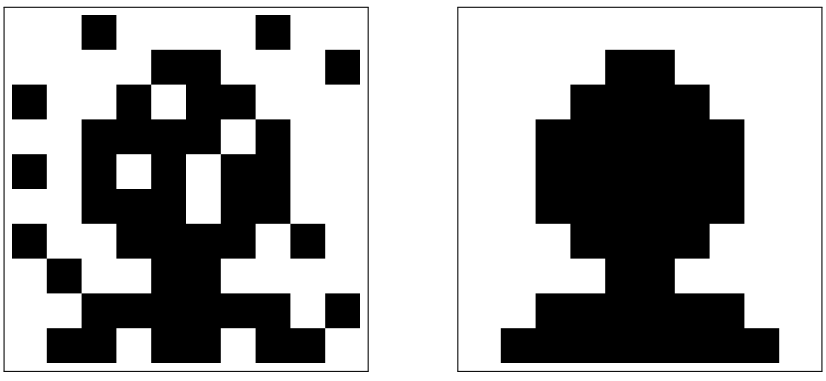

Figure 3: Left: Binary image, corrupted by noise. Right: possible 'cleaned up' image.

3 Semantic Image Segmentation

We now address a far more challenging problem: semantic image segmentation. Given a natural image, we aim at predicting a segmentation mask: all pixels that show a *horse* should be assigned the value 1, all other pixels should be assigned a value 0, see Figure 4.

We will again make use of the Ising model and mean field inference. The main differences are that a) the images are much larger, so we must make sure that our code is efficient, and b) we *learn* the unary potentials from training examples.

Figure 4: Left: image of a horse. Right: segmentation mask.

3.1 Model

To segment an image, we create a grid graph of the same size consisting of binary variables. For each variable, y_i, a value 1 indicates *horse* at the corresponding pixel x_i in the image, and 0 indicates *background*. To obtain unary potential scores, we construct an auxiliary model that will tells us, which colors are typical for *horse* pixels and which colors are typical for *background* pixels. For this, we use a *training set*: a separate set of images for which manually generated segmentation masks are available. In the training images we identify all *horse* pixels and fit a 3-dimensional Gaussian distribution (with full covariance matrix) g_{horse} to their RGB values. Analogously, we fit a Gaussian, g_{bg} to all *background* pixels. Afterwards, for every pixel i in the image we want to segment, we define a unary term, ϕ_i based on the negative logarithms of the Gaussian densities, evaluated at the RGB value of the pixel: $\phi_i(0) = -\log g_{\text{bf}}(x_i)$ and $\phi_i(1) = -\log g_{\text{horse}}(x_i)$. As pairwise interactions we use the same (Ising) terms as before.

Mean field inference provides us with approximate marginal probabilities for each pixel to be of the *horse* or *background* class. We obtain a segmentation mask by thresholding the marginal probabilities at 0.5.

3.2 Exercise

Implement the above steps. Load the training images `image-0.png`, ..., `image-7.png` from the `horses.zip` archive, as well as their segmentation masks, `mask-0.png`, ..., `mask-7.png`. Fit Gaussian distributions of *horse* and *background* color distributions based on these images.

Load the image `image-8.png` and use steps described above to compute a segmentation mask for it. Try different values for $\eta \in \{0, 0.5, 1\}$ for best effect and visualize the approximate marginal probabilities and resulting segmentations after one to five steps.

You will notice that for $\eta > 0$, the segmentations have few smaller regions with *horse* label, *i.e.* they are less noisy, but overall, the segmentaiton is still not very good yet. Two tricks will improve this: 1) instead of the unary terms above, use their values multiplied by 0.1. This reflects that the color model we use is not very good, so we should not trust it too much. 2) run the mean field inference for many more steps, *e.g.* 500. For this, you might want to disable the visualization and only display the result after all steps are done.

After completing the above steps, load the true segmentation `mask-8.png` and compute the *segmentation accuracy* of your result, *i.e.* the fraction of pixels in the segmentation masks that are correctly predicted. My model achieves 96.6% accuracy. Can your model beat this?[3]

Now load the image `image-9.png` and run the segmention method. Most likely, results are not good. Can you explain what went wrong? What accuracy score do you expect? After deciding on a number, load the ground truth segmentation mask `mask-9.png` and compute the accuracy of your result. Did you guess correctly? If not, what happened?

