Ruprecht-Karls-Universität Heidelberg

Fakultät für Mathematik und Informatik

Institut für Informatik

Masterarbeit

Bestärkendes Lernen für semantische Segmentierung von Bilddaten

Reinforcement Learning for semantic segmentation of image data

Name: Paul Hilt

Matrikelnummer: 3533310

Betreuer: Prof. Dr. rer. nat. Fred Hamprecht

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.
Sowohl inhaltlich als auch wörtlich entnommene Inhalte wurden als solche kenntlich gemacht.
Die Arbeit ist in gleicher oder vergleichbarer Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Heidelberg, den

____

Paul Hilt

Abstrakt

Ein Convolutional Neural Network (CNN) zu trainieren hat in der Regel den Nachteil, dass das Optimierungssignal der Parameter sich aus einer differenzierbaren Loss Funktion ergeben muss. Das ist ein Grund dafür, dass die besten Resultate für viele Probleme die mit neuronalen Netzwerken gelöst werden von Methoden des überwachten Lernens kommen, da in diesem Fall eine aussagekräftige und differenzierbare Loss Funktion auf Basis von Ground Truth instanzen definiert werden kann. Gegensätzlich dazu bei unüberwachtem Lernen, wo es in der Regel schwer fällt, eine differenzierbare Loss Funktion nur auf Basis vorher definierter Regeln zu definieren die das Wissen, welches zur Herstellung von Ground Truth instanzen benötigt werden würde, formalisieren.
Bestärkendes Lernen bietet die Möglichkeit eine Vorhersage zu evaluieren und von dieser Evaluation zu lernen wobei keine Restriktionen an den Evaluierungsprozess existieren. Die Methode die in dieser Arbei vorgestellt wird, formuliert das Problem der Segmentierung von Bildern als Problem welches mit bestärkendem Lernen gelöst werden kann.
Mehrere Algorithmen des bestärkenden Lernens wie etwa ACER [1] oder SAC [2] werden an dem Problem getestet. Die zentralen neuronalen Netzwerke basieren auf graph Faltungen und treffen Vorhersagen für Kantengewichte auf Kanten in Graphen, welche sich aus Superpixel Segmentierungen der Rohdaten ergeben. Auf diesen Vorhersagen gründend wird anschließlich mithilfe des Multicut Algorithmus eine finale Segmentierung erstellt.
Die Methode wird auf einem künstlich erstellten Datensatz getestet, wobei überwachtes Lernen sowie unüberwachtes Lernen zum Einsatz kommt.

Abstract

Training a CNN has the drawback that the feedback signal for the parameter optimization has to come from a differentiable loss function. This is one reason why the best results stem from supervised learning where an expressive and differentiable loss can be defined using ground truth labels. Unsupervised learning does not yet provide methods to evaluate a prediction in a differentiable way such that the result is an expressive loss that can compete with a supervised loss.
Reinforcement learning on the other hand allows for a evaluation used for a feedback signal that does not have to be dependent on the neural networks parameters and therefore does not have to be differentiable.
The method proposed in this work formulates the image segmentation problem as a reinforcement learning problem and provides a basic pipeline to solve this task.
Several reinforcement learning algorithms like ACER [1] and SAC [2] have been tested on the problem. The core networks are based on graph convolutions on a region adjacency graph (rag) of a superpixel segmentation. In this graph the representations of the nodes are based on predictions of an upstream CNN predicting pixel embeddings. The final predictions are edge weights for the edges in the rag. Based on this edge weights the multicut algorithm provides a final segmentation.
The method was tested on a toy dataset where supervised, semi supervised and unsupervised approaches have been used.

Contents

1 Introduction **6**
1.1 Motivation 6
1.2 Contribution 6

2 Preliminaries **7**
2.1 Image segmentation 7
2.2 Reinforcement learning (RL) 8
2.2.1 Value functions 10
2.2.2 Q-learning 10
2.2.3 Policy gradient methods 11
2.2.4 Maximum Entropy Reinforcement Learning 12
2.2.5 Soft Actor-Critic (SAC) 13
2.2.6 Common optimization methods 14
2.3 Geometric deep learning 16
2.4 Mutex watershed 17
2.5 Image partitioning by multicuts 19
2.6 Principal component analysis 20
2.7 Loss functions 21
2.7.1 Generalized dice loss 21
2.7.2 Contrasive loss 22
2.7.3 Triplet loss 23
2.8 The reparameterization trick 24
2.9 The Heugh transform 25

3 Methods **27**
3.1 Using RL for the image segmentation task 27
3.2 Using RL for pixel affinity predictions 27
3.3 Overview over the proposed pipeline 28
3.4 The pipeline in reinforcement learning terminology 29
3.4.1 The state 29
3.4.2 The actions 29
3.4.3 The reward 30
3.4.4 The agent 31
3.4.5 The environment 31
3.4.6 The problem of local optima 31
3.4.7 Definition of the RL algorithm 32
3.5 Obtaining superpixels from mutex watershed 33

3.6 The embedding network 33
3.7 The actor critic networks 34
3.8 Finding subgraphs 37
3.8.1 Thoughts on dependence 38
3.9 Technical details 39
3.9.1 Batch processing 39

4 Experiments and results **40**
4.0.1 Supervised training 41
4.0.2 Unsupervised training 43

5 Conclusion **45**
References 45

Introduction

1.1 Motivation

Unsupervised learning is on a completely different level of causal inference than supervised learning. In supervised learning the labels that are provided form an intermediate step between the knowledge of the causal connection between raw data and labels, and the application of that knowledge to a learning process. Getting rid of the label generation and directly apply the necessary knowledge to the learning process would safe the whole expensive labeling process.

Unsupervised image segmentation algorithms like graphical models or clustering algorithms usually lack in incorporating semantic information in the image. Also classical training of CNNs is not fit for unsupervised learning because the computation that leads to the loss has to be differentiable. There are yet no methods to formulate prior knowledge in the form of constraints for size, shape, texture etc. as a differentiable loss term. Even simply formulating them, such that the formulations are exact and complete in the sense that the information in them is sufficient to describe the dependence between raw data and labels, is usually a hard task. However assuming the problem is easy enough and such formulations can be generated, it is still not clear how to train a neural network on such rules.

Reinforcement learning (RL) methods have the property that learning is based on temporal differences in rewards. Here the rewards can depend on the parameters of the prediction model but do not have to. This of course means that their generation does not have to be differentiable. Usually the rewards are obtained by measurements in a complex physical environment which is too difficult to model virtually. This property motivates to use RL in other learning problems such as the image segmentation problem.

1.2 Contribution

This work proposes an image segmentation pipeline based on RL and graph neural networks (GCN) where the level of supervision can be adjusted as needed. The pipeline has been tested on the toy problem of segmenting discs. It was tested in an almost unsupervised setting where the only supervision stems from the pretraining of a feature extractor network. The main training of the pipeline is usupervised where a non differentiable evaluation of the predictions is based on prior knowledge on the objects in the image.

Preliminaries

2.1 Image segmentation

Image segmentation or image partitioning is defined as the process of dividing a digital image into sets of pixels also known as the objects the image.
There are many variations concerning the segmentation method itself as well as concerning the goal that is to be achieved. With the rise of CNNs, nowadays these methods are usually fully or py parts learning based. That means, that some parameterized function is used to approximate a target distribution which can only be described by prior knowledge and/or by drawing samples from it. For the task of image segmentation these samples are of the form (x, y) which is a realization of a random variable (X, Y) with probability distribution $P_{X,Y}(x, y)$. Samples represent the raw input image x that is to be segmented and the desired segmentation y also referred to as label or label image. The goal is to learn a function $f(x)$ such that it approximates $\mathbb{E}[Y|X = x]$ at best. Since $P_{X,Y}$ is initially not known and only few samples and/or some prior knowledge on its properties are available, the approximation can only be achieved by the Monte Carlo estimate of the expectation obtained from those samples and by dexterous use of the prior knowledge.
Some of the variations of learning based methods for image segmentation are distinguished by their level of supervision during the optimization. This is mainly defined by the amount of data samples and prior knowledge on the distribution $P_{X,Y}$ that is available.

- **supervised segmentation** is the highest level of supervision. Here only samples from $P_{X,Y}$ are available to the method. If enough samples are available such that all regions in the domain of the distribution are covered sufficiently, this is usually all one needs to arrive a a satisfactory result. However for most applications the set of available samples is only very limited.

- **unsupervised segmentation** is the lowest level of supervision. Here no samples from $P_{X,Y}$ are available to the method. The optimization method has to completely rely on prior knowledge on the underlying distribution. Realizations of X are usually still available and can be used for the learning process.

- **semi-supervised segmentation** is the transition between the previous two. Similar to supervised learning, semi-supervised learning uses samples from $P_{X,Y}$ but not only. There are also realizations of X available as well as some prior knowledge on $P_{X,Y}$ which is used during the learning process as well.

- **self-supervised learning** is usually referred to when the method generates some kind of supervisory signal for itself. E.g. an automated labeling procedure to generate sample approximations $(x, \bar{y})$. Self-supervised is a special case of unsupervised learning.

variations that focus more on the goal that should be achieved are

- **semantic segmentation** is the process of assigning class labels to each pixel in the image. Different objects instances of the same class are labeled equally. Usually only some object classes of interest get a unique class label assigned to. All other object classes receive the label background.
- **instance segmentation** is similar to semantic segmentation in the sense that each pixel in the image is assigned a label to. This label assigns a pixel either to background or to an instance of an object class. Therefore different objects of the same class are labeled differently. Here the label of an instance is also referred to as object id.
- **panoptic segmentation** is a fusion of the previous two. For all pixels belonging to instances of some defined set of classes, instance segmentation is performed. For the remaining pixels, semantic segmentation without a background label is performed. The object instances for which instance segmentation is performed are referred to as ”things” (objects with a well defined shape like cars, buildings ...) and the object instances for which semantic segmentation is performed are referred to as ”stuff” (background regions like grass, sky ...).

2.2 Reinforcement learning (RL)

Please note that this is an aggressively shortened summary. For a deeper introduction please refer to [3]. The Reinforcement learning problem originates from the idea of learning by interacting with an environment. The object that is learning is doing so by retrieving information from cause and effect. The causal model that is learned in such a way is updated with each change of the environment that can be related to some action. Therefore the learned model fits the true model increasingly better with the number of induced causes and observed effects.
This type of learning problem can be modeled by ”Finite Markov Decision Processes”. Such a model usually needs the following elements:

- **Environment**
  The environment is an object inheriting one ore multiple complex physical processes. Information on the current state of that process are typically obtained by measurements in the physical environment itself or by evaluating a computer model of the actual environment.
- **State**
  The state s_t is generated by the environment and reflects the current state that the environment is in. It changes over time according to the dynamics within the environment.
- **Action**
  An action a_t is a cause that might change the state of the environment. Actions are produced by the agent.

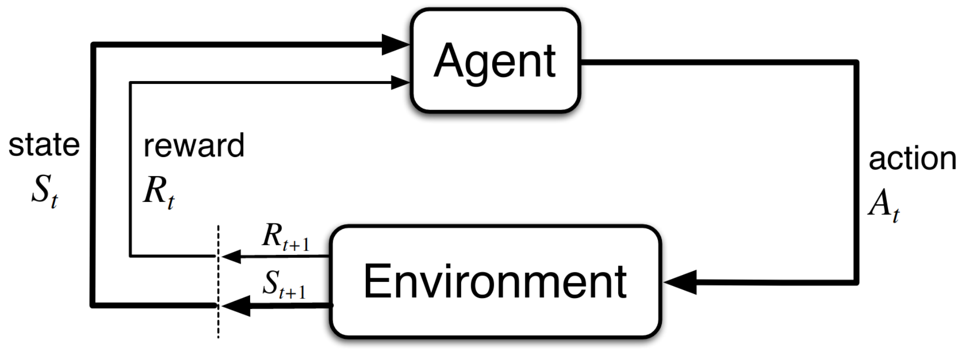


Figure 2.1: agent environment interaction [3]

- **Reward** The reward r_t is a scalar value that is produced by the environment and received by an agent.
- **Agent**
  The agent is an object which generates actions a_t and observes the caused change from s_{t-1} to s_t of the environment as well the along going reward r_t.
- **Policy**
  A policy $\pi(a_t|s_t)$ is a probability distribution over the set of possible actions a_t given the state s_t at a time step t. A agent is essentially defined by its policy as each action that is taken is sampled from that policy.

The signal flows between agent and environment are depicted in figure 2.1
All signals in this model are time dependent. Such a model satisfies the Markov Property if the next state s' and reward r only depend on the current state action tuple (s, a). If assuming finite state and action spaces together with the Markov property yields a Finite Markov Decision Process. The environment dynamics can therefore be represented by the bi-variate probability distribution

$$p(s', r|s, a) = Pr\{r_{t+1} = r, s_{t+1} = s'|s_t = s, a_t = a\} \tag{2.1}$$

Further, if a_t is sampled from $\pi(a_t|s_t)$ the Markov Property induces conditional independence of (s_{t-1}, r_{t-1}) and (s_{t+1}, r_{t+1}) given s_t.

The agents task is to predict a policy that maximizes the expected future rewards. This objective is given by

$$\underset{\pi}{\arg\max} \underset{p_\pi}{\mathbb{E}} \left[\sum_{t=0}^{T} r_t | s_0 \right] \tag{2.2}$$

Here T marks the time limit of the process and p_π represents the environment dynamics following a action history sampled from π.

