# Scalable Machine Learning and Deep Learning - Review Questions 5

By

Akhil Yerrapragada (akhily@kth.se)

Gibson Chikafa (chikafa@kth.se)

Group Name – gibson_akhil

1. **1 point.** Assume we have a stacked autoencoder with three hidden layers $h_1$, $h_2$, and $h_3$, in which each layer applies the following functions respectively, $\mathbf{h_1} = f_1(\mathbf{x})$, $\mathbf{h_2} = f_2(\mathbf{h}_1)$, and $\mathbf{h}_3 = f_3(\mathbf{h}_2)$, and the output of the network will be $\mathbf{y} = f_4(\mathbf{h}_3)$. Do you think if it is a good autoencoder if it generates $f_4(f_3(f_2(f_1(\mathbf{x})))) = \mathbf{x}$ for all input instances $\mathbf{x}$. How can we improve it?

**Ans: If the output is similar to the input then we can say that the model is overfitted, therefore, it is not a good autoencoder.**

**Using stacked auto encoders can help reduce the risk of overfitting the model. It achieves this by halving the number of weights.**

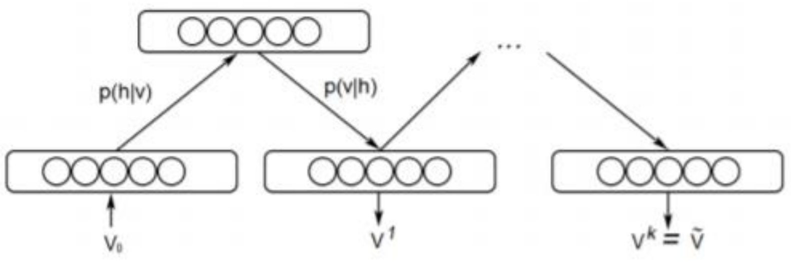
---

2. **1 point.** How does Gibbs sampling work? When do we need to use Gibbs sampling?

**Ans:**

**Gibbs sampling works as following:**

**Step1: Assuming 2 probability distributions h (for hidden layer) and v (for visible layer), we consider the values of one of the probability distribution to be fixed and given the fixed distribution, we measure the probability of the other. P(h|v) determines the probability of h w.r.to fixed v. P(v|h) determines the probability of v w.r.to fixed h. This process is repeated k times.**



**Step2: Here we compute the gradient descent. We call it Contrastive divergence.**

$$\mathbf{w} = \mathbf{w} + \eta(\texttt{positive}(\mathbf{e}) - \texttt{negative}(\mathbf{e}))$$

**If the possible combinations between v and h are higher, it will be difficult to calculate the joint probability. In such a situation, we use Gibbs sampling. Here, we calculate the conditional probability of h given v.**

---

3. **1 point.** How do you tie weights in a stacked autoencoder? What is the point of doing so?

**Ans: The weights can be tied as following:**

1) **Create a custom dense layer that takes another dense layer as an input.**
2) **Obtain the weights from input dense layer and transpose it.**
3) **Set the current weight of custom dense layer with obtained transposed weight of the dense layer taken as input.**

   **Doing so speeds up training process and limits the risk of overfitting since tying weights halves the number of weights in the model.**

---

4. **1 point.** What are minibatch standard deviation layers? Why will they help training GANs?

**Ans: It is one of three layers required when implementing progressive growing GANs. It provides statistics of a batch and is added at the end of discriminator. Here, statistics mean the computed standard deviation and less standard deviation means the batch contains images with less variety.**

**When training, it helps discriminator to measure the similarity of images in a batch so that it can reject the batch that lacks diversity. This motivates generator to produce images of greater variety thereby reducing the chance of model collapse.**

5. **1 point.** What is Nash Equilibrium? How does it relate to GANs?

**Ans:**

**Nash Equilibrium is a concept in game theory. A game may contain multiple Nash equilibria or none. In Nash Equilibrium, an optimal outcome is where no player can deviate from his strategy after considering an opponent's choice. Here an individual cannot obtain any benefit from changing their action considering other players actions. Here we assume that other players remain constant with their strategies.**

**Let us consider two players as functions – generator (G) and discriminator (D) each of which is differentiable w.r.to its inputs and parameters. The discriminator takes x an input and uses $\theta^{(D)}$ as parameters and the generator takes z as an input and uses $\theta^{(G)}$ as parameters. Both players have cost functions (J) defined. The discriminator minimizes the cost function $J^{(D)}$ ($\theta^{(D)}$, $\theta^{(G)}$) by controlling only $\theta^{(D)}$ and similarly generator minimizes the cost function $J^{(G)}$ ($\theta^{(G)}$, $\theta^{(D)}$) by controlling only $\theta^{(G)}$. Here, we can observe that a player's cost depends on other players parameters, but a player cannot change other players parameters. Now to obtain a minimum i.e. a point in parameter space that best represents all the neighboring points is Nash equilibrium in this context. Here Nash equilibrium is a local minimum of $J^{(D)}$ w.r.to $\theta^{(D)}$ and $J^{(G)}$ w.r.to $\theta^{(G)}$ i.e. a tuple ($\theta^{(D),}$ $\theta^{(G)}$)**

**GANs are designed to reach Nash equilibrium but nothing guarantees that it will be reached and is considered more difficult problem.**