1 Neuroscience

1.1 Visual Cortex

1.1.1 Primary Visual Cortex

The primary visual cortex is the simplest, earliest cortical visual area. It is highly specialized for processing information about static and moving objects and is excellent in pattern recognition.

1.2 Receptive Fields

The term receptive field originally described an area of the body service where a stimulus could elicit a reflex (Sherrington, 1906). The definition was later extended to sensory neurons defining the receptive field as a restricted region of visual space where a luminous stimulus could drive electrical responses in a retinal ganglion cell: *"Responses can be obtained in a given optic nerve fiber only upon illumination of a certain restricted region of the retina, termed the receptive field of the fiber"* (Hartline, 1938). $http://www.scholarpedia.org/article/Receptive_field$

General definition spanning different types of neurons across sensory modalities: the receptive field is a portion of sensory space that can elicit neuronal responses when stimulated. The sensory space can be defined in a single dimension (e.g. carbon chain length of an odorant), two dimensions (e.g. skin surface) or multiple dimensions (e.g. space, time and tuning properties of a visual receptive field). The neuronal response can be defined as FIRING RATE (i.e. **number of action potentials generated by a neuron**) or include also subthreshold activity (i.e. depolarizations and hyperpolarizations in membrane potential that do not generate action potentials).

1.2.1 Visual Receptive Fields

2 Supervised vs. Unsupervised learning

3 Hebbian Learning Rule

3.1 PCA

3.2 ICA

Gaussianity vs. non-gaussianity

4 Competitive Learning

4.1 Clustering

4.2 Neuronal Version of Clustering

Kohonen learning rule

5 Reinforcement learning

RL vs. supervised learning Supervised learning, used in statistical pattern recognition and artifical neural networks is learning from examples provided by a knowledgable external supervisor whereas in RL an agent is instructed to learn from its own experience. It is a "behavioral" learning problem: Through interaction between the learning system and its environment, whereas the system seeks to achieve a specific goal.

trade-off exploration vs. exploitation An agent has to exploit what it ALREADY KNOWS in order to obtain a reward, but it also has to EXPLORE in order to make BETTER ACTION SELECTIONS in the future. The agent must try a variety of actions and progressively favour those that appear to be best, because neither schemes can be pursued exclusively without failing a the task. Estimate reward probabilities vs. take action to maximize reward.

RLs main characteristic RL explicitly considers the WHOLE problem of a goal-directed agent INTERACTING with an uncertain environment. All reinforcement learning agents have explicit goals, can sense aspects of their environments, and can choose actions to influence their environments. Moreover, it is usually assumed from the beginning that the agent has to operate

despite significant UNCERTAINTY ABOUT THE ENVIRONMENT it faces. When reinforcement learning involves PLANNING, it has to address the *interplay between planning and real-time action selection*, as well as the question of how environmental models are acquired and improved.
An agent's actions are permitted to affect the future state of the environment (e.g., the next chess position, the level of reservoirs of the refinery, the next location of the robot), thereby affecting the options and opportunities available to the agent at later times. Correct choice requires taking into account indirect, delayed consequences of actions, and thus may require foresight or planning. The knowledge the agent brings to the task at the start–either from previous experience with related tasks or built into it by design or evolution–influences what is useful or easy to learn, but interaction with the environment is essential for adjusting behavior to exploit specific features of the task.

5.1 Elements of reinforcement learning

there are four main subelements of a reinforcement learning system.

Policy

commonly denoted by π, it defines the learning agents way of behaving at a given time. Mapping from the perceived states of the environment to the actions that present themselves to the agent in these states. The policy alone is sufficient to determine the behaviour of an agent. In oder words, a policy is a rule used by the learning system to decide what to do, given the current state of the environment.

Reward function

defines the goal in a reinforcement learning problem. Maps each perceived state of the environment to a single number, a reward, which describes the intrinsic desirability of the state. The agent's sole objective is to maximize its total gain from the rewards it receives.
The reward function may serve as a basis for altering the agent's policy.

Value function

While the reward function indicates which actions are immediately beneficial to the agent, a **value function** specifies what is good for the agent in the long run. The value of a state is the EXPECTED total amount of rewards the agent will accumulate from this state onwards. For example, a state might always yield a low immediate reward but still have a high value because it is regularly followed by other states that yield high rewards.