2.2.1 Value functions

Most methods aiming to solve eq. (2.2) use estimations of so called value functions. This are functions that provide a quality measure for an agent evaluating a state or a state action tuple.
Commonly three value functions are used. The state value function, the state action value function and the advantage value function.
They all depend on the expected discounted future rewards

$$g_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1}. \tag{2.3}$$

Here γ is referred to as the discount factor. Its value usually determines how prospective future rewards are weighted in the value function. E.g if $\gamma < 1$ rewards that are closer to t get a higher weight than those that are occuring at later time steps. For $\gamma > 1$ the contrary holds.
The state value function is defined by

$$V_\pi(s) = \mathop{\mathbb{E}}_{p_\pi} \left[g_t | s_t = s \right], \tag{2.4}$$

the state action value by

$$Q_\pi(s, a) = \mathop{\mathbb{E}}_{p_\pi} \left[g_t | s_t = s, a_t = a \right] \tag{2.5}$$

and finally the advantage value by

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s) \tag{2.6}$$

it follows

$$V_\pi(s) = \mathop{\mathbb{E}}_{a \sim \pi} \left[Q_\pi(s, a) \right] \tag{2.7}$$

Usually the objective is to maximize either one or more value functions. Note that, $\max_\pi V_\pi(s)$ and $\max_\pi Q_\pi(s, a)$ satisfy Bellman's principle of optimality. Hence they can be solved exactly with Dynamic Programming. This is referred to as the tabular solution. However for most problems this is not feasible and the value functions are approximated by neural networks.

2.2.2 Q-learning

Q-learning is a method to find the state action value maximizing policy by minimizing temporal differences. This is often the backbone for policy gradient algorithms. Let

$$\pi(a|s) = \delta(a - \arg\max_{a'} Q_\pi(s, a')) \tag{2.8}$$

where δ is the Dirac delta function. Therefore sampling from the policy yields the maximum action state value. The objective is to approximate Q_π which is achieved by the temporal difference loss

$$\mathcal{L}_{TD} = \frac{1}{2}\left(r_t + \gamma \max_a Q_\pi(s_{t+1}, a) - Q_\pi(s_t, a_t)\right)^2. \tag{2.9}$$

this kind of approximation is usually referred to as one step temporal difference method. The optimality follows directly and the convergence of the approximation under certain conditions has been proven in [4].
In contrast to one step temporal difference methods there are Monte Carlo methods which collect the loss over whole episodes. One episode is defined by the history of temporal differences between the starting state $t = 0$ and an end state $t = T$. This methods usually use eligibility traces [5].
Optimizing eq. (2.9) is referred to as on-policy policy optimization where the target policy π is the policy which is used when actions are sampled. This however is problematic as the target policy, defined by eq. (2.8) depends on Q_π which is not trustworthy in early training iterations as this is the approximating function. In order to have more control over the sampling of actions which is usually referred to as exploration, a data collection policy $\mu(a|s)$ is used during training. Therefore in an off-policy setting, eq. (2.9) is redefined to

$$\mathcal{L}_{TD} = \frac{1}{2}\left(r_t + \gamma \max_a Q_\mu(s_{t+1}, a) - Q_\mu(s_t, a_t)\right)^2. \tag{2.10}$$

and during inference

$$\bar{\pi}(a|s) = \delta(a - \arg\max_{a'} Q_\mu(s, a')) \tag{2.11}$$

is used. Off policy methods fight the issue of the distributional mismatch between the target policy π and the exploration policy μ. There are many solutions to overcome this mismatch. Many use importance sampling or variance reduction techniques.[6]

2.2.3 Policy gradient methods

This class of algorithms optimize the parameters θ of a policy $\pi_\theta(a|s)$. Let

$$\rho(\pi) = \sum_{t=1}^{\infty} \mathop{\mathbb{E}}_{\substack{s\sim d_\pi(s)\\ a\sim\pi(a|s)}} [r_t|s_0] = V_\pi(s_0) \tag{2.12}$$

be the expected, discounted future reward per step and let

$$d_\pi(s) = \sum_{t=0}^{\infty} \gamma^t Pr\left\{s_t = s|s_0, \pi\right\} \tag{2.13}$$

be the discounted stationary distribution of states under π. Then

$$\frac{\partial \rho_\pi}{\partial \theta} = \sum_s d_\pi(s) \sum_a \frac{\partial \pi(a|s)}{\partial \theta} \bar{Q}_\pi(s, a) \tag{2.14}$$

Is the policy gradient with which gradient ascent on the policy can be performed in order to maximize ρ. A proof and a thorough discussion can be found in [7]. Note that the policy gradient is on-policy and

that $\bar{Q}_\pi$ is an approximation of the exact state action value function Q_π.
Since π is a probability distribution it follows that $\sum_a \frac{\partial \pi(a|s)}{\partial \theta} = 0, \forall s \in S$. Eq. (2.14) can therefore be rewritten as

$$\frac{\partial \rho_\pi}{\partial \theta} = \sum_s d_\pi(s) \sum_a \frac{\partial \pi(a|s)}{\partial \theta} \left[\bar{Q}_\pi(s,a) + b(s)\right], \qquad b : S \to \mathbb{R}\,. \tag{2.15}$$

The function b is called a baseline and is often used to reduce variance and bias in the gradient. Using $\frac{\nabla_\theta \pi(a|s)}{\pi(a|s)} = \nabla_\theta ln(\pi(a|s))$ and $\mathbb{E}_{x\sim p(x)}[f(x)] = \sum_x p(x)f(x)$, rewriting eq (1.13) yields

$$\frac{\partial \rho_\pi}{\partial \theta} = \mathop{\mathbb{E}}_{\substack{s\sim d_\pi(s)\\ a\sim\pi(a|s)}} \left[\nabla_\theta ln(\pi(a|s)) \left[\bar{Q}_\pi(s,a) + b(s)\right]\right]. \tag{2.16}$$

In practice there are large state and action spaces, thus the expectations w.r.t s and a become infeasible to obtain. Using $\mathbb{E}_{x\sim p(x)}[f(x)] = \frac{1}{n}\sum_n f(x), n \to \infty, x \sim p(x)$, sample based learning uses enough samples of s and a in order to obtain a good enough approximation of the expectations. Therefore the approximation of eq. (2.16) is

$$\frac{\partial \rho_\pi}{\partial \theta} \simeq \frac{1}{n}\sum_n \nabla_\theta ln(\pi(a|s)) \left(\bar{Q}_\pi(s,a) + b(s)\right), \qquad s \sim d_\pi(s), \quad a \sim \pi(a|s), \quad n \to N \tag{2.17}$$

Here N is the number of samples drawn which is in practice usually $N = 1$. This leads to Actor Critic methods where there are two instances that are updated in a turn based fashion. The critic is the state action value function approximation and the actor is approximating the policy with the policy gradient. Intuitively the critic evaluates the action taken by the actor who uses this evaluation to scale its gradient update (this is the role of $\bar{Q}_\pi$ in eq. (2.17)).

2.2.4 Maximum Entropy Reinforcement Learning

In off-policy settings it is common to use a data collection policy μ which has a large entropy in order to encourage exploration of the action and state spaces. The principle of maximum causal entropy has been introduced by [8] and was elaborated on, among others, by [9]. The key idea is to incorporate an entropy term into the objective function, acting like a regularizer.

$$\rho^{\mathcal{H}}(\pi) = \sum_{t=1}^{\infty} \mathop{\mathbb{E}}_{\substack{s_t\sim d_\pi(s_t)\\ a_t\sim\pi(a_t|s_t)}} [r_t + \alpha(t)\mathcal{H}(\pi(\cdot|s_t))|s_0] \tag{2.18}$$

Here α is a non-negative regularization weight which is usually monotonically decreasing with increasing t and $\mathcal{H}$ is some entropy measure. If α becomes 0, eq. (2.18) is equal to eq. (2.12), therefore intuitively α should be high in the early phase of training the policy and value function and converge to 0 as the policy gets closer to the optimal policy.
This objective is on-policy and still gives control over the exploration behavior. Usually it does not bother if the policy has high entropy, since during inference the action where ”π” has maximum probability is selected strictly. This makes especially unimodal distributions attractive. They fixate on single actions and they imply few parameters only that need to be learned (e.g. mean and variance of normal dis-

tributions). However often more expressive multimodal distributions fit the true distribution which is approximated better. In [10] this idea has been extended by using normalizing flows. Normalizing flows [11] are based on the idea of transforming a probability density function by letting each sample undergo a transformation. If this transformation is a diffeomorphism, the probability of the transformed sample can be determined. Let T be a diffeomorphism of a real vector u sampled from $q(u)$.

$$x = T(u) \quad \text{where} \quad u \sim q(u) \tag{2.19}$$

then

$$p(x) = q(T^{-1}(u))|detJ_T(u)|^{-1} \tag{2.20}$$

where $J_T(u)$ is the Jacobian matrix of T w.r.t. u. In practice, an invertible neural network can be trained to transform a simplistic density function into a more expressive one.
E.g. let the agent predict mean and variance of a Normal distribution. Then in the data collection process, actions are sampled from a transformed Normal distribution where the transform encourages entropy and maybe also multiple modes. Assuming the sampling happened using the reparametrization trick (see section 2.8), then the log probabilities and their gradient in eq (2.17) can still be calculated. This is a on-policy training with a expressive density function and the advantage is that easy reparameterization can still be used, as sampled is from the base distribution.

2.2.5 Soft Actor-Critic (SAC)

This algorithm was introduced by [2]. They aim is to maximize the objective in eq. (2.18). Particularly they focus on the selection of the weight factor α and show, that it can be seen as a learnable parameter which is trained jointly with actor and critic networks.
The SAC is derived from the soft policy iteration where the temporal difference equation for the action value depends on the soft value function which is defined as

$$V_\pi(s_t) = \mathop{\mathbb{E}}_{a \sim \pi(a|s_t)} \left[Q_\pi(s_t, a) - \alpha log(\pi(a|s_t)) \right]. \tag{2.21}$$

Here the negative log probabilities correspond to the entropy measure $\mathcal{H}$ in eq. (2.18). The state action value function loss yields

$$\mathcal{L}_{critic} = \frac{1}{2}(Q_\pi(s_t, a_t) - (\gamma \mathop{\mathbb{E}}_{s_{t+1} \sim d_\pi(s)} [V_\pi(s_{t+1})] + r_t))^2. \tag{2.22}$$

For the policy improvement step the policy is updated such that it approximates softmax$_a(\frac{1}{\alpha}Q_\pi(s,a))$ where Q_π is the soft action value function, learned by minimizing eq. (2.22). The loss for the policy then yields

$$\mathcal{L}_{actor} = DKL_a \left[\pi(a|s_t) \middle|\middle| \frac{exp(\frac{1}{\alpha}Q_\pi(s_t, a))}{Z(s_t)} \right]. \tag{2.23}$$

Here DKL_a is the Kullback Leibler Divergence over the actions and $Z(s_t)$ is the partition function in the softmax distribution.

$$Z(s_t) = \sum_a Q_\pi(s_t, a) \tag{2.24}$$

It is usually too expensive to evaluate $Z(s_t)$ since it involves integrating/summing over the state action value space which means many forward passes through the neural network that represents Q_π. Since $\mathcal{L}_{actor}$ is minimized by gradient descent methods, only the gradient of eq. (2.23) is needed. If the KL-divergence term is expanded, $log(Z(s_t))$ becomes an additive term and therefore vanishes in the gradient.
Note that α in eq. (2.23) gives control over the differences between state action values and therefore over the entropy in the resulting softmax distributution. *Lemma 2* in [2] claims the improvement of Q_π with each optimization step of $\mathcal{L}_{actor}$.
The gradient of eq. (2.23) w.r.t. the parameters θ of π yields

$$\nabla_\theta \mathcal{L}_{actor} = \nabla_\theta \mathop{\mathbb{E}}_{\substack{s_t \sim d_\pi(s_t) \\ a_t \sim \pi(a_t|s_t)}} [\alpha log(\pi(a_t|s_t)) - Q_\pi(s_t, a_t)] \tag{2.25}$$

approximating eq. (2.24) by a sample based method yields

$$\nabla_\theta \bar{\mathcal{L}}_{actor} = \nabla_\theta \left[\alpha log(\pi(a_t|s_t)) - Q_\pi(s_t, a_t)\right] \tag{2.26}$$

minimizing this loss by a gradient descent method involves backpropagating the gradient through a sampling procedure. This can be made differentiable by the reparameterization trick (see section 2.8).
The authors in [2] also provide a method to determine the entropy adjustment α such that it takes the minimal value needed to maximize the maximum entropy objective in eq. (2.18), assuming a fixed policy π.
In practice, reinforcement learning problems have high dimensional action spaces but only one dimensional rewards. Therefore the learned state action value function of the critic is also one dimensional in contrast to the actor who predicts the statistics for either the policy for each action dimension or the statistics for a multivariate probability distribution covering all actions. For the former the joint probability is the product of probabilities over all actions. This results in summing the log probabilities of the actions in eq. (2.26).

2.2.6 Common optimization methods

There are numerous optimization methods for reinforcement learning problems. This is just a listing of only very few but important ones, reviewed and evaluated in [12].

- **Double Q-learning**
  Conventional Q-learning is affected by an overestimation bias of state action values. The authors in [13] decouple the action selection from its evaluation by learning two state action value functions independently resulting in the loss

$$\mathcal{L}_{TD} = \frac{1}{2}\left(r_t + \gamma Q_\pi^{(\bar{\phi})}\left(s_{t+1}, \arg\max_a Q_\pi^{(\phi)}(s_{t+1}, a)\right) - Q_\pi^{(\phi)}(s_t, a_t)\right)^2. \tag{2.27}$$

Here ϕ and $\bar{\phi}$ are the parameters of the independently trained state action value functions respectively. A similar method trains two action value functions independently and and takes for all evaluations the min value of the two network predictions. Both methods show a reduction in overestimation.

