Part 1: share your thoughts on the following questions:

1. You are a data scientist in ABC company. Your task is to build a model using deep learning. It is about classification of 5 type of customers. your model will decide whether a customer is type 1 or type 2 ... or type 5. Each customer data is composed of 100 features that is already captured. we can supply as many number of customer information you want (e.g. 100, 10000 or more data samples). What is your strategy on requesting the number of data samples? is more and more data always good?

2. You build your model, what is your strategy on selecting the number of epochs?

3. How you train a model with unbalanced data? for example, you are given 100000 data sample regarding to Mastercard transactions where .01% of them are flagged to be fraud. What is your strategy for training your model to detect the fraud?

4. What are the available optimizers in Keras and what is their difference?

Answers:

1. For training a neural network, in general, the more data the better. However, there are a few concerns when requesting the data.
   - Time and cost. Collecting, processing more data take more human resource and computing power, and consume more time. Estimate the profit to determine the amount of time and money for collecting data.
   - Change in trend. Customer's behaviour may change according to the time
   - of the year or fashion trend. In a fast-changing environment, it may be more beneficial to update the model frequently rather than training with a lot of out-dated data.
   - Data quality. Inspecting a portion of the data is necessary to determine if they are worth collecting. For example, if the data are naturally corrupted, or whether the feature has meaningful values?
   - Type of data. Numerical data are easy to process. But if there are a number of textual features in the data, they may be hard to handle and may not be cost-effective.

2. Depending on the size of the dataset, learning rate, network structure, and other parameters, the number of epochs may vary. If the training is fast, one can put an arbitrary number, say 1000 epochs, and then adjust the value empirically. When the dataset is large and training is slow, a better alternative would be using early stopping techniques, such as monitoring the change of loss or the accuracy on validation set. Once one or multiple thresholds are met, the training can stop automatically.

3. Unbalanced data cannot be used directly for training, because the algorithm may 'cheat' by trying to predict all samples to be the majority class and achieving high accuracy. One strategy is to use **oversampling**, which resamples the original dataset to a 50:50 balanced one. During the resampling, the minority class gets sampled repeatedly, thus called oversampling. Dataset after this process is ready for regular training. Keras also provided a way to train the model with different weights on each class. **Class weights** tell the

model to ‘pay more attention’ to samples from a minority class. This method must use weighted metrics for training and evaluation.

Also, when evaluating on the whole dataset, class-specific evaluation metrics, such as **precision** and **recall**, instead of simple accuracy or loss can give a better representation of the performance.

4. Keras provides a number of optimizers to choose, here are a few popular ones:
   - SGD (Stochastic Gradient Descent) is the most basic training algorithm. each weight in the neural network is updated by multiplying its gradient and a **fixed learning rate**. The gradient is the partial derivative of the loss function with regard to each weight parameter.
   - Adagrad implement **adaptive learning rates**. They are adjusted by the frequency of a parameter gets updated. The more updates a parameter receives, the smaller the updates. This allows the model to achieve tiny adjustments when the loss curve hits a plateau. However, it still requires a base learning rate.
   - Adadelta improves on Adagrad, with a more complex mechanism to determine the learning rates. No learning rate needs to be manually set.
   - RMSprop uses **momentum**, which is maintained by recording a window of history gradients. If a parameter gets updated a lot in the recent past, it will still get a big update even though the current gradient is small. This helps the algorithm to go through local minimum. The learning rate is not auto-adjustable.
   - Adam also uses a window of history gradients but incorporates both first-order and **second-order moments**. This makes the convergence curve less ‘bouncy’ compare to other momentum-based algorithms. Adam is one of the best optimizers to start with. The learning rate is not auto-adjustable.
   - Nadam is Adam with Nesterov momentum.
   - Adamax is another variant of Adam, in some cases performs better than Adam.