Model of the environment

A model mimics the behaviour of the environment. *For example, given a state and action, the model might predict the resultant next state and next reward.* Models are used for PLANNING, by which we mean any way of deciding on a course of action by considering possible future situations before they are actually experienced. The incorporation of models and planning into reinforcement learning systems is a relatively NEW DEVELOPMENT. Early reinforcement learning systems were explicitly trial-and-error learners; what they did was viewed as almost the opposite of planning. Nevertheless, it gradually became clear that reinforcement learning methods are closely related to dynamic programming methods, which do use models, and that they in turn are closely related to state-space planning methods.

5.2 Evolutionary methods

Genetic algorithms, genetic programming, simulated annealing, and other function optimization methods have also been used to solve reinforcement learning problems. These methods search directly in the space of policies without ever appealing to value functions. We call these **evolutionary methods** because their operation is analogous to the way biological evolution produces organisms with skilled behaviour even when they do not learn during their individual lifetimes (organisms that have chosen adaptive policies survive). Effective when space of policies is sufficiently small or accurately sensing the state of the environment is impossible.

5.3 Reward-based action learning: Q-values

$Q(S,a)$ is the expected reward and can be written as $\sum_{s'} P^{a}_{\underrightarrow{s\,s'}} R^{a}_{\underrightarrow{s\,s'}}$. The optimal strategy is to take the action such that $Q(s,a^*) > Q(s,a_j)$. Q values are not known at the beginning, therefore, we need to learn them -¿ iteratively update. $\Delta Q(s,a) = \theta[r - Q(s,a)]$. If the mean update has converged, then $< \Delta Q(s,a) >= 0$

There are different strategies:

- Greedy strategy

- η - greedy strategy $P = 1.\eta$
- Softmax strategy $P(a') = \frac{exp[\beta Q(a')]}{\sum_a exp[\beta Q(a)]}$
- Optimistic greedy (Q values too big at start)

5.4 Bellmann Equation and Sarsa

Multi step horizon reward-based action learning!

$$\Delta Q(s,a) = \eta[r - (Q(s,a) - \gamma Q(s',a'))]$$

We can use the Bellman equation to write $Q(s,a)$

$$Q(s,a) = \sum_{s'} P^a_{s \to s'} * [R_{s \to s'} + \gamma \sum_{a'} \pi(s',a') Q(s',a')]$$

The Bellman equation can be used for the SARSA algorithm:

1. Being in state s, chose action a according to policy
2. Observe reward r, next state s'
3. Choose action a' in state s' according to policy
4. Update $\Delta Q(s,a)$
5. s' -¿ s; a' -¿ a
6. Goto 1)

We can rewrite the Bellman equation and express it as State values V.

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{\underset{s}{} \overrightarrow{s'}}[R^a_{\underset{s}{} \overrightarrow{s'}} + \gamma * V^\pi(s')]$$

which we can write in form of the value function as:

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{s \to s'}[R^a_{s \to s'} + \gamma V^\pi(s')]$$

The difference between Q and V is the following: The value function is the expected return starting from state s_t and following policy π. The state-action value function $Q(s,a)$ is the expected return starting from state s, taking action a and thereafter following policy π.

5.5 On-policy vs off-policy learning

On-policy means SARSA; off-policy means Q-learning. The difference lies in the method how to compute $Q(s,a)$. While for SARSA, we use the policy for the discounted term $\gamma \sum_{a'} \pi(s',a')Q(s',a')$, the Q-learning algo simply does $\gamma * max_{a'}Q(s',a')$. Therefore, Q-learning uses greedy for update and the policy for action selection (η – greedy)

5.6 Eligibility traces

At the moment of the reward update, update also previous action values along trajectories. The problem is that learning is very slow and the information diffusion across several states takes time.

γ is the memory reduction parameter. Every time an action a is taken, one should do the following:

- If a = action taken in state j: $e_{aj}(t) = \gamma * \lambda * e(s,a) + 1$
- else: $e(s,a)(t) = \gamma\lambda e(s,a)$

Then, for all states s and actions a:

$$\Delta Q(s,a) = \eta\delta_t e(s,a)$$

An eligibility trace is a temporary record of the occurrence of an event, such as the visiting of a state or the taking of an action (backward view). The trace marks the memory parameters associated with the event as eligible for undergoing learning changes. The more theoretical view is that they are a bridge from TD to Monte Carlo methods.

