EXAM 07.06.19

Question 1

1). When we have to use this method we split our dataset D into K subsets $S_1 \dots S_K$ with $|S_i| > 30$. Each subset must be disjointed from another, $S_i \cap S_j = \emptyset \ \forall i,j$

for $i = 1 \dots K$ do:

$T_i = D - S_i$

$h_i = L(T_i)$

$\delta_i = \text{error}_{S_i}(h_i)$

$\text{error}_{L,S} = \text{avg}_i \{\delta_i\} = \frac{1}{K} \sum_{i=1}^{K} \delta_i$

2). for $i = 1 \dots K$ do:

$T_i = D - S_i$

$h_A = L(T_A)$

$h_B = L(T_B)$

$\delta_i = \text{error}_{S_i}(h_A) - \text{error}_{S_i}(h_B)$

$\bar{\delta} = \text{avg}_i \{\delta_i\} = \frac{1}{K} \sum_{i=1}^{K} \delta_i$

If $\bar{\delta} < 0$ L_A is better than L_B.

Question 2

1).

2). The difference is that SVM aims at maximum margin with the better accuracy, in this the probability that a sample will be misclassified is very low, because we have the maximum margin. Perceptroin is based on a sequential learning algorithm (SGD) so we update w to find the best w but is possible that we don't find a separation surface with maximum margin and is also possible that we don't find a solution if we do a wrong choice for η.

3). I prefer SVM because we have the maximum margin and the uncertainty of our prediction is minimum and because we can use SVM also if the dataset is not perfectly linearly seperable introducing slack variables.

Question 3

1). In the K-armed bandit problem we have a single-state MDP: $MDP = \langle \{x_0\}, A, \delta, r \rangle$ with $\{x_0\}$ single state, A set of actions, $\delta: X \times A \to X$ transition function and $r: X \times A \to \mathbb{R}$ reward function. We know that $\delta(x_0, a_i) = x_0 \ \forall a_i \in A$ and $r(a_i) = r_{(i)}$. We want to find an optimal policy function $\pi^*(x_0)$.

- If $r(a_i)$ is deterministic and known: $\pi^*(x_0) = \underset{a_i \in A}{\operatorname{argmax}}\, r(a_i)$

- If $r(a_i)$ is deterministic and unknown:

- We execute each action $a_i \in A$ and we collect $r(i)$
- $\pi^*(x_0) = a_i$ with $i = \underset{i=1\dots|A|}{\operatorname{argmax}}\, r(i)$

- If $r(a_i)$ is non-deterministic and known: $\pi^*(x_0) = \underset{a_i \in A}{\operatorname{argmax}}\, E[r(a_i)]$

- If $r(a_i)$ is non-deterministic and unknown:

For each entry, we initialize $\Theta(0)[i] \leftarrow 0$ and $C[i] \leftarrow 0$

for each time $t = 1 \dots T$:

We choose an index $\hat{i}$ such that $a(t) = a(\hat{i})$.
We execute the action
We collect the reward
Increment $C[\hat{i}]$
$\Theta(t)[\hat{i}] \leftarrow \frac{1}{C[\hat{i}]}\left(r + (C[\hat{i}] - 1)\,\Theta(t-1)[\hat{i}]\right)$

$\pi^*(x_0) = a_i$ with $i = \underset{i=1\dots|A|}{\operatorname{argmax}}\, \Theta(T)[i]$

2). $Q_n(a_i) \leftarrow Q_{n-1}(a_i) + \alpha[r - Q_{n-1}(a_i)]$

with $\alpha = \frac{1}{1 + V_{n-1}(a_i)}$

with $V_{n-1}(a_i)$ = #executions of the action a_i up to time $n-1$.

Question 4

1). In probabilistic classification models, given a target function $f: X \to C$ with a dataset $D = \{(x_i, c_i)_{i=1}^N\}$, we are interested in $p(c_i|x)$. In generative models we compute $p(x|c_i)$ and then we move to $p(c_i|x)$ using the Bayes theorem. In discriminative models we compute directly $p(c_i|x)$.

2).

$$p(c_1|x) = \frac{p(x|c_1)p(c_1)}{p(x|c_1)p(c_1) + p(x|c_2)p(c_2)} = \frac{1}{1+e^{-a}} = \sigma(a)$$

with $a = \ln \frac{p(x|c_1)p(c_1)}{p(x|c_2)p(c_2)} = \frac{N(x, \mu_1, \Sigma_1)p(c_1)}{N(x, \mu_2, \Sigma_2)p(c_2)}$

Question 5

1). In the convolutional stage is performed the convolution operation:

$$I * K(i,j) = \sum_{m \in S_1} \sum_{n \in S_2} I(m,n)\, K(i-m, j-n)$$

This is the most important operation in a CNN. After the convolution operation we can also find padding, in this mode we can convolve the output without skipping any pixel

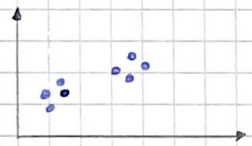
2). In parameter sharing we are imposing that some values must be the same. In this mode we reduce the number of parameters and we improve invariance to traslation.

In sparse connettivity output depends only on a few inputs, we don't connect each layer to the following one.

Question 6

1). In supervised learning we have a target function $f: X \to Y$ and the dataset is $D = \{(x_i, y_i)_{i=1}^N\}$, to each input value is associated a label. In unsupervised learning we have a target function $f: X \to Y$ and the dataset is $D = \{x_n\}$, so we have input values but in this case we don't have labels.

2).



We have as input the dataset $D = \{x_n\}$ and the value of K = number of clusters and as ouput $\mu_1, \mu_2 \dots \mu_K$. We can use K-means to solve this problem.

1). We select the value of K = #clusters

2). We have to split our instances: we can do this randomly or systematically as follows

- We take the first K samples and we assign each sample to a single element cluster.
- For the remaining $N-K$ samples we assign each sample to the cluster with the nearest centroid and, after each assignment, we recompute the centroid.

3). We analyze all the samples in sequence and if a sample is not in the correct cluster we move the sample to the correct cluster and we recompute both the centroid.

4). We repeat step 3 until convergence is achieved (where the aren't switches). The algorithm converges if the sum of the distances decreases.