- **Prioritized replay**
  During data collection the tuples (s_t, a_t, r_t, s_{t+1}) are stored in a replay buffer. During training phases it is sampled uniformly from that buffer.
  In [14] the sampling is not uniform but rather with a probability p_t relative to the last encountered loss of that replay tuple.

$$p_t \propto \mathcal{L}_{TD}^{\omega}. \tag{2.28}$$

  Raising the loss to the power of the parameter ω determines the shape of the distribution. New transitions that did not produce a loss yet are always sampled with maximum priority.

- **Multi-step learning**
  Q-learning bootstraps from single step temporal difference losses. The authors in [4] introduced multi-step temporal differences which is the transition from Monte Carlo methods to single step temporal difference methods. The n-step return is defined as

$$r_t^{(n)} \equiv \sum_{k=0}^{n-1} \gamma_t^k r_{t+k+1} \tag{2.29}$$

  then the multi step temporal difference loss yields,

$$\mathcal{L}_{TD} = \frac{1}{2}\left(r_{t+1}^{(n)} + \gamma^n \max_a Q_\pi(s_{t+n}, a) - Q_\pi(s_t, a_t)\right)^2 \tag{2.30}$$

  optimizing multistep temporal difference losses with n sampled uniformly from the interval $[1..T]$ results in faster learning.

- **Dueling networks**
  This was introduced by [15]. It is an optimization method based on the neural network atrchitecture of value functions in value based RL. It features one state-value and one advantage-value stream of computation that both share a common state feature extractor network $f(s)$. This leads to this factorization of action-values

$$Q_\pi^{(\phi)}(s, a) = V_\pi^{(\eta)}(f^{(\xi)}(s)) + A_\pi^{(\psi)}(f^{(\xi)}(s), a) - \frac{\sum_{a'} A_\pi^{(\psi)}(f^{(\xi)}(s), a')}{N_{actions}} \tag{2.31}$$

Here η, ψ and ξ are the parameters of the state value function, the advantage value function and the state feature extractor respectively. ϕ is the concatenation of η, ψ and ξ.
The last term in eq. (2.31) approximates $\mathbb{E}_{a' \sim \pi} A_{\pi}^{(\psi)}(f^{(\xi)}(s), a') = 0$. This equality follows from combining eq (2.6) and eq(2.7). Then eq. (2.31) follows from eq. (2.4), eq. (2.5) and eq. (2.6). This method outperforms value based RL methods on common RL benchmarks.

- **Asynchronous Advantage Actor Critic (A3C)**
  The authors in [16] define an asynchronous learning method for actor critics. Here multiple actors and critics optimize the same parameters of value function and policy in parallel based on a common replay buffer. Depending on how much memory and cpu/gpu cores are available this can speed up learning significantly.

2.3 Geometric deep learning

Convolutional neural networks use the convolution operation to "filter" a regular grid graph of certain dimension with a filter consisting of learnable parameters. Graph convolution, as introduced by [17] generalizes this notion of convolutional filtering to arbitrary graphs. Through that it is possible to learn functions on non eucledian, structured domains that have a notion of locality.
Since then the field developed rapidly, [18] provides a good overview of the research that was done so far.
Most of the research focuses on the application where graphs are constructed from discretizations of 2-dimensional manifolds, embedded in a 3-dimensional eucledian space, usually called point clouds. However the principle can be used for arbitrary graphs that contain features in their nodes.
Typically there are two equivalent definitions of convolution on graphs. One is the spectral definition which suffers from large complexity in terms of memory and time. The other one is a spatial construction motivated from signal flows on graphs which is much faster as it operates with sparse representations of the graph. A quick summary of the latter as in [19] is given below.

Let a graph be represented by $G = (X, (A, E))$ where $X \in \mathbb{R}^{N \times m}$ is a node feature matrix of N, m-dimensional node feature vectors, with nodes ecoded as $i \in [1..N]$. A is a set of adjacency tuples where $A \in \mathbb{N}^{2 \times |E|}$ encodes the set of $|E|$ edges with n-dimensional edge features $E \in \mathbb{R}^{|E| \times n}$.
The generalization of the convolutional operator, locally expressed by means of the neighborhood $\mathcal{N}(i)$ around node i, is

$$\vec{x}_i' = \gamma \left(\vec{x}_i, \underset{j \in \mathcal{N}(i)}{\boxplus} \phi \left(\vec{x}_i, \vec{x}_i, \vec{e}_{ij} \right) \right) \tag{2.32}$$

where $\boxplus$ is a differentiable and permutation invariant function such as the sum or the mean. γ and ϕ are differentiable functions represented by multi layer perceptrons. This convolution is also referred to as message passing scheme. All this convolution operations w.r.t. to each node in the graph allow parallel computation what makes this schemes fast.

2.4 Mutex watershed

The Mutex Watershed algortihm introduced in [20] is a image partitioning algorithm based on watersheding that is able to operate without a prior seeding.
Like most image partitioning algorithms it is defined on a graph $G = (V, E^+ \cup E^-, W^+ \cup W^-)$ with a set of vertices V, a set of edges as the disjoint union of attractive edges E^+ and repulsive edges E^- and a set of corresponding edge weights $W^+ \cup W^-$. Each vertex in this graph represents uniquely a pixel in the corresponding image. The edge weights are based on the affinity between the incidental vertices the respective edge. The affinity between two nodes i and j is the probability p_{ij} of the nodes belonging to the same partition in the posterior partitioning. These affinities can be based on differences in pixel intesities or be predicted by e.g. a CNN.
Attractive edges $e^+_{ij} \in E$ have edge weights $w^+_{ij} \in W$ with $w^+_{ij} = p_{ij}$. Repulsive edges $e^-_{ij} \in E$ have edge weights $w^-_{ij} \in W$ with $w^-_{ij} = 1 - p_{ij}$.
A partitioning on G is defined by the disjoint union of a set of attracive and a set of repulsive edges by the active set $A = A^+ \cup A^-$ that encode hard merges and mutual exclusions of vertices and partitions. To represent a valid partitioning, the set A has to satisfy cycle constraints. Defining the set $\mathcal{C}_i(A)$ with $A \subseteq E$ as the set of all cycles with exactly i active repulsive edges

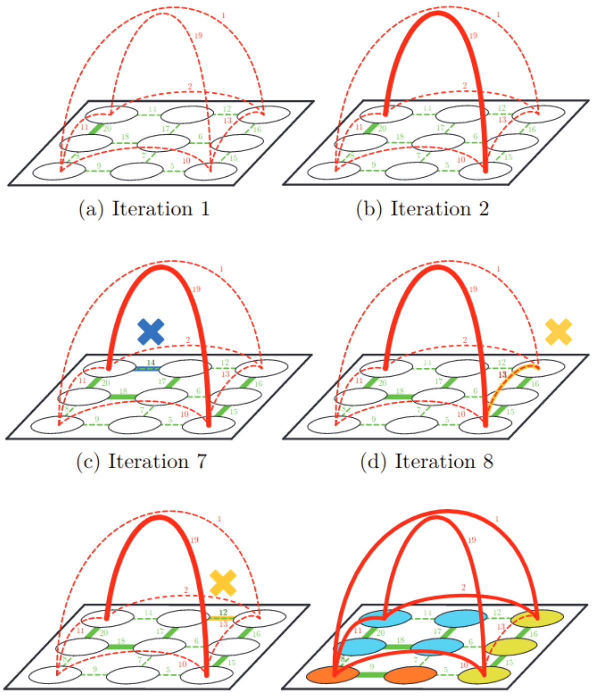
$$\mathcal{C}_i(A) := \left\{ c \in cycles(G) | c \subseteq A \text{ and } |c \cap E^-| = i \right\}, \tag{2.33}$$

a valid partitioning can only be inferred from an active set A if $\mathcal{C}_1(A) = \emptyset$. If additionally $\mathcal{C}_0(A) = \emptyset$, the algorithm can be defined as the search for the minimal spanning tree in each partition.

Algorithm 1: Mutex Watershed [20]

```
  Data: weighted graph $G = (V, E^+ \cup E^-, W^+ \cup W^-)$
  Result: clusters defined by spanning forest $A^\star \cap E^+$
1 Initialization: $A = \emptyset$;
2 for $(i, j) = e \in (E^+ \cup E^-)$ in descending order of $W^+ \cup W^-$ do
3     if $\mathcal{C}_0(A \cup \{e\}) = \emptyset$ and $\mathcal{C}_1(A \cup \{e\}) = \emptyset$ then
4         $A \leftarrow A \cup e$ ;
5 $A^\star \leftarrow A$ ;
6 return $A^\star$
```

An example walk through of the algorithm is depicted in figure 2.2.



(a) Iteration 1 (b) Iteration 2

(c) Iteration 7 (d) Iteration 8

(e) Iteration 9 (f) Final active set

Figure 2.2: [20] Some iterations of algorithm 1 applied to a graph with weighted attractive edges (green) and repulsive (red) edges. Edges that are part of the active set A at each iteration are shown in bold. On termination (f), the connected components in $A \cap E^+$ represent the partitions of the final partitioning. Edges that are not added to A because of the violation of $\mathcal{C}_0$ or $\mathcal{C}_1$ are highlighted in blue and yellow respectively.

Algorithm 1 minimizes a energy functional that is defined by the active set A. This requires the following definition

Definition 1 ***Dominant power*** *[20]:*
Let $G = (V, E, W)$ be an edge weighted graph, with unique edge weights $w_e \in \mathbb{R}_0^+$, $\forall e \in E$. Then $p \in \mathbb{R}^+$ is called a dominant power if:

$$w_e^p > \sum_{\substack{t \in E \\ w_t < w_e}} w_t^p \qquad ,\forall e \in E \tag{2.34}$$

This allows the definition of the objective that is solved by algorithm 1

Definition 2 ***Mutex Watershed Objective*** *[20]:*
Let $G = (V, E, W)$ be an edge weighted graph, with unique edge weights $w_e \in \mathbb{R}_0^+$, $\forall e \in E$ and $p \in \mathbb{R}^+$ a dominant power. Then the Mutex Watershed Objective is devined as the integer linear program:

$$\min_{a \in 0,1^{|E|}} -\sum_{e \in E} a_e w_e^p \tag{2.35}$$

$$\textit{s.t.} \quad \mathcal{C}_0(A) = \mathcal{C}_\infty(A) = \emptyset, \tag{2.36}$$

$$\textit{with} \quad A := \{e \in E | a_e = 1\} \tag{2.37}$$

2.5 Image partitioning by multicuts

The multicut problem is a graph cuts problem that is in [21] redefined for the image partitioning task. The unsupervised partitioning on a grid graph $G = (V, E)$ where each node $v \in V$ corresponds to a pixel in an image can be defined as the following minimization problem

$$\min_{x \in L^{|V|}} \sum_{uv \in E} \beta_{uv} I(x_u \neq x_v), \qquad L = \{1, ..., |V|\} \tag{2.38}$$

where I is a indicator function that maps a boolean expression to 1 if it is true and to 0 otherwise. L is the set of all possible labels, β_{uv} is the edge cost that is active if u has a different label than v. $x = (x_v)_{v \in V} \in L^{|V|}$ is a node labeling that defines a partitioning of V into subsets of nodes S_l assigned to class l such that $\bigcup_{l \in L} S_l = V$. Eq. (2.38) defines the unsupervised partitioning problem where the maximum number of classes in the final labeling is the number of nodes $|V|$ in the graph G. Therefore the coefficients β can depend on the data but are assumed not to depend on prior information about a fixed number of classes L.
A *multicut* on a graph $G = (V, E)$ with a partitioning $\bigcup_{l \in L} S_l = V$ is defined as

$$\delta(S_1, ..., S_k) := \{uv \in E | \exists i \neq j : u \in S_i \text{ and } v \in S_j\} \tag{2.39}$$

Where the sets $S_1, ..., S_k$ are called the *shores* of the multicut. To obtain a polyhedral representation of the set of multicuts on a graph on needs to define incidence vectors $\mathcal{X}(F) \in \mathbb{R}^{|E|}$ for each subset

$F \subseteq E$:

$$\mathcal{X}_e(F) = \begin{cases} 1, & \text{if } e \in F \\ 0, & \text{if } e \in E \backslash F \end{cases} \tag{2.40}$$

then the multicut polytope is given by

$$MC(G) := conv\,\{\mathcal{X}(\delta(S_1, ..., S_k)) | \delta(S_1, ..., S_k) \text{ is a multicut of } G\} \tag{2.41}$$

and the unsupervised image partitioning problem eq. (2.38) can be written as the equivalent multicut problem

$$\min_{y \in MC(G)} \sum_{uv \in E} \beta_{uv} y_{uv} \tag{2.42}$$

defining cycle constraints allows to rewrite eq. (2.42) as the integer linear program (ILP)

$$\min_{y \in [0,1]^{|E|}} \sum_{uv \in E} \beta_{uv} y_{uv} \tag{2.43}$$

$$\text{s.t.} \quad \sum_{uv \in C} y_{uv} \neq 1, \qquad \forall \text{ cycles } C \subseteq E \tag{2.44}$$

The cycle constraints in eq. (2.47) enforce that y lies inside the multicut polytope by guaranteeing that there are no active edges inside a shore. There are many solution methods for this problem. The one used in [21] is based on iteratively solving the ILP in eq. (2.43) without cycle constraints initially, then finding violated constraints in the sense of eq. (2.44), adding them to the ILP and reiterate until there are no more violated cycle constraints.
Violated constraints can be found by projecting a obtained solution y to the multicut polytope and checking for differences in the solution and the projection y'. The projection is achieved by assigning a label to each connected component in $G = (V, \{uv | y_{uv} = 0\})$ which produces a valid partition for which the respective *multicut*, and therefore y', can be obtained easily. If there exists an active edge uv inside the solution that is not active within the projection then this is an edge inside a shore and one of the respective violated cycle constraints is obtained by computing the shortest path between u and v inside the shore and adding the active edge uv to that path, yielding a cycle.

