Scalable Machine Learning and Deep Learning - Review Questions 5

Deadline: December 8, 2019

Anna Martignano, Daniele Montesi

May 4, 2020

1. What is a generative model? Can you name a type of generative autoencoder?

   A generative model has the purpose of modeling the joint probability density function of the couple input x output t $p(t, x)$, which allows to generate also new data from what has been learned. These models are used for data augmentation.
   Variational autoencoders are an example of generative autoencoders, they differ from other autoencoders since they produce a mean coding μ and a standard deviation σ instead of directly producing a coding for a given input.
   The actual coding is then sampled from the Gaussian distribution with mean μ and standard deviation σ learnt by the recognition network. While the generative network works exactly as in others autoencoders by decoding the sampled coding normally.
   The only modification which needs to be performed to train Variational Autoencoders is the cost function: a part of the utilization of the reconstruction loss it is important to add as well the latent loss which forces variational autoencoders to learn a coding that as much close to the actual distribution of the codings with the Kullaback-Leibler divergence.

---

2. What are the main difficulties when training GANs?

   Training GANs has carried out many difficulties in the last decade. First problems were about the reaching of the Nash Equilibrium. However, nothing guarantees to reach this equilibrium. Very often **Mode Collapse** happens: the generator starts to output similar things without being able to offer diversity. An intuitive example of Mode Collapse is when the generator becomes convincing in generating pictures of shoes and forgets how to produce others, viceversa, the discriminator becomes good in discriminating shoes but forgets about all the other classes.
   It is common as well to end up with unstable parameters during the training since generator and discriminator are constantly pushing against each other making the parameters oscillate. Due to this complex "competing" dynamics, GANs are very sensitive to the hyperparameters.
   There exists solution to tackle the GANs training difficulties, one of them is Experiece Replay Buffer (the same used in Deep-Q-Networks), another one is using mini-batch.

---

3. What are minibatch standard deviation layers? Why will they help training GANs?

   GAN models are powerful tools to generate new content, however, problems can arises about the variety of classes that they are able to represent. This problem stems from the same root as "mode collapse". The Layer is present on the discriminator, and it inputs feature statistics (std_dev) across images in a batch. The statistic is computer for every channel, and then is averaged over all the inputs to get a single value, which is used as a mapped feature. This helps the discriminator to understand whether the "variety" of the generator is high or low. In the latter case, it is going to vote for "fake". In turns, the generator will be pushed to produce more variety. This helps to solve the Mode Collapse phenomenon.

---

4. What is Nash Equilibrium? How does it relate to GANs?

   Nash Equilibrium is a proposed solution in game theory in which there are two or more players that are competing against each other and they are supposed to know the equilibrium strategies of the other players.
   GANs are composed of two neural networks that are competing against each other during the adversarial training:

   - A **generator** network which tries to create new data to similar training data.
   - A **discriminator** network which tries to distinguish among real data and generated one.

   As training goes on, the networks may end up with no guarantees in the Nash Equilibrium state in which the generator produces perfectly realistic images, and the discriminator is forced to guess (50% real, 50% synthetic).

---

5. How does Reinforcement Learning differ from Supervised learning and Unsupervised Learning?

   Reinforcement learning is defined as the search for the optimal policy that guarantees the maximization of the discounted cumulative reward on a certain problem. Reinforcement learning is different from Supervised/Unsupervised learning because the experience is an episode, i.e. a sequence of actions from states. An agent performs an action A starting from a state S, received by an entity called Environment, that in turns emits another state S', a reward R, and defines if the experience has finished or not (terminal state). The above description is intended as a sequential Markov Decision Process, where a State represent all what is needed to predict the future (i.e. the next state/return received by the Environment).

---

6. What is the credit assignment problem?

   The concept of *credit assignment problem* was introduced by Minsky in 1961 in his paper "Steps Toward Artificial Intelligence".
   This problem emphasizes the following question: "How do you distribute credit for success among the many decisions that may have been involved in producing it?".
   In Reinforcement learning the credits (i.e. rewards) are defined by a Function that must be designed according to the specific problem. It is however very crucial for the specific task, because if may guide away the agent from the specific goal of the problem. For instance, let's assume that we want to model Candy Crush: if we are assigning the credits as Sutton and Barto suggests (-1 for lose, +1 for win) the process will be very slow at learning. We can introduce **denser** reward functions i.e. when a combo is performed, however, then the goal of the agent will become to perform combos instead of winning the game. That is why credit assignment problem is very important in RL.

---

7. Considering the following slot machines that show the number of times they have been visited and the total reward each individual machine has dealt so far. Using the following exploration-exploitation strategies, which slot machine should you play on the next turn?

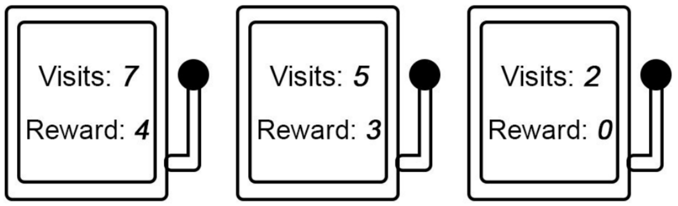
   a. UCB1 ($Q(i) + c\sqrt{\frac{\ln n}{n_i}}$), where $c = 1$

      - UCB1(machine a) $= \frac{4}{7} + \sqrt{\frac{log(14)}{7}} = 1.06281089$
      - UCB1(machine b) $= \frac{3}{5} + \sqrt{\frac{log(14)}{5}} = 1.128760817$ ✓
      - UCB1(machine c) $= \frac{0}{2} + \sqrt{\frac{log(14)}{2}} = 0.548662005$

   b. ϵ-greedy, where $\epsilon = 0.1$

- $\epsilon - greedy(\text{machine a}) = \frac{4}{7}$
- $\epsilon - greedy(\text{machine b}) = \frac{3}{5}$ ✓
- $\epsilon - greedy(\text{machine c}) = \frac{0}{2}$

There is a 90% probability that the machine b will be selected, and a 10% to select either a or c.



---

8. In the implementing the Q-learning algorithm, when should one use the tabular version and when should one use a function approximator, e.g., Neural Network?

   Q-learning is a method for prediction tasks in Model-free environments (i.e. environments where all the states are not known/visible by the agent), and hence it must be sampled and approximated according to some formulas. In particular, Q-learning is an off-policy algorithm that helps to compute the optimal policy in such model-free scenarios. However, it is possible to use it also when the environment is well known, but the states are too many (i.e. think about "Go" states are $\sim exp(150)$). The problem with Q-learning is that we must keep in memory a table (Q-table) where all the possible values of Q(s,a) are stored. For every possible state in the set S of the states we have found, we have to keep a matrix having shape $|S| \times |A|$, where A is the set of actions possible. In that case, Function approximators are the best choice because does not require to keep such a table. A typical example of those techniques is Deep-Q-Networks, where a Neural Network is used to compute all the Q-value function of a State for all the possible actions. We can use the output of the network to get the maximum Q-value predicted and use it to compute the Greedy-action over the Q-learning algorithm and update the policy.