5.7 Continuous states

$$Q(s,a) = \sum_j w_{aj} * r_j(s) = \sum_j w_{aj} * \phi(s - s_j)$$

If the time interval between subsequent weight updates approaches zero, the difference between two consecutive states $s' - s = v(a) * \Delta t$, where $v(a)$ is the velocity resulting from the choice of the action a. $s' - s$ approximates zero, therefore $s' \approx s$. Similarly, the finer the time resolution becomes, the likelier is that $a \approx a'$. Thus, for small time steps, an online gradient descent on the erro term E_t becomes closer to a δ_t modulated Hebbian rule.

5.8 Temporal Difference TD(λ)

TD incorporates eligibility and continuous state space. We do a function approximation using weights and instead of updating $Q(s,a)$, we do a weight update.

$$Q_w(s,a) = \sum_{i=1}^{n} w_i^a * r_i(s)$$

$$\Delta w_{aj} = \eta \delta_t e_{aj}$$

We can write the eligibility trace as $e_{aj}(t) = \gamma \lambda e_{aj}(t - \Delta t) + r_j \delta_{a,a'}$

6 Energy-Based Models (EBM)

A scalar energy is associated to each configuration of the variables of interest in the model (cost function). Learning then corresponds to modifying the energy function so that its shape has desirable properties, for example we would like desirable configurations to have low energy. The probabilistic distribution of for the model is define by an energy function:

$$p(x) = \frac{\exp(-E(x))}{Z}$$

This distribution is the Boltzmann or Gibbs distribution which originates from classical statistical mechanics.
The normalization factor Z is called the partition function $Z = \sum_x \exp(-E(x))$.
An energy-based model can be learnt by performing (stochastic) gradient descent on the empirical negative log-likelihood of the training data.

6.1 EBMs with hidden units

Often one does not observe the examples x fully or one wants to introduce some non-observed variables to increase the expressive power of the model. So we consider a hidden part y additionally to the observed part x, giving:

$$P(x) = \sum_y P(x,y) = \sum_y \frac{\exp(-E(x,y))}{Z}$$

By introducing the notation of FREE ENERGY

$$\mathcal{F} = -\log \sum_y \exp(-\mathcal{F})$$

we can the rewrite the probability distribution in the condensed of the gibbs distribution:

$$P(x) = \frac{exp(-\mathcal{F}(x))}{Z}, with Z = \sum_x \exp(-\mathcal{F}(x))$$

6.2 Restricted Boltzmann Machines (RBM)

Boltzmann Machines (BMs) are a particular form of log-linear (Gibbs distribution) Markov Random Fields (MRF), where the energy function is linear in its free parameters. In order to represent complicated distributions (limited parametric setting $\rightarrow$ non-parametric setting) we consider some of the variables to be hidden (not observed). One can increase the modeling capacity of the Boltzmann Machine (BM) by adding for hidden variables/units. Furthermore a RBM is restricted to a bipartite Graph, because there no lateral connections visible-visible and hidden-hidden are allowed. We introduce the notation of $v = x$ for visible units and $h = y$ for hidden units.
The energy function therefore is defined as:

$$E(v, h) = -b^t v - c^t h - h^t W_{u,v} u$$

where $W_{u,v}$ represents a connection matrix of weights for hidden and visible units and b, c are the offsets of the visible and hidden layers respectively.
By plugging in the energy function we get the following representation of the free energy formula:

$$F(v) = -b^t v - \sum_i \log \sum_{h_i} \exp(h_i(c_i + W_i v))$$

Because of the specific structure of RBMs, visible and hidden units are conditionally independent of one-another

A particular set of "synaptic" weights is said to constitute a perfect model of the environmental structure if it leads to exactly the same probability distribution of the states of the visible units as when these units are clamped by the environmental input vectors.

RBMs with binary units

Commonly RBMs are studied with binary units $v_i, h_i \in \{0, 1\}$ ... The free energy simplifies to:

$$\mathcal{F}(v) = -b^t v - \sum_i \log(1 + \exp(c_i + W_i v))$$

deeplearning.net/tutorial/rbm.html
sigmoid function (sigm):

$$\sigma(t) = sig(t) = \frac{1}{1+\exp(-t)} = \frac{1}{2} \cdot (1 + \tanh(\frac{t}{2}))$$