**First the Coding Problem:**

This is the GitHub repo for it: [GitHub Repo](https://github.com/MohamedElashry1196/Questions-Answers-Bert-Model)

This is deployed version URL on Vercel: [Exposed Web App](https://questions-answers-bert-model.vercel.app/)

I tried both approaches with Heroku and SageMaker, note that there is a Q&A.ipynb notebook in repo for SageMaker deployment.

Feel free to test the questions answered with any sample and get scoring on any sample of the dataset with the format and add-in repo.

(you can get more samples from this link: [Data Samples](http://downloads.cs.stanford.edu/nlp/data/coqa/coqa-train-v1.0.json))

**Second Technical Questions:**

1 -

1. the answer will be : **32 \* 8**
2. I prefer using **PyTorch** because the total number of models available on HuggingFace are either PyTorch or TensorFlow exclusive or available for both frameworks. As we can see, the number of models available for use exclusively in PyTorch absolutely blows the competition out of the water. Almost 85% of models are PyTorch exclusive, and even those that are not exclusive have about a 50% chance of being available in PyTorch as well. In contrast, only about 16% of all models are available for TensorFlow, with only about 8% being TensorFlow-exclusive. also, PyTorch has invested in making deployment easier, previously being notoriously lackluster in this arena. Previously, PyTorch users would need to use Flask or Django to build a REST API on top of the model, but now they have native deployment options in the form of TorchServe and PyTorch Live.
3. If I think of this problem with a large neural network, there is a technique besides model parallelism that can be used to improve deep learning performance. It is called multi-GPU processing and can be used only if you have several GPUs aboard.If it is true to my system, I must definitely try multi-GPU processing as it will be both fast and effective. There are two ways of working with multiple GPUs. The first approach suggests using each GPU for a separate task. Thus, you will be able to experiment and run multiple algorithms at once. However, you will not achieve better speed.The second type requires combining several GPUs in one computer (making a GPU cluster) to achieve better performance. It is called GPU parallelism and can be used to work with the data parallelism concept with no fear of the large neural network obstacle. Let’s make this clear. You can simply use each separate GPU as a worker node. Thus, your model’s performance will improve drastically**. So, I would check that the new GPUs are accessible from CUDA.**
4. The solution is that call the **SyncBatchNorm** instead of the BatchNorm in multi-GPU training. More precisely, we use the convert\_sync\_batchnorm() method to convert. [SyncBatchNorm](https://pytorch.org/docs/stable/generated/torch.nn.SyncBatchNorm.html#torch.nn.SyncBatchNorm.convert_sync_batchnorm 27) The phenomenon may be caused by the BatchNorm statistics being computed within each GPU, whereas the statistics largely differ from other GPUs in the Non-IID context.

2-

1. **Possible solutions**

**Money-costing solution**: One possible solution is to provide our instance with a larger RAM that is capable of handling the entire dataset. then create some clustering arrangement to handle the workload.

**Time-costing solution:** our RAM might be too small to handle our data, but often, that hard drive is much larger than our RAM. So, why not Using the hard drive to deal with our date will make the processing of it much slower because even an SSD hard drive is slower than a RAM.

**method 1:** Enable Dynamic Memory Allocation

**method 2:** Incrementally Reduce the Nodes in Each Layer

makes sense to reduce the nodes on the less important layers before reducing the nodes on the more important layers.

**method 3:** Explore a Deeper Network with fewer Nodes

In some cases, having a deeper network with fewer nodes may yield the same results as a shallow network with more nodes. However, a deeper network can have more stability issues during training.

**method 4:** Check the GPU Memory Usage

Once the error messages have been resolved, check the GPU memory usage. This confirms that the model is using most of the GPU memory and that the model is running at the optimal size.

**method 5:** use map-reduce

try using parallel batch processing on multiple nodes.

1. there are tradeoffs between time and cost for Instance Selection based on Intuition and Background Knowledge we work on more training jobs; we acquire an intuition on the appropriate instances for different endeavors. In addition, the background knowledge of how a training algorithm works can help us narrow down our choices. For example, knowing that an algorithm works best on GPUs, we can focus our attention on g/p instance families. On the other hand, some algorithms can’t use GPUs for training. In such cases, choosing a GPU instance is an obvious waste of money.

I proposed this instance to start trying to work for each case:

**for 70M -- > ml.p3.2xlarge**

**for 100M -- > ml.p3.8xlarge**

**for 800M -- > ml.g4dn.8xlarge**