2.6 Principal component analysis

The principal components of a collection of data points can be thought of as the directions in which the variance of the data points is the highest. The magnitude of the variance in the direction of a principal component is referred to as the score of that principal component. All principal component vectors form a orthonormal basis.
Consider a data matrix $X \in \mathbb{R}^{n \times p}$ of n, p-dimensional samples from an arbitrary distribution. The first principal component is

$$w_{(1)} = \operatorname*{arg\,max}_{\|w\|=1} \|Xw\|^2 \tag{2.45}$$

Since this is a convex optimization problem, the solution can be found by finding the stationary points of the Lagrange function

$$\mathcal{L}(w, \lambda) = w^T C w - \lambda(w^T w - 1) \tag{2.46}$$

where $C = X^T X$. Note that C is hermetian. The partial derivatives yield

$$\nabla_w \mathcal{L}(w, \lambda) = 2Cw - 2\lambda w \tag{2.47}$$
$$\nabla_\lambda \mathcal{L}(w, \lambda) = -(w^T w - 1) \tag{2.48}$$

setting eq. (2.47) to 0 yields

$$0 = Cw - \lambda w \tag{2.49}$$
$$Cw = \lambda w \tag{2.50}$$

it strikes that eq. (2.50) is an eigenvalue problem. Since C is hermetian its eigenvectors form a orthonormal basis, therefore eq. (2.47) equals to 0 if w is an eigenvector. To see that the solution to eq. (2.45) is in fact the eigenvector with the largest eigenvalue, one has to substitute eq. (2.50) into eq. (2.46).

$$w^T C w - \lambda(w^T w - 1) = w^T C w = \lambda w^T w = \lambda. \tag{2.51}$$

Since eq. (2.45) is a maximization problem, λ has to be the largest eigenvalue. Therefore the first principal component is the eigenvector $w_{(1)}$ with the largest eigenvalue $\lambda_{(1)}$ of C. $\lambda_{(1)}$ is also referred to as the score of the principal component $w_{(1)}$.

The remaining principal components are given by the other eigenvectors sorted by their eigenvalues.

2.7 Loss functions

This section reviews some important loss functions.

2.7.1 Generalized dice loss

The dice score is a measure based on relative spatial overlap of a prediction and a binary ground truth. The generalized dice loss as defined in [22] defines the dice score for the evaluation of a prediction for multiple classes.
For images with N pixels and L classes, let y_{ln} be the binary ground truth label for a pixel $n \in N$ and a class $l \in L$ and let p_{ln} be the prediction for a pixel $n \in N$ and a class $l \in L$. Then the dice loss is defined as

$$\mathcal{L} = 1 - 2\frac{\sum_{l=1}^{L} w_l \sum_{n=1}^{N} y_{ln} p_{ln}}{\sum_{l=1}^{L} w_l \sum_{n=1}^{N} y_{ln} + p_{ln}} \tag{2.52}$$

where w_l are weights for the corresponding classes.

2.7.2 Contrasive loss

This loss, as defined in [23], applies to the task of instance segmentation in images. A differentiable function predicts points in a feature space that is embedded in a n-dimensional eucledian space. Each predicted point in the embedding space corresponds to a pixel in the image. Ideally the n-dimensional embedding vectors for each pixel are close to each other in the embedding space if the corresponding pixels belong to the same instance and distant to each other if not. If that is the case, a clustering algorithm can assign an instance/cluster to each embedding vector.
The loss penalizing predictions can be thought of as a force that pulls pixel embeddings of the same instance together and pushes those of different instances apart. This loss consists of three additive parts. Assuming C is the number of instances in an image (each label id in the ground truth image represents an instance except for the background label id). N_C is the number of elements in cluster $c \in [1..C]$, x_i is an embedding vector where i denotes a pixel. μ_c is the mean embedding of cluster c, $||\cdot||$ is the $L1$ or $L2$ distance and $[x]_+ = \max(0, x)$. δ_v and δ_d are margins that are used to hinge the pull and push forces. That is, the forces are only applied if the distance that is under consideration is larger than δ_v for intra cluster pulling forces, and smaller than $2\delta_d$ for inter cluster pushing forces. This allows to learn a more expressive embedding space since the points are allowed to move freely if there are no forces applied on them. The three additive terms of the final loss are

- **variance term**
  this exerts a intra cluster pull force that draws pixel embeddings towards the cluster center of their respective instance if the distance to the center is larger than δ_v.

$$\mathcal{L}_{var} = \frac{1}{C}\sum_{c=1}^{C}\frac{1}{N_c}\sum_{i=1}^{N_c}\left[||\mu_c - x_i|| - \delta_v\right]_+^2 \tag{2.53}$$

- **distance term**
  this exerts a inter cluster push force that pushes clusters away from each other by penalizing distances between cluster centers that are smaller than $2\delta_d$.

$$\mathcal{L}_{dist} = \frac{1}{C(C-1)}\sum_{c_A=1}^{C}\sum_{\substack{c_B=1 \\ c_B \neq c_A}}^{C}\left[2\delta_d - ||\mu_{c_A} - \mu_{c_B}||\right]_+^2 \tag{2.54}$$

- **regularization term**
  this is a small pull force that keeps the predicted embedding vectors close to the origin.

$$\mathcal{L}_{reg} = \frac{1}{C}\sum_{c=1}^{C} \|\mu_c\| \tag{2.55}$$

Then the final loss yields

$$\mathcal{L} = \alpha\mathcal{L}_{var} + \beta\mathcal{L}_{dist} + \gamma\mathcal{L}_{reg} \tag{2.56}$$

where α, β and γ are weights for the respective terms.

The behavior for the different terms in the loss is sketched in figure 2.3

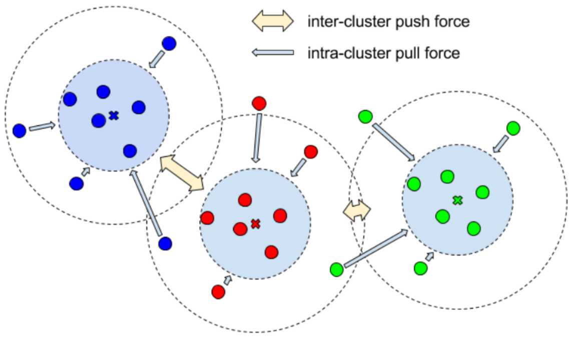


Figure 2.3: [23] As defined by the loss, this are the hinged inter pulling and intra pushing forces acting on points in the embedding space

2.7.3 Triplet loss

This loss, as defined in [24] is related to the contrastive loss (see section 2.7.2) in the sense that it is also defined over data points in an embedding space where the resulting embeddings form ideally per instance clusters. The loss function expects three embedding vectors as an input. An anchor x_i^a, a embedding that is of the same class x_i^p as the anchor and one that is of a different class x_i^n. Then the loss yields

$$\mathcal{L}_{trpl} = \sum_{i}^{N} \left[\|x_i^a - x_i^p\|^2 - \|x_i^a - x_i^n\|^2 + \alpha \right]_+ . \tag{2.57}$$

Again $[x]_+ = \max(0, x)$. N is the number of all possible triplets $(x_i^a, x_i^p, x_i^n) \in \mathcal{T}$ in the embeddings, $\|\cdot\|$ is the $L2$ norm and α is a enforced margin between positive and negative pairs. The behavior during training using $\mathcal{L}_{trpl}$ as a loss is depicted in figure 2.4.

Often times N is very large and the loss calculation requires too many resources. Therefore it is crucial to select triples that are active, namely $\left\{ (x_i^a, x_i^p, x_i^n) \middle| \left[\|x_i^a - x_i^p\|^2 - \|x_i^a - x_i^n\|^2 + \alpha \right]_+ \neq 0 \right\}$.

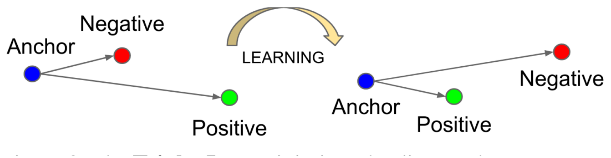


Figure 2.4: [24] Position of triplets in the embedding space before and after a optimization step

However different selection strategies end in very different results regarding training time and convergence to satisfactory optima. E.g. picking only triplets that produce a high loss value can result in a training converging quickly into bad local optimum.
To prevent the embeddings from diverging, a regularizing term is added in order to constrain the embedding vectors to the unit hypersphere surface.

$$\mathcal{L}_{reg} = \sum_{i}^{M} \frac{1}{2}(\|x_i\| - 1)^2 \tag{2.58}$$

Here M is the number of pixels in the image. The final loss therefore yields

$$\mathcal{L} = \mathcal{L}_{trpl} + \beta \mathcal{L}_{reg} \tag{2.59}$$

where β is a scalar weight for the regularizer.

2.8 The reparameterization trick

The reparameterization trick was proposed in [25]. Since then it is commonly used to make a sampling process, used to approximate an expectation, differentiable with respect to the statistics of the corresponding distribution. Not all distributions have definitions for their reparameterization yet but for some basic ones such as the normal distribution it exists.
The problem is stated as follows. Given a realization y of a continuous random variable and its conditional distribution $y \sim q_\theta(y|x)$ where θ are the parameters of the distribution. Then the Monte Carlo estimate of the expectation $\mathbb{E}_{y \sim q_\theta(y|x)}[y] \approx \frac{1}{N}\sum_{n=0}^{N} y$ is not differentiable w.r.t. θ, because the underlying sampling process, generating y is not differentiable.

If a vector valued and differentiable function $g_\theta(\epsilon, x)$ where ϵ is an auxiliary variable distributed by $\epsilon \sim p(\epsilon)$, can be found such that $\mathbb{E}_{y \sim q_\theta(y|x)}[y] = \mathbb{E}_{\epsilon \sim p(\epsilon)}[g_\theta(\epsilon, x)]$ then the Monte Carlo estimate of the expectation $\mathbb{E}_{\epsilon \sim p(\epsilon)}[g_\theta(\epsilon, x)] \approx \frac{1}{N}\sum_{n=0}^{N} g_\theta(\epsilon_n, x)$ is differentiable w.r.t. θ.

For the realization of a Normal distributed random variable $y \sim \mathcal{N}(\mu, \sigma^2)$ such a function would be

$$g(\epsilon, \mu, \sigma) = \frac{1}{\sigma}(\epsilon - \mu), \qquad \epsilon \sim \mathcal{N}(0, 1) \tag{2.60}$$

this is useful in RL since there is an expectation, approximated by the Monte Carlo estimate that needs to be differentiable with respect to the parameters of the policy which was sampled from.

2.9 The Heugh transform

The Hough transform for the application of detecting curves in an image was first introduced by [26]. Reducing this problem to finding lines in a set of N points $(x_i, y_i) \in \Omega$ where $x_i \in [0..X]$, $y_i \in [0..Y]$ and $i = 0, ..., N$. X is the size in the first x-dimension and Y the size in the second y-dimension of the image.
Assuming each point lies on a line and each of those points is transferred to the latent space of lines defined by the *normal parameterization* of a line. As shown in figure 2.5 this representation is defined by the angle θ of the line's normal and its algebraic distance ρ from the origin.

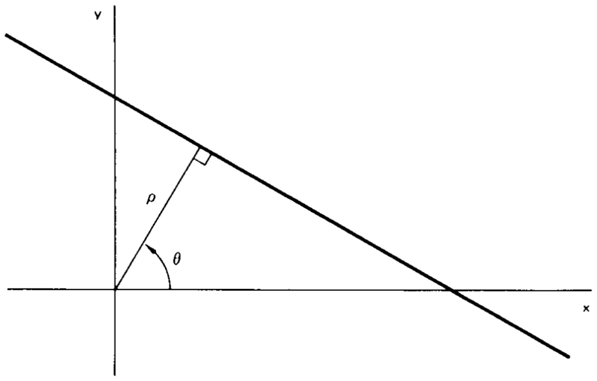


Figure 2.5: [26] latent variables of a line

Transforming all points in Ω to sinusoidals in the θ-ρ plane yields

$$\rho_i(\theta) = x_i \cos\theta + y_i \sin\theta, \qquad i = 0, ..., N. \tag{2.61}$$

One property of this point to curve transformation is, that an intersection of two or more of those functions means, that there is a line in the image plane that is going through all points the functionals correspond to.
Therefore lines are found by discretizing the θ-ρ plane and counting in each cell the intersections of functions $\rho_i(\theta)$ going through that cell.
Thresholding leaves only the cells with a count of intersections higher than the thresholding parameter and each of those cells can be transformed to a line in the image space.
This idea can be extended to the detection of circles resulting in the Circle Heugh transform (CHT). A parametric representation for a circle in the x-y plane is given by

$$c^2 = (x-a)^2 + (y-b)^2. \tag{2.62}$$

Each point (x_i, y_i) in the image place can be transformed to a surface in the a-b-c parameter space.

$$c_i^2(a,b) = (x_i - a)^2 + (y_i - b)^2 \tag{2.63}$$

this surface correponds to a right circular cone. Again intersections of multiple functionals $c_i^2(a,b)$ means that the corresponding points in the image plane correspond to the circle defined by the intersection point in the a-b-c parameter space. The same procedure of discretizing and counting as for the line detection method results in the CHT as defined in [26].

Methods

3.1 Using RL for the image segmentation task

In this section the task of image segmentation is fit to the RL framework.
The agent takes the role of predicting action distributions where sampled actions perform some kind of change to the input of a segmentation algorithm. Rewards can be calculated from the resulting segmentation. The segmentation algorithm would therefore be part of the environment.
In order to optimize the agents parameters by the RL losses, the segmentation algorithm can be non differentiable.
Common RL problems are usually problems that are easy to evaluate like determining the winner of a board game or evaluating a robots position relative to a target position. Also often intermediate results are not rewarded at all or given a small negative reward in order to put pressure on the fast arrival of the final state.
Typically in RL there is a physical environment that can be measured by sensory input a reward calculation based on that signals can take place. For the task of image segmentation such a environment can not be found or measured. The labeling would have to be projected from the image plane into the ”real world” where measurements could take place. Of course it is not clear how to do that, therefore the reward generation has to be based on a virtual model of the environment. A segmentation can only be evaluated if enough information on the objects within the input image is known a priori. E.g number of object instances per object class, position, texture, shape etc..
Given such an evaluation and given that the function that is optimized by the agent is capable of capturing the underlying probability distribution of data label pairs, the described RL setting is theoretically able to learn the task of image segmentation in a fully unsupervised way.

3.2 Using RL for pixel affinity predictions

This work started with the idea of an RL setting where an agent manipulates affinities between pixel pairs. Those affinities form an input to the Mutex Wateshed algorithm (see section 2.4).
The edge set that the Mutex Watershed typically works with are short range attractive edges and long lange repulsive edges where the length of the repulsive edges is dependent on the object sizes in the image. That means that there should be exactly one attractive edge for every directly neighboring pair

of pixels and some long range repulsive edges. Defining the problem by manipulating affinities leads to a hard task since there are simply so many.
RL methods learns from rewards resulting from actions and an initial state. To achieve convergence, initially there need to be some rewards of a high value, that means that the actions taken lead to a fairly good segmentation. However when starting learning a network from noise it outputs completely arbitrary actions that unlikely lead to a high reward because the action space is simply too large.
The RL-typical bootstrapping works only if there is a meaningful gradient in the reward signal, even for random actions. It is not uncommon in RL to have a large action space while the reward is a single scalar value. If it would be possible to calculate more meaningful rewards, say per subregion in the image, one could compute a RL loss term for each of those subregions only considering the actions that manipulated affinities within this region.

To downsize a image segmentation problem, it is common to work with superpixels [21] rather than with pixels. A superpixel segmentation or oversegmentation is usually achieved by watersheding or smoothing algorithms. Using Mutex Watershed one can simply globally decrease the edge weights of the attractive edges in order to arrive at an oversegmentation. Starting from such an oversegmentation and assuming that the ground-truth segmentation is a partitioning of the superpixels, focusing solely on merging and unsmerging superpixels would be sufficient. Concerning the Mutex Watershed algorithm, a merge of two superpixels would be done by turning all the repulsive edges between them into attractive ones and vice versa for unmerging. Therefore one needs a decision/action for every neighboring pair of superpixels.
However there are two problems with this, first one can only perform hard merges and unmerges which leads likely to contradictions for example consider 3 adjacent superpixels and there are 2 merges and 1 unmerge predicted. This is of course nothing Mutex Watershed cannot handle but the result is likely to be not a partitioning of the superpixels and therefore not the intended result.
The second issue is that for CNNs it is difficult to make predictions on affinities between adjacent superpixels due to the irregularity of the region adjacency graph of the superpixels.

3.3 Overview over the proposed pipeline

To overcome the two problems stated in the previous section, the following is done. Making hard decisions on merges and unmerges between superpixels is bad because of the arising contradictions where cycle constrains easily become violated but also because a hard thresholding of predictions has to be performed which takes away all the information that lies in the uncertainty of this predictions.
Both those issues can be overcome by predicting affinities on the edges between neighboring superpixels which can be transformed into costs and a minimum cost multicut (see section 2.5) of the superpixel graph can be computed. This leverages the uncertainty of the network predictions and overcomes violated cycle constraints.
The second issue that arises from the irregularity of the superpixel graph can be overcome by using a graph neural network (GCNN)that predicts edge weights between superpixels by performing graph convolutions (see section 2.3). However this generates the problem that there have to be regular representations for superpixels. Of course one solution would be to have two GCNNs, one for intra superpixel convolution and pooling, down to a vector representation of each superpixel and one for inter superpixel convolution using the vector representations. The problem here is the intra superpixel GCNN. Regular graph covolutions on image data makes not a lot of sense because it is not able to capture spatial infor-

mation like object shapes very good. There are some proposals [27] that introduce spatial dependent features to graph cnvolutions. However considering the power of CNNs with regular images, it makes more sense to use those in favor of GCNNs.
Getting superpixel representations from a CNN can be done using a embedding network that predicts pixel embeddings by minimizing a contrastive loss (see section 2.7.2). Still assuming, that the ground truth segmentation is a partitioning of the superpixels, it is safe to assume similar pixel embeddings in terms of the distance used in the loss for all pixels within a superpixel. Therefore one can average over all pixel embeddings within a superpixel in order to arrive at a vector representation that is used as node features in the following GCNN.
The described model is depicted in figure 3.1.
The following sections go into detail of every part in the sketch in figure 3.1

3.4 The pipeline in reinforcement learning terminology

As shown in figure 2.1, in all RL problems there are two main instances acting and sending signals to each other. A closer look at each instance and signal as well as their definition within this context of the task is given below.

3.4.1 The state

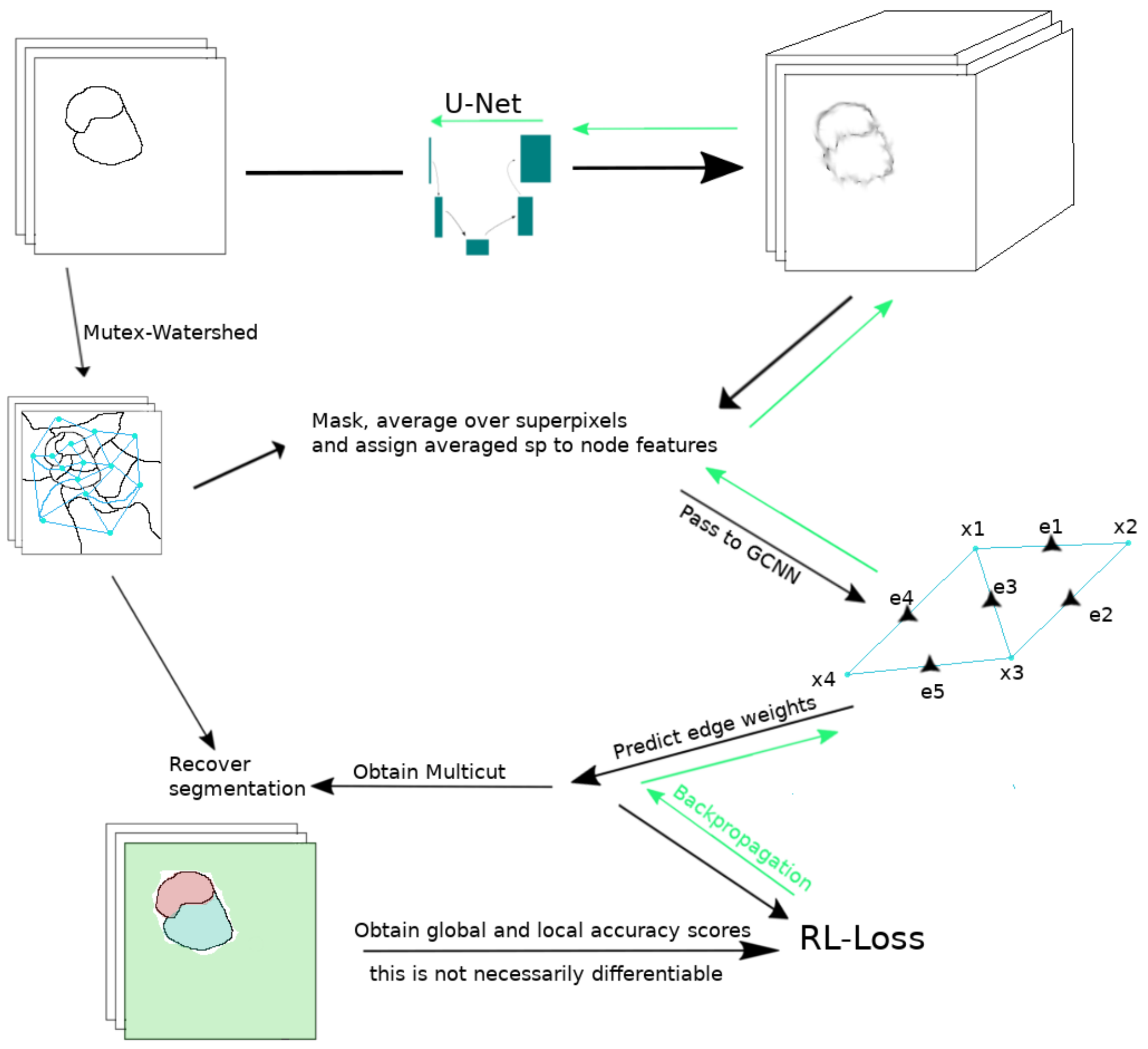
The current state is defined as the concatenation of the raw data rw, a superpixel segmentation sp of the raw data and a final segmentation seg_t that is a partitioning of the superpixels $s_t = [rw, sp, seg_t]$. Therefore the only part of the state that is ever updated during training on a single image rw is seg_t. That means, the only part of the state space that needs to be explored by the agent is defined by seg_t.

3.4.2 The actions

The formulation of the actions depend on the RL algorithm used. There are algorithms for discrete and for continuous action spaces. The predicted target are probabilities for merge affinities, therefore values between in the interval $[0, 1]$. So it would be natural to predict continuous actions in $[0, 1]$ that can then directly be taken for the target merge affinity.
However most algorithms with a policy based on value functions like Q-learning are defined only for discrete actions. Algorithms incorporating policy gradients can usually be defined for both discrete and continuous action spaces. When working with discrete actions there are two possibilities for their definition.

- one possibility is to directly predict values or the statistics of a categorical distribution for discretisized affinities. Depending on the degree of discretization this might lead to a large complexity. Also this makes it possible to diverge away from an initial state very fast, namely within one step. If there is an initial state which is likely already close to the ground truth this is not favorable.

- the other possibility is to predict values for actions that are operating on the current state of edge values. E.g a state action value to add or subtract a fixed scalar c. Here the level of discretization depends on the magnitude of c which does not change the memory complexity of the output but



U-Net
Mutex-Watershed
Mask, average over superpixels
and assign averaged sp to node features
Pass to GCNN
x1
e1
x2
e3
e4
e2
x4
e5
x3
Predict edge weights
Recover
segmentation
Obtain Multicut
Backpropagation
Obtain global and local accuracy scores
this is not necessarily differentiable
RL-Loss

Figure 3.1: A rough sketch of the proposed pipeline. Starting from raw data (top left), a superpixel graph is obtained with the Mutex Watershed algorithm 1 and pixel embeddings (top right) are obtained with an embedding network. With node features computed as the average pixel embedding per superpixel a GCNN predicts logits on the edges of the superpixel graph. The logits are used to compute chances which in turn are used to compute costs based on which a min cost multicut of the superpixel graph is computed from which a segmentation is obtained. This segmentation is then evaluated and a reward is produced which is then used in the RL loss.

has a direct affect on the number of steps that are necessary to arrive at a target state. This method also favors a more continuous and controllable slow divergence from an initial state.

3.4.3 The reward

The reward is crucial for the whole training behavior. The right modeling of the reward signal principally decides for fast convergence to the target solution and the avoidance of converging into local optima. If a set of raw image and label pairs is available it makes sense to derive a ground truth value for every edge in the superpixel graph. Then the reward is per edge simply by the distance of the current edge state to the ground truth edge. This is the most accurate reward that can be obtained. Although it should be considered, that this version comes with the drawback of generating large variance in the updates of the state action value function. A other possibility would be the prediction of a single state action value. This certainly smooths out any variance present in the single predictions but it is also too coarse when it comes to larger superpixel graphs. This problem is discussed in more detail in section 3.8.

3.4.4 The agent

The role of the agent is taken mainly by the embedding network and the involved GCNNs. Its input is the input to the embedding network which is the current state s_t. It outputs statistics of a probability distribution per edge. Depending of the choice of algorithm this can be arrays of probabilities for a categorical probability distribution in the case of discrete actions, or the statistics of a probability density function in the case of continuous actions. The latter requires a sigmoid transformation of the samples to guarantee they fit the requirement of being in the interval $[0, 1]$.

3.4.5 The environment

The environment receiving actions a_t that act on a state s_t producing s_{t+1} as well as a reward r_t. Therefore it mainly consists of the Multicut algorithm updating the state based on the actions and of some evaluation scheme for the new state in order to calculate rewards. This scheme can be based on ground truth segmentations or on prior knowledge or both.

3.4.6 The problem of local optima

Usually the ground truth of edge weights reveals an imbalance in attractive and repulsive edges. Due to the nature of an oversegmentation there are a more attractive edges than repulsive edges. This imbalance generates the local optimum of exclusively attractive edges. RL algorithms are known of converging to local optima and even strong perturbations in the rewards might not prevent this.
This kind of local optimum is known in image segmentation problems and has been addressed by many losses like focal loss or dice loss. The dice score can directly be transferred to edge value predictions. The problem here is that this produces a single scalar reward. This is a problem because there can be a few hundred or even thousands of edges within a superpixel graph. Having a scalar reward signal is to vague to propagate a meaningful gradient to actions on single edge values.
Most RL benchmarks incorporate action dimensions that are less than 10 which is a small enough number to have one global reward value.
Transferring this to the prediction of edge values on a graph would be a single state action value per subgraph of roughly size 10. This has the advantage of training a state action value function globally for the predictions on each subgraph. Therefore, if ground truth is available, one can use the dice score over a subgraph as a reward signal. This smoothes out class inbalances as well as variances in single edge state action values. This method is shown in figure 3.2. Figure 3.2 also sketches a method to compute per subgraph rewards in an unsupervised fashion.
The subraphs can and should overlap in order to smooth out variances in the reward signal. Since a GCNN is used for the agents prediction it makes sense to select subgraphs with a high density which increases the information flow in the graph convolution.

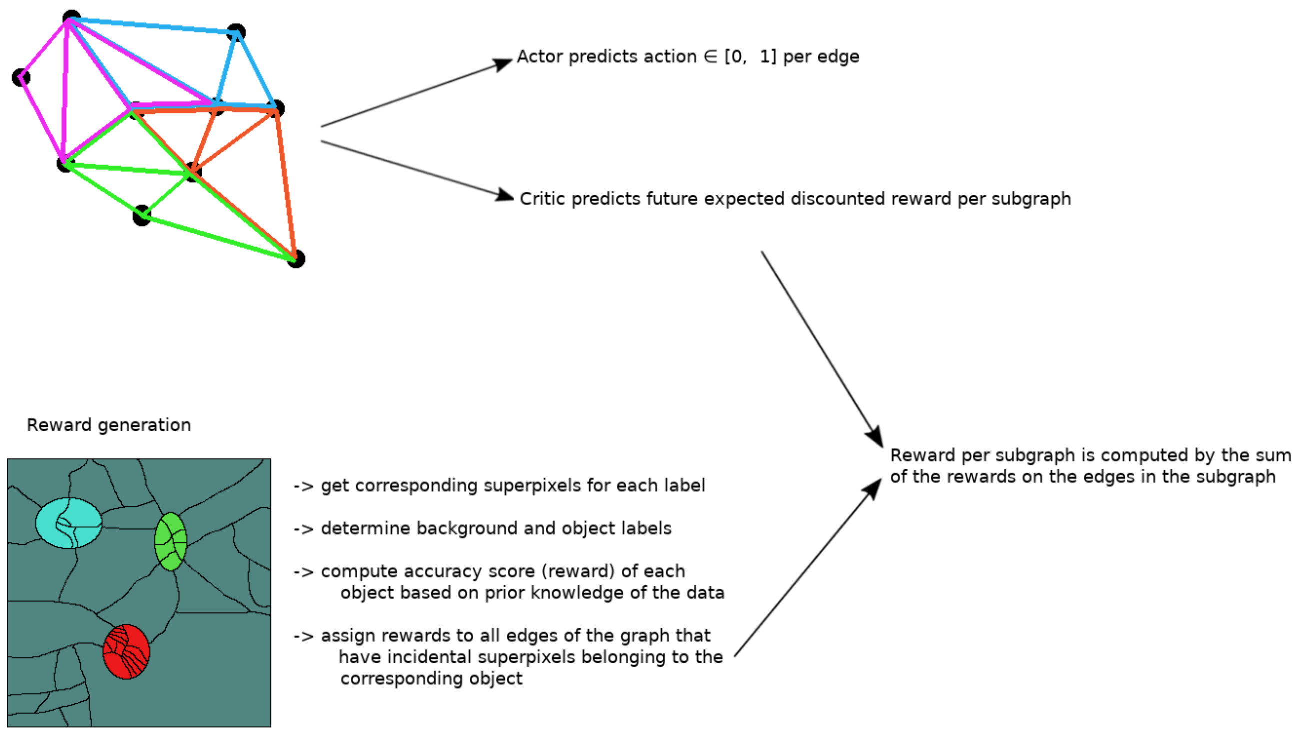

Actor predicts action ∈ [0, 1] per edge
Critic predicts future expected discounted reward per subgraph
Reward generation
-> get corresponding superpixels for each label
-> determine background and object labels
-> compute accuracy score (reward) of each object based on prior knowledge of the data
-> assign rewards to all edges of the graph that have incidental superpixels belonging to the corresponding object
Reward per subgraph is computed by the sum of the rewards on the edges in the subgraph

Figure 3.2: A rough sketch of the reward calcuation on subgraphs with 7 edges and the resulting losses in an actor critic setting

3.4.7 Definition of the RL algorithm

Most RL algorithms like Q-learning operate in discrete action spaces. The advantages of that have been mentioned. However for the prediction of probabilities it is more natural to use a continuous action space. The drawback is that it is possible to diverge fast from an initial state. Such a divergence can easily be penalized by the reward signal e.g by calculating the distance of current state to the initial state and subtracting that distance from the reward when it surpasses a certain margin.
Therefore the SAC algorithm 2.2.5 is used which is a comfortable choice because it is defined for continuous actions and, by construction, it takes care of sufficient exploration.
It is easy to adjust eq. (2.26) in section 2.2.5 for predictions on subgraphs. Considering a subgraph size of 10 and selecting the Normal distribution for the policy π. A policy GCNN predicts for every edge mean μ and variance σ^2 of its respective action distribution.
Drawing a reparameterized 2.8 sample from this distribution follows a Sigmoid transform of the sample. Since the Sigmoid function is a diffeomorphism, the change of variables formula [11] can be applied and allows for the computation of the probability density of the transformed sample.

The joint probability density of all actions per subgraph is given by the product of their respective densities. Therefore eq. (2.26) in section 2.2.5 is rewritten as

$$\nabla_\theta \bar{\mathcal{L}}_{actor} = \nabla_\theta \sum_{sg \in G} \left[\alpha \sum_{a_t \in sg} log(\pi(a_t|s_t)) - Q_\pi(s_t, a_t)_{sg} \right] \tag{3.1}$$

Here G is the set of sets that contain the respective actions for each subgraph. $Q_\pi(s_t, a_t)$ is a function mapping the current state action tuple (s_t, a_t) to $\mathbb{R}^n$ where n is the number of subgraphs in s_t. $Q_\pi(s_t, a_t)_{sg}$ denotes the predicted state action value for subgraph sg.
Eq. (2.22) in section 2.2.5 does not change considering the rewards are per subgraph as well.
Additionally to the optimization techniques within the SAC 2.2.5 algorithm, prioritized experience replay 2.2.6 is used.
RL problems are usually of the form that, starting from an initial state s_0, multiple steps lead to an end state s_T. Since here, directly sampling affinity values from the policy makes it possible to reach any state within one step, it is sufficient to define $T = 1$. Stopping after one step has the advantage that the state action function becomes much simpler. The feature to be able to reach any state from any other state makes all parts of the state that are dependent on t redundant. Therefore s_t can be redefined to $s_t = s = [rw, sp]$. Setting $T = 1$ the loss in eq. (2.22) in section 2.2.5 becomes

$$\mathcal{L}_{critic} = \frac{1}{2}(Q_\pi(s_t, a_t) - r_t)^2 \tag{3.2}$$

While this yields a simple action state function to approximate, there is also a point in saying that this definition is not a ”real” RL setting anymore. However the RL loss still gives the advantage that the supervision signal (here the reward), does not have to be differentiable, which is the main justification for this pipeline.

3.5 Obtaining superpixels from mutex watershed

There are many algorithms that can be used to compute a superpixel segmentation. Mutex Watershed (see section 2.4) is a very flexibly method because it relies on affinities and there are many methods to obtain those like learned affinities or directly based on pixel intensities.
With a CNN one can train a very generic affinity predictor. Globally scaling repulsive and/or attractive edge weights allows for control over the granularity of the superpixels. In addition to that, Mutex Watershed is a comparatively fast algorithm under certain conditions that are mainly dependent on the amount of strong repulsive edges.
Having the superpixel segmentation, it is straight forward to generate a region adjacency graph from that.

3.6 The embedding network

The embedding network was realized with a U-Net architecture [28] where the final layer outputs 16 channels. There are several ways to train the embedding network. If ground truth is available it can be

trained prior to the SAC training with the contrastive loss 2.7.2.
If there is no ground truth there are three different bootstrapping procedures that can be included to the training of the SAC.

- The first one is the optimization of the embedding network through the RL losses. This is possible, because the masking of the embeddings by the superpixels followed by the averaging procedure is differentiable w.r.t. the pixel embeddings. The embedding network forms one common feature extractor for actor and critic. Intuitively it should be more stable if it is optimized only by one of the two optimizers. Since the actor smooths out variances and wrong probabilities in the Gibbs distribution of the critics prediction the better choice for optimizing the embedding network should be the backpropagated gradient of the actors loss function.

- The second procedure is a mixture of contrastive loss 2.7.2 and triplet loss 2.7.3 based on the action predictions of the agent (defined by the mean of the predicted policy). Hereby all pixel embeddings belonging to the same superpixel should be close to each other in the embedding space. This is namely eq. (2.53) in 2.7.2 where each superpixel corresponds to one cluster.
  For the inter pushing *or* pulling forces, a triplet loss of the form in eq. (2.57) is used. The triplets can be found by evaluating the action predictions of the agent.
  A positive superpixel pair is defined by the existence of a path between those superpixels in their region adjacency graph that has only action values on the edges that are below a certain threshold tl. On the contrary a negative pair is one that has at least one path between the two superpixels where at least one action value on the edge is above a certain threshold th.
  Attractive paths between superixels in the region adjacency graph can be found with Dijkstra's algorithm by setting all weights (means of the action distributions) in the weighted adjacency matrix that are above tl to $+\infty$ and then accepting all paths in the result that are not $+\infty$. In contrary, repulsive paths between superpixels can be found by setting all weights in the weighted adjacency matrix that are above th to $+\infty$ and accepting all paths in the result that are $+\infty$. This selection of triplets does not protect from contradictions in the underlying segmentation. A valid underlying segmentation can be obtained by removing superpixel pairs that violate cycle constraints similar to eq. (2.36) in section 2.4.

- The third method is in its result very similar to the second method, but here the triplet selection is based on the final segmentation seg_t. This method does not suffer from violated cycle constraints and is chosen in favor of the second method. However keep in mind, that this method is based on the sampled actions and not on the mean action of the policy as method two. To get the result based on the expected actions there needs to be an additional run of the Multicut algorithm.

The last two methods should happen interchangeably to the SAC training, where the optimization step frequency of the embedding network should be much lower and the loss should be over a "larger" mini batch considering the variance in early SAC predictions.

3.7 The actor critic networks

In this section the involved GCNN's are discussed in detail. There is one network predicting the statistics for the policy. This is referred to as the actor network. Let the directed region adjacency graph of the superpixel segmentation be $G = (V, E)$. The conversion from the undirected region adjacency graph to a directed graph is achieved by replacing each undirected edge by a pair of opposing directed edges with the same incidental nodes. The implemented graph convolution on G for K convolution iterations is defined by the update functions

$$\vec{e}^{1}_{ij} = \sigma\left(\phi_0\left(\vec{x}^{0}_{i}, \vec{x}^{0}_{j}\right)\right) \tag{3.3}$$

$$\vec{x}^{1}_{i} = \sigma\left(\gamma_0\left(\vec{x}^{0}_{i}, \frac{1}{deg(\mathcal{N}(i))}\sum_{j\in\mathcal{N}(i)} \vec{e}^{1}_{ij}\right)\right) \tag{3.4}$$

For the first iteration as there are no edge features initially. The superscripts refer to the convolution step and x^0_i is the node feature vector obtained by avaraging the pixel embeddings for each superpixel $i \in V$. ϕ^k and γ^k are multi layer perceptrons at step k and σ is an elementwise non linear function. $\vec{e}^{k}_{ij}$ is the edge feature vector at update step k for edge $(ij) \in E$ where $i \in V$ is always the node index of the sink node and $j \in V$ the node index of the source node. x^k_i is the node feature vector at update step k for node $i \in V$.
The following $K - 2$ iterations are defined by the update functions

$$\vec{e}^{k+1}_{ij} = \sigma\left(\phi_k\left(\vec{x}^{k}_{i}, \vec{x}^{k}_{j}, \vec{e}^{k}_{ij}\right)\right) \tag{3.5}$$

$$\vec{x}^{k+1}_{i} = \sigma\left(\gamma_k\left(\vec{x}^{k}_{i}, \frac{1}{deg(\mathcal{N}(i))}\sum_{j\in\mathcal{N}(i)} \vec{e}^{k+1}_{ij}\right)\right) \tag{3.6}$$

$$\text{for } k = 1...K - 2 \tag{3.7}$$

The final iteration is defined by the update function

$$\vec{e}^{K}_{ij} = \sigma\left(\phi_{K-1}\left(\vec{x}^{K-1}_{i}, \vec{x}^{K-1}_{j}, \vec{e}^{K-1}_{ij}\right)\right) \tag{3.8}$$

Where the the number of elements in the output vector $\vec{e}^{K}_{ij}$ corresponds to the number of the required scalar values that define the distribution used to describe the policy. For this pipeline, mean and variance of a Normal distribution are predicted.