[3]Of course, this number is just anectodal evidence. For a proper scientific evaluation, many more test images would be required. See the additional exercises for this.

4 Additional exercises and open questions

Here are some additional exercises (in no particular order) that you can try after finishing the above ones, or during the rest of the week.

4.1 Other inference techniques 1

The *synchronous* mean field method we used is known to be fast but not very stable. For example, oscillations occur when η is large. Adjust the mean field method above such that at each iteration, only a random subset of all variables is updated. This should stabilize the results, but more updates will be required to converge.

4.2 Other inference techniques 2

Besides mean field, many other approximate inference techniques exist, *e.g.*, 1) loopy belief propagation, 2) Gibbs sampling, 3) TRW. You can implement these yourself or rely on existing libraries, such as `libdai`. Which ones work best in which setting?

At least for the 5×5 grid you can even compute the marginals exactly by brute-force summation. How much slower is this? How different are the results from the approximate techniques?

4.3 Better color models

In the image segmentation example, we used Gaussians to model the distribution of foreground and background colors. Clearly, this is a very limited model and far from the true distribution. Also, the set of 8 training examples is very small.

Try other color models, *e.g.* histogram-based or a mixture of Gaussians, and see which ones work best. Can you imagine other ways to come up with unary values besides colors? Note that it is not required that the values depend only on single image pixels (because the distribution is *conditioned* on the image). One could, *e.g.*, also make use of a neighborhood.

You can make use of the images and masks in the `morehorses.zip` archive as additional training data.

4.4 Other pairwise terms

In the pairwise terms of the Ising model, all pairs of pixels are treated equally. In computer vision, it has become common practice to instead exploit the image contents and define *contrast-sensitive* pairwise terms: $\psi_{ij}(0,0) = \psi_{ij}(1,1) = 0$ as before, but $\psi_{ij}(0,1) = \psi_{ij}(1,0) = \eta_{ij}$ with $\eta_{ij} = \eta e^{-\lambda \|x_i - x_j\|^2}$ for some parameter $\lambda > 0$. This is based on the insight that if two pixel are similar in appearance, then it is unlikely that the pixels have different labels (large ψ_{ij}). But, if the pixels are dissimilar, there might be an object boundary in the image, and then the pixels having different labels is more likely (small ψ_{ij}).

Mean-field updates with constrast-sentive pairwise terms are easily implemented. The update rules are

$$\tilde{q}_i \leftarrow \exp\Big(-\phi_i + \sum_{j \in N(i)} \eta_{ij} q_j \Big), \qquad q_i \leftarrow \frac{\tilde{q}_i}{\tilde{q}_i(0) + \tilde{q}_i(1)}, \tag{2}$$

Try to segment the horse images using these pairwise terms (you will have to try different values of λ and η to succeed). How do the segmentation results change?

4.5 Proper evaluation

The archive `evenmorehorses.zip` contains additional images and segmentation masks. Use these to evaluate your methods properly. For any method you implemented, try $\eta = 0$ and $\eta = 1$. Can you establish that the results of both settings are statistically significantly different?

4.6 Other evaluation measures

In the computer vision literature, image segmentation methods are typically not evaluated by their accuracy, but by their *intersection-over-union* (IoU) score, $\mathrm{IoU}(y, y') = \frac{\sum_i \min(y_i, y'_i)}{\sum_i \max(y_i, y'_i)}$.

Implement the IoU measure and reevaluate your previous results on `image-8.png` and `image-9.png`. Can you imagine why IoU would be preferable over *accuracy* as a measure of segmentation quality?

4.7 A challenge

Use all of the above data and any of the above methods to build the possible horse segmentation method. Use the archive `challengehorses.zip` and segment all images it contains. Send an archive with the resulting segmentation masks to `chl@ist.ac.at` with the subject line *horse challenge*.

The creator of the best performing model (evaluated by average IoU over all images) will win a price!