The GCNNs approximating the state action values are referred to as the critic networks. There are two of them of equal architecture but distinct parameters, incorporating Double Q-Learning (see section 2.2.6). The first convolution step in the critic network architecture is defined by the update step

$$\vec{e}^{1}_{ij} = \sigma\left(\eta_0\left(\vec{x}^{0}_{i}, \vec{x}^{0}_{j}, \vec{a}_{ij}\right)\right) \tag{3.9}$$

$$\vec{x}^{1}_{i} = \sigma\left(\psi_0\left(\vec{x}_{i}, \frac{1}{deg(\mathcal{N}(i))}\sum_{j\in\mathcal{N}(i)} \vec{e}^{1}_{ij}\right)\right) \tag{3.10}$$

Where $\vec{a}_{ij}$ is the action a_t corresponding to the edge ij (and to the edge ji in the digraph). η^k and ψ^k

are multi layer perceptrons at update step k. The following $M-2$ convolution steps are the updates

$$\vec{e}_{ij}^{k+1} = \sigma\left(\eta_k\left(\vec{x}_i^k, \vec{x}_j^k, \vec{e}_{ij}^k\right)\right) \tag{3.11}$$

$$\vec{x}_i^{k+1} = \sigma\left(\psi_k\left(\vec{x}_i^k, \frac{1}{deg(\mathcal{N}(i))}\sum_{j\in\mathcal{N}(i)} \vec{e}_{ij}^{k+1}\right)\right) \tag{3.12}$$

$$\text{for } k = 1...M-2 \tag{3.13}$$

and the final iteration, again only updating edge features

$$\vec{e}_{ij}^{M} = \sigma\left(\eta_{M-1}\left(\vec{x}_i^{M-1}, \vec{x}_j^{M-1}, \vec{e}_{ij}^{M-1}\right)\right) \tag{3.14}$$

Following that, the graph G with edge features e_{ij}^M is split into unconnected subgraphs $SG = (SV, SE)$ such that each connected component in SG is a subgraph in G with exact l edges and the union of all those subgraphs covers G completely. It is continued with the edge features only because, for the state action value approximation, only information of affinities between superpixels is important. The first update on the subgraphs is defined by the update

$$\vec{x}_i^1 = \sigma\left(\beta_0\left(\frac{1}{deg(\mathcal{N}(i))}\sum_{j\in\mathcal{N}(i)} e_{ij}^M\right)\right) \tag{3.15}$$

With $ij \in SE$ where $i \in SV$ is always the node index of the sink node and $j \in SV$ the node index of the source node. This is followed by the $N-2$ updates

$$\vec{x}_i^{k+1} = \sigma\left(\beta_k\left(\vec{x}_i^k, \frac{1}{deg(\mathcal{N}(i))}\sum_{j\in\mathcal{N}(i)} \sigma\left(\delta_k\left(\vec{x}_i^k, \vec{x}_j^k\right)\right)\right)\right) \tag{3.16}$$

$$\text{for } k = 1...N-2 \tag{3.17}$$

Again, δ^k and β^k are multi layer perceptrons at step k. The last graph convolution iteration is the edge feature update

$$\vec{e}_{ij} = \sigma\left(\delta_{N-1}\left(\vec{x}_i^{N-1}, \vec{x}_j^{N-1}\right)\right) \tag{3.18}$$

the scalar state action value per subgraph sg is obtained by

$$Q\pi(s_t, a_t)_{sg} = \gamma_Q\left(\frac{1}{l}\sum_{ij\in sg} \vec{e}_{ij}\right) \tag{3.19}$$

where γ_Q is a multi layer perceptron outputting a scalar value. $\vec{e}_{ij}$ is dependent on (s_t, a_t) by the introduced upstream pipelines in the way that x_i^0 depends on s_t and the embedding network and a_t are reparameterized samples from distributions dependent on the predicted statistics in eq. (3.8).

3.8 Finding subgraphs

The selection of subgraphs has the only hard restrictions that all selected subgraphs should consist of l edges and that the union of subgraphs should cover the region adjacency graph of the superpixel segmentation. Additional to that it is encouraged to select overlapping subgraphs and subgraphs of high density. The latter can be rewritten by finding subgraphs with the smallest possible node counts. Also the overlaps should not be too large such that the result is still a feasible number of subgraphs.
Finding the densest subgraph of size l in a graph $G = (V, E)$ is in general a NP-hard problem [29]. The implemented algorithm is a fast heuristic that leverages the properties of the region adjacency in G where one can assume a relative even density over the whole graph.
The heuristic starts by sampling a random edge $(ij) \in E$, that is not contained in any subgraph so fara, adds it to the new subgraph $SG = (SV, SE)$ and pushes its incidental nodes to a priority queue pq with starting priority value 0 (smaller value corresponds to higher priority). Nodes are drawn from pq until the respective subgraph has the right amount of edges. Drawing a node n from pq is followed by iteratively verifying if there is a node m s.t. $(nm) \in E$ and $m \in SV$, if yes than (nm) is added to SG and m is added to pq with a priority that is incremented by 1. If not all to n adjacent nodes where accepted and the corresponding edges added to SG, the priority of n is decreased by the amount of edges that where added and pushed into pq again.
The next iteration starts by drawing the next node from pq. If all elements in pq where drawn without an edge being added to SG and SG being still incomplete, the last drawn nodes n last examined neighbour m is added to pq and the edge nm is added to SG.
This is repeated until all subgraphs cover G. The worst case of this method would be tree-like subgraphs overlapping completely except for one edge. However for region adjacency graphs this is unlikely too happen and can be ignored.
The pseudo code for the described heuristic is given in algorithm 2.

Algorithm 2: Dense subgraphs in a rag

```
   Data: $G = (V, E), l$
   Result: subgraphs by sets of $l$ edges
 1 Initialization:$SG = \emptyset$;
 2 while $E \setminus SG \neq \emptyset$ do
 3     pq = PriorityQueue;
 4     prio = 0;
 5     n_draws = 0;
 6     $sg = \emptyset$;
 7     $i, j = (ij)$ s.t. $(ij) \in E \setminus SG$;
 8     pq.push($i$,prio);
 9     pq.push($j$, prio);
10     $sg = sg \cup (ij)$;
11     while |sg| < $l$ do
12         $n$, n_prio = pq.pop();
13         n_draws ++;
14         prio ++;
15         $adj = \{(nj) | \exists (nj) \in sg\}$;
16         forall $(nj) \in adj$ do
17             pq.push(j, prio);
18             $sg = sg \cup (nj)$;
19             n_draws = 0;
20         if $|adj| <$ deg($n$) then
21             n_prio -= $(|adj| - 1)$;
22             pq.push($n$, n_prio);
23         if n_draws = pq.size() then
24             $j \in \{j | (nj) \in E\}$;
25             pq.push($j$, prio);
26             $sg = sg \cup (nj)$);
27     $SG$ = $sg \cup SG$
28 return $SG$
```

3.8.1 Thoughts on dependence

The actors predicts a univariate probability distribution on each edge in the superpixel graph. Looking at the predicted statistics for this distributions as random variables, one can consider the predicted distribution itself as a random variable. Since all predictions are dependent on each other through the graph convolutions there is an underlying multivariate probability distribution for the statistics on all the edges. This makes sense intuitively because actions should depend on broader local neighborhoods in the graph if not on the whole graph.
The same holds therefore for the update of the critics networks. Since the subgraphs are moved into the batch dimension (see section 3.9.1) they have no dependence to each other through the convolution on the subgraphs. This is the reason for the upstream network doing the convolution on the whole graph.

Also the overlaps of the subgraphs help building interdependence between subgraphs.

3.9 Technical details

This section reviews some technical details of the implementation of the pipeline. Except for algorithm 2 and some more helper functions for operations on graphs that have been implemented in c++ all has been written in python under heavy use of the libraries pytorch, numpy, scipy and skimage. The c++ implementations are called from the python interpreter using the pybind11 interface together with the xtensor libraries. The GCNNs have been implemented using the library pytorch-geometric, introduced in [30].
pytorch multiprocessing is used for parallelization and synchronization in the fashion of the A3C (section 2.2.6). After each update step through graph convolutions in section 3.7 node and edge features are normalized by a Batch Normalization layer [31].
The embedding space is $\mathbb{R}^{16}$ and the node features in section 3.7 are in the embedding space $x_i^0 \in \mathbb{R}^{16}$. The number of convolution iterations in section 3.7 is $K = 5$ as well as for $M = 5$. The number of convolution iterations for the subgraph GCNNs is $N = 10$ because here it is important that the information in the graph is spread over all nodes because of the global pooling operation afterwards. The multilayer perceptrons in eq. (3.3-3.14) in section 3.7 all have 1 hidden layer. The first mlp in each of the actors convolution blocks has $in = 16 \cdot 2$ input features and $in \cdot 10$ output features. All the following except the last one have $in \cdot 10$ input and $in \cdot 10$ output features. The last one squashes the feature vectors again to 16 dimensions for node and edge features. For the critics first networks that operate on the whole graph the same structure holds except, here the input features are $in = 16 \cdot 2 + 1$ since the actions a_{ij} are concatenated to the node features x_i^0.

3.9.1 Batch processing

Batching a set of irregular graphs can be achieved by converting the set of graphs into one large graph whose connected components form each graph in the batched set. Doing graph convolution on the large graph yields the same result as doing the convolution on each of the batched graphs separately with the difference that the operation in performed in parallel. The same property is used for the convolution on the subgraphs.

Experiments and results

This chapter evaluates the proposed method on a toy example which consists of 128 by 128 pixel images with 6 to 10 disks with radii between 6 and 8 that are non overlapping but might be touching. The images are single channel and the disks are seperating themselves from the background by phase shiftet interfering frequencies. An example is depicted in figure 4.1.

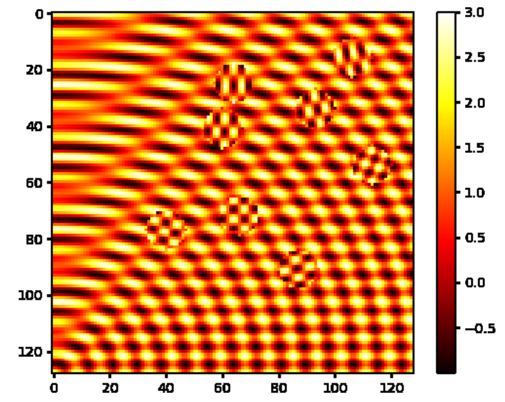


Figure 4.1: Example raw image

The incentive is to do instance segmentation for the disks. That is labeling each of the discs in a unique object ID. There where some foreground background segmentations created along with the raw data. This segmentations are used to train the embedding network as described in section 3.6. Note that training the embedding network on foreground background images with contrasive loss generates two clusters only in the embedding space, one for foreground and one for background. For the node features it is only important to be distinctive between those two classes because that is all it needs for a merge/unmerge decision.
The pixel embeddings are visualized by their values in the first three principal components (see section 2.6) of the whole set of pixel embeddings. The three values per pixel are converted to an rgb image. Note that for most trained pixel embeddings the majority of the information is contained within their first three principal components. In figure 4.2 this method of visualization was used on the pixel embeddings of the example image.

The affinities that are needed for the mutex watershed algorithm are calculated based on the fist and second gradient of the raw data for short range attractive edges. For long lange repulsive edges the concept of the first and second gradient computation by forward differences is used for pixelwise differ-

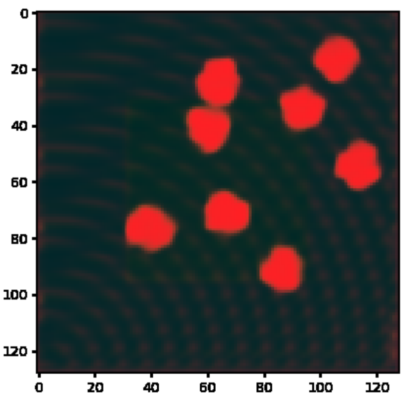


Figure 4.2: Values in first three principal components of pixel embeddings

ences between the incidental pixels of the long range edges.
The algorithm mainly tested on this data operates in a continuous action space where the actor predicts mean and variance for a normal distribution sitting on every edge of the superpixel graph. The rewards are calculated in a supervised fashion based on the ground truth segmentation and in a unsupervised fashion based on the number of objects the knowledge that they are discs and their approximate radius.

4.0.1 Supervised training

For the supervised method a reward per subgraph is obtained by the dice score over the set of edges in the subgraph and the groundtruth edges obtained by evaluating if the larger parts of the incidental superpixels are of different classes in the grount truth segmentation. The larger parts are taken here because it is not guaranteed that the superpixels do not cross edges in the ground truth segmentation. The setup includes a replay buffer storing 100 experience tuples of $(s_t, a_t, r_{t+1}, s_{t+1})$. Initially experience is sampeld until the buffer is full. Then after each collected experience tuple there are 10 optimization steps performed on sampled experience tuples. The number of edges in each subgraph is set to $l = 10$. Optimization is done by stochastic gradient descent on mini batches of size 10 using the Adam optimizer, introduced in [32]. For the setting where the embedding network was ”warmed up” on the image - ground truth pairs prior to training the embedding networks parameters remain fixed after and no more optimization is performed on them. The optimization of the embedding network by the actors optimizer and loss function has been tested ended in messy embeddings. This can be related to the high variance in the actors loss function.
An example of the result using the per subgraph dice score as a reward is shown in figure 4.7.

The values in their first 3 principal components of the edge features, extracted from one of the critics networks are displayed in figure 4.4. After training they clearly carry the ground truth information in them.

The emperically predicted means of the actor network are visualized by histograms in figure 4.5. The prediction sepearates nicely and shows the class inbalance of 0 and 1 edges. The separation also shows, that the there is almost no uncertainty in the predictions anymore, leaving the multicut algorithm with a very simple task.

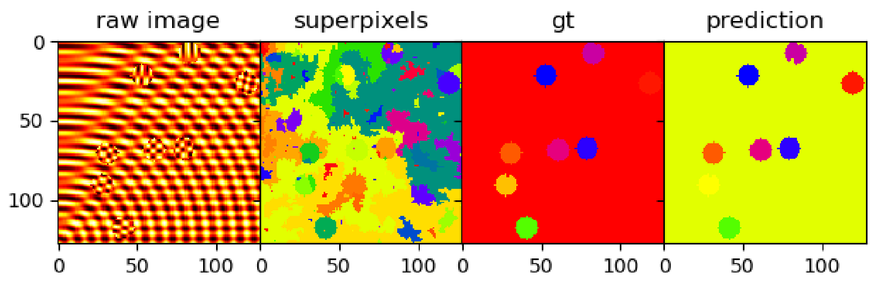


Figure 4.3: Result using the dice reward after 80000 optimization steps on mini batches of size 10. The gt image was obtained by computing the multicut based on the ground truth edges

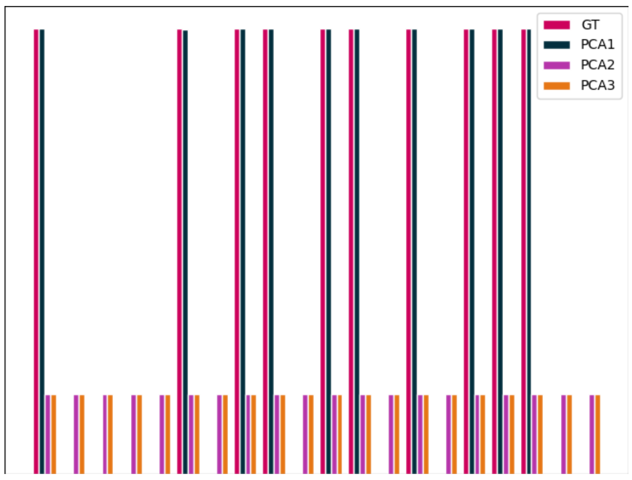


Figure 4.4: Visualization of the values of the edge features is their first 3 principal components (PCA1-3) and the ground truth edge values (GT) after training with the dice rewards for 80000 optimization steps on mini batches of size 10.

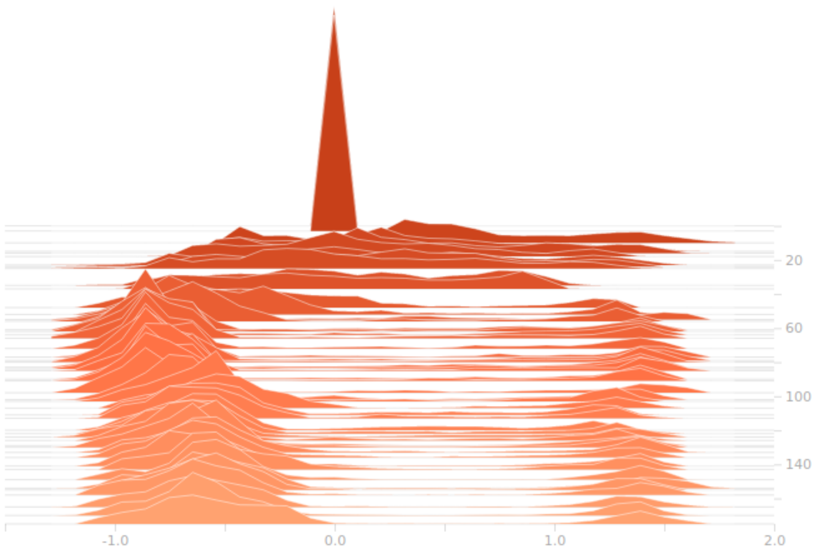


Figure 4.5: History of predicted means for the policy (supervised reward setting)

4.0.2 Unsupervised training

Since there is still some ground truth involved in the warmup training for the feature extractor, this setup cannot be considered fully unsupervised. But there is no more ground truth involved in obtaining the rewards which results in a large loss of supervision. The rewards are calculated locally on superpixel basis as well as globally in the following fashion.
From the segmentation obtained from the multicut based on the sampled action values objects larger than 32^2 pixels are considered background and all remaining objects are considered foreground. If there are too few foreground objects (number of objects is part of prior knowledge), all superpixels covered by background objects receive a negative reward. If there are too many objects, all the superpixel covered by foreground objects receive a negative reward.
A score for the shape of the foreground objects can be obtained using the Heugh transform for circles (see section 2.9). The Heugh transform of the corresponding radii (part of prior knowledge) is applied on the edge image of the resppective segmentation. All superpixels that are covered by discs that are drawn at the peaks of the Heugh transform above a threshold of 0.8 receive a positive reward dependend on the value in the Heugh transform at their respective peak.
In a final step the rewards are collected over each subgraph and averaged to a single value. This method indeed generates quite noisy rewards but the proposed pipeline was drafted to compensate for that.
Figure 4.7 shows an example of the results where the policy was trained under the unsupervised rewards.

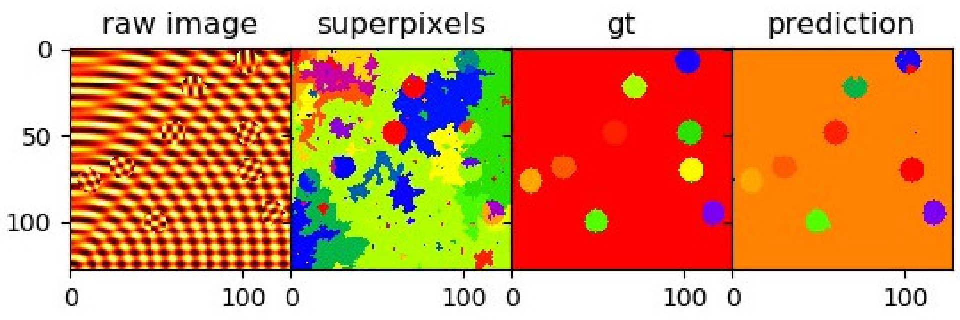


Figure 4.6: Result using the unsupervised reward after 80000 optimization steps on mini batches of size 10. The gt image was obtained by computing the multicut based on the ground truth edges

As the histogram of the predicted means for the policy in figure 4.7 shows, the predictions are not so clearly separated into means that are larger than 1 or smaller than 0 as is the case in the supervised setting. This can be attributed to the noisy rewards and leaves the multicut algorithm with a harder problem than in the supervised setting.

Data conformity

Supervised learning naturally provides a strict relation to the input data because the ground truth sample directly depends on the raw data.
Unsupervised learning typically needs a data term that encourages the solution to be close to the input data. The data term in this pipeline are the affinities and therefore the superpixel segmentation. The

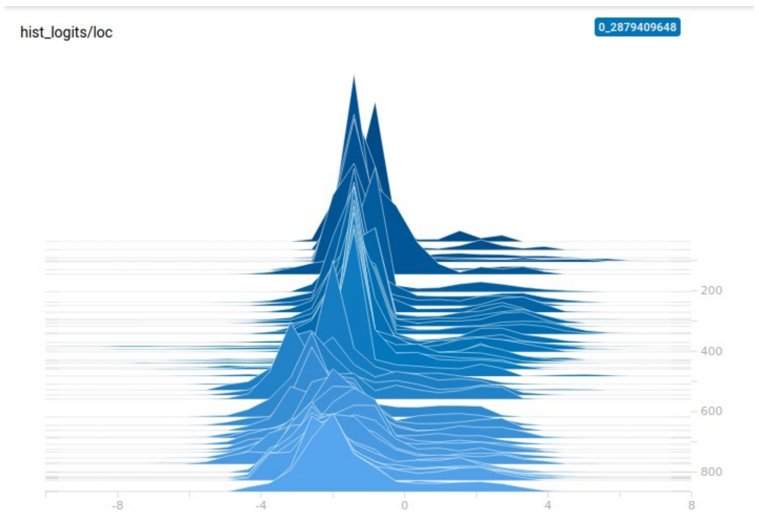


Figure 4.7: History of predicted means for the policy (unsupervised reward setting)

policy network learns ideally to distinguish between disc and non-disc superpixels. In an unfortunate case however it also could learn to build disks out of background superpixels. This would be the case if the shape of each superpixel is encoded in the pixel embeddings and the policy network learns to build discs based only on this shapes. Preventing the prediction being dependent on the superpixel shapes can be done by removing the superpixel segmentation from the state s_t and therefore from the input to the embedding network.

Affinities can also be obtained from pixel embeddings that where optimized by the contrastive loss in 2.7.2. Doing that produces less superpixels because the semantic affinities are based on object non-object rather than on differences in pixel intesities. This only makes sense if there is some ground truth which the emebdding network can be optimized with using contrastive loss. However after that the parameters of the embedding network should not be trained on unsupervised signals because the affinities are dependent on the pixel embeddings and the affinities form the data term. Changing the affinities based on an unsupervised signal might cause that the affinities do not represent the data anymore.

There have also be some thoughts of clustering pixels directly in the embedding space using the Mean Shift algorithm [33]. This adds some degrees of freedom the differentiability w.r.t the superpixel generation but also comes with the downside of loosing the data term.

Conclusion

The latest successes of reinforcement learning methods justifies to think about transferring the concept to the field of statistical learning. This has been done in this work for the image segmentation task. One drawback of reinforcement learning methods is the requirement of large amounts of experience data which makes it not suitable for supervised learning where a common challenge is to achieve satisfactory results with the possibly least amount of training data.
Therefore unsupervised and self supervised learning becomes more attractive in this setting. Luckily reinforcement learning adds a lot flexibility by not requirering that the generation of the supervision signal is differentiable. Given enough prior knowledge a supervision signal can be created for simple objects quite easily. This works given there is a good enough projection into the embedding space of the raw data. For this projection function there still needs to be some ground truth present. However for the training of the embeddings significantly less ground truth is necessary. The proposed methods for training the embedding network from scatch in an unsupervised fashion have been tested to some extend with so far no noteworthy results.
The proposed setting also gives rise to transform a segmentation problem into a classification problem with generating rewards on object level by having a pretrained predictor that classifies images of single objects. The classification scores can then be used to generate a reward.
The intention in the starting phase of this work was of course to have a method working on ”real” data for an existing problem and not only a toy problem. This intention turned out to go beyond the timeframe of this project. This among other things is reserved for future work on this topic.

Bibliography

[1] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, “Sample efficient actor-critic with experience replay,” 2016.

[2] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, “Soft actor-critic algorithms and applications,” 2018.

[3] R. S. Sutton and A. G. Barto, “Reinforcement learning:an introduction,” 2015.

[4] ——, “Reinforcement learning:an introduction,” 2015, pp. 74–121.

[5] ——, “Reinforcement learning:an introduction,” 2015, pp. 157–194.

[6] Q. Liu, L. Li, Z. Tang, and D. Zhou, “Breaking the curse of horizon: Infinite-horizon off-policy estimation,” 2018.

[7] R. Sutton, D. Mcallester, S. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” *Adv. Neural Inf. Process. Syst*, vol. 12, 02 2000.

[8] J. B. a. K. D. BrianD. Ziebart, Andrew Maas, “Maximum entropy inverse reinforcement learning,” 2008. [Online]. Available: https://www.aaai.org/Papers/AAAI/2008/AAAI08-227.pdf

[9] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, “Reinforcement learning with deep energy-based policies,” *CoRR*, vol. abs/1702.08165, 2017. [Online]. Available: http://arxiv.org/abs/1702.08165

[10] P. N. Ward, A. Smofsky, and A. J. Bose, “Improving exploration in soft-actor-critic with normalizing flows policies,” *CoRR*, vol. abs/1906.02771, 2019. [Online]. Available: http://arxiv.org/abs/1906.02771

[11] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, “Normalizing flows for probabilistic modeling and inference,” 2019.

[12] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, “Rainbow: Combining improvements in deep reinforcement learning,” 2017.

[13] H. van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” *CoRR*, vol. abs/1509.06461, 2015. [Online]. Available: http://arxiv.org/abs/1509.06461

[14] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” 2015.

[15] Z. Wang, N. de Freitas, and M. Lanctot, “Dueling network architectures for deep reinforcement learning,” *CoRR*, vol. abs/1511.06581, 2015. [Online]. Available: http://arxiv.org/abs/1511.06581

[16] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” 2016.

[17] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, “Spectral networks and locally connected networks on graphs,” 2013.

[18] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, “Geometric deep learning: Going beyond euclidean data,” *IEEE Signal Processing Magazine*, vol. 34, no. 4, p. 18–42, Jul 2017. [Online]. Available: http://dx.doi.org/10.1109/MSP.2017.2693418

[19] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” 2017.

[20] S. Wolf, A. Bailoni, C. Pape, N. Rahaman, A. Kreshuk, U. Köthe, and F. A. Hamprecht, “The mutex watershed and its objective: Efficient, parameter-free image partitioning,” 2019.

[21] J. H. Kappes, M. Speth, B. Andres, G. Reinelt, and C. Schn, “Globally optimal image partitioning by multicuts,” in *Energy Minimization Methods in Computer Vision and Pattern Recognition*, Y. Boykov, F. Kahl, V. Lempitsky, and F. R. Schmidt, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 31–44.

[22] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin, and M. Jorge Cardoso, “Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations,” *Lecture Notes in Computer Science*, p. 240–248, 2017. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-67558-9_28

[23] B. D. Brabandere, D. Neven, and L. V. Gool, “Semantic instance segmentation with a discriminative loss function,” 2017.

[24] F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2015.7298682

[25] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” 2013.

[26] R. O. Duda and P. E. Hart, “Use of the hough transformation to detect lines and curves in pictures,” *Commun. ACM*, vol. 15, no. 1, p. 11–15, Jan. 1972. [Online]. Available: https://doi.org/10.1145/361237.361242

[27] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein, “Geometric deep learning on graphs and manifolds using mixture model cnns,” 2016.

[28] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” 2015.

[29] K. Samir and S. Barna, “On finding dense subgraphs,” 2009.

[30] M. Fey and J. E. Lenssen, “Fast graph representation learning with PyTorch Geometric,” in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[31] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” 2015.

[32] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014.

[33] Yizong Cheng, “Mean shift, mode seeking, and clustering,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 8, pp. 790–799